Graphen und Algorithmen

Vorlesung #6: Eulersche Touren und Hamiltonsche Graphen

Dr. Armin Fügenschuh

Technische Universität Darmstadt

WS 2007/2008

Übersicht

Übersicht

* Eulersche Touren

Übersicht

* Eulersche Touren
  * Sätze von Euler und Hierholzer

Übersicht

* Eulersche Touren
  * Sätze von Euler und Hierholzer
  * Algorithmus von Fleury

Übersicht

* Eulersche Touren
  * Sätze von Euler und Hierholzer
  * Algorithmus von Fleury
  * Problem des chinesischen Briefträgers

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal
  - Satz von Grinberg

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal
  - Satz von Grinberg
  - Problem des Handlungsreisenden

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal
  - Satz von Grinberg
  - Problem des Handlungsreisenden
    - Greedy-Heuristiken

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal
  - Satz von Grinberg
  - Problem des Handlungsreisenden
    - Greedy-Heuristiken
    - Parametrisierte Greedy-Heuristiken

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal
  - Satz von Grinberg
  - Problem des Handlungsreisenden
    - Greedy-Heuristiken
    - Parametrisierte Greedy-Heuristiken
    - Untere Schranken

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal
  - Satz von Grinberg
  - Problem des Handlungsreisenden
    - Greedy-Heuristiken
    - Parametrisierte Greedy-Heuristiken
    - Untere Schranken
    - Approximationsalgorithmen

Übersicht

- Eulersche Touren
  - Sätze von Euler und Hierholzer
  - Algorithmus von Fleury
  - Problem des chinesischen Briefträgers
- Hamiltonsche Graphen
  - Sätze von Dirac und Bondy-Chvatal
  - Satz von Grinberg
  - Problem des Handlungsreisenden
    - Greedy-Heuristiken
    - Parametrisierte Greedy-Heuristiken
    - Untere Schranken
    - Approximationsalgorithmen
    - Schnittebenenverfahren

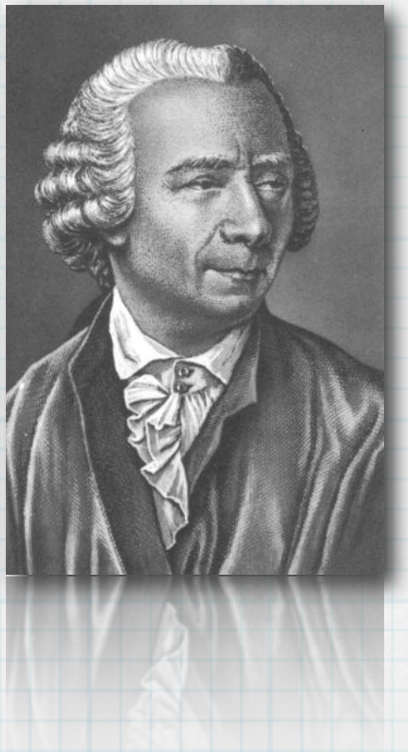
Die Wiege der Graphentheorie

Die Wiege der Graphentheorie

* Das Königsberger Brückenproblem (Leonhard Euler, 1707–1783)

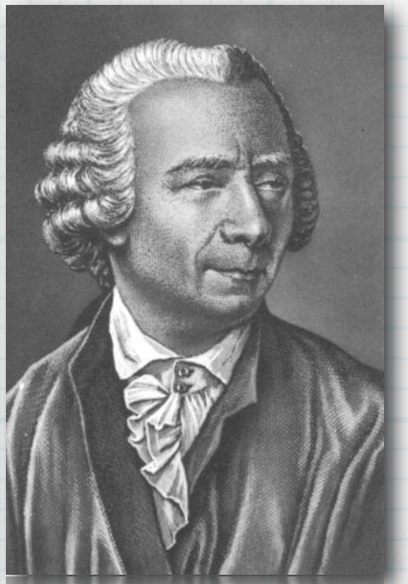
Die Wiege der Graphentheorie

* Das Königsberger Brückenproblem (Leonhard Euler, 1707–1783)



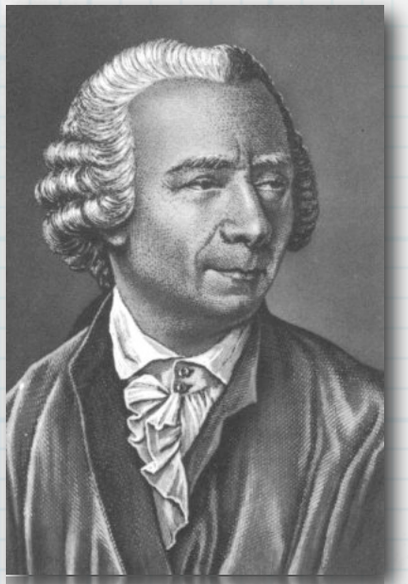
Die Wiege der Graphentheorie

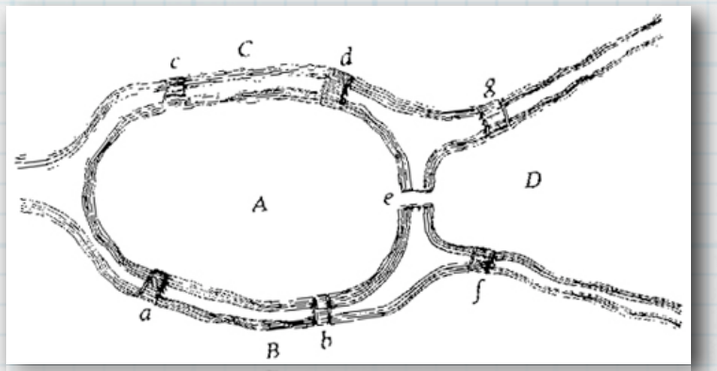
* Das Königsberger Brückenproblem (Leonhard Euler, 1707–1783)



Die Wiege der Graphentheorie

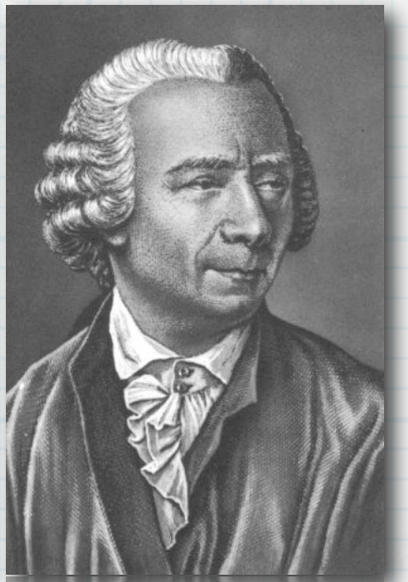
* Das Königsberger Brückenproblem (Leonhard Euler, 1707–1783)

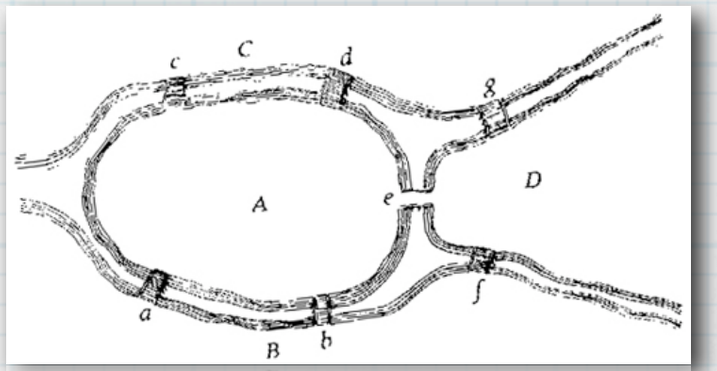




Die Wiege der Graphentheorie

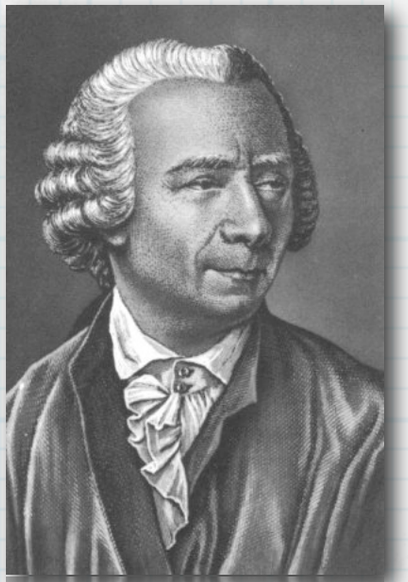
* Das Königsberger Brückenproblem (Leonhard Euler, 1707–1783)
* Gibt es einen Weg über alle Pregel-Brücken, so dass jede Brücke genau einmal benutzt wird?

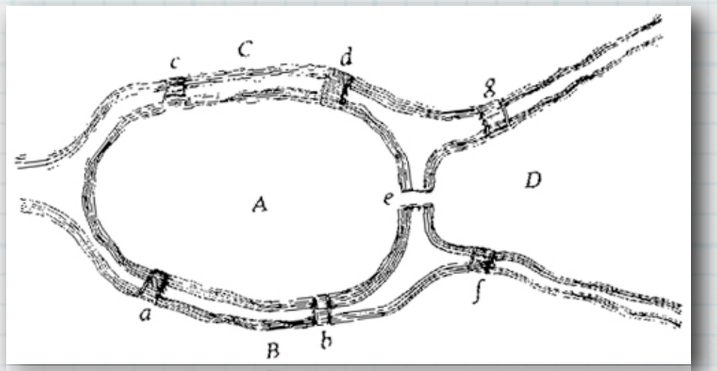




Die Wiege der Graphentheorie

* Das Königsberger Brückenproblem (Leonhard Euler, 1707–1783)
* Gibt es einen Weg über alle Pregel-Brücken, so dass jede Brücke genau einmal benutzt wird?
* ...und man am Ende der Wanderung wieder am Ausgangort ankommt?





Eulersche Graphen

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.

Eulersche Graphen

- **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
- Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
- **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
- **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
- Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

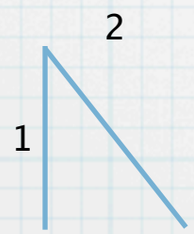
1

Das

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

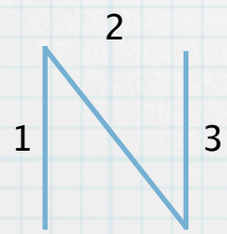


Das ist

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

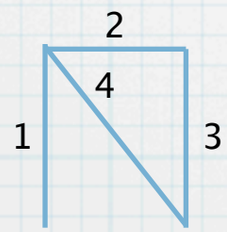


Das ist das

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

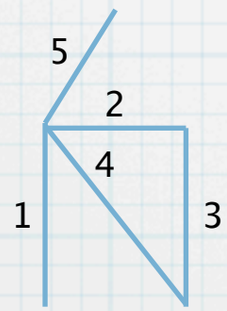


Das ist das Haus

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

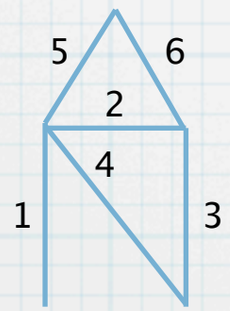


Das ist das Haus des

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

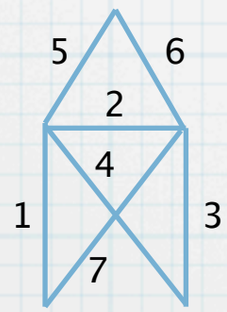


Das ist das Haus des Ni-

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

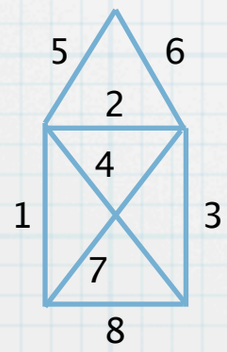


Das ist das Haus des Ni-ko-

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

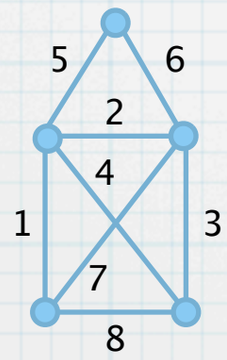


Das ist das Haus des Ni-ko-laus!

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)

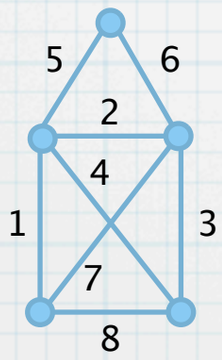


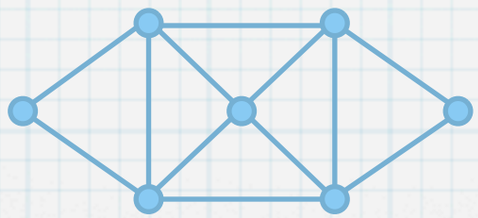
Das ist das Haus des Ni-ko-laus!

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)



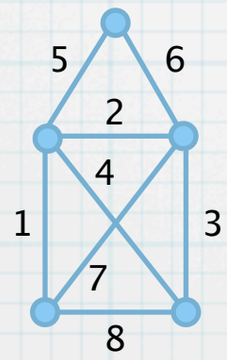


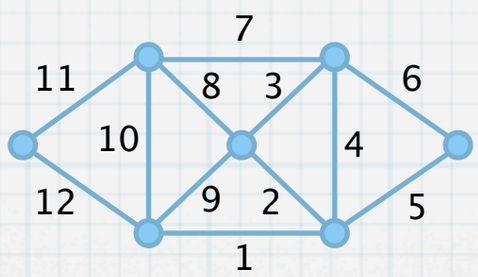
Das ist das Haus des Ni-ko-laus!

Eulersche Graphen

* **Definition 1**:
  Sei $G = (V, E)$ ein Graph. Ein Weg in G heißt **Eulerscher Weg**, wenn er jede Kante von G enthält.
* Bemerkung: Da in einem Weg die Kanten per Definition paarweise verschieden sind, enthält ein Eulerscher Weg damit jede Kante genau einmal.
* **Definition 2**:
  Eine **Tour** von G ist ein geschlossener Kantenzug, die jede Kante von G mindestens einmal enthält.

  Eine **Eulersche Tour** von G ist eine Tour, die jede Kante von G genau einmal enthält.
* **Definition 3**:
  Ein Graph G heißt **Eulersch**, wenn er eine Eulersche Tour enthält.
* Beispiel: Eulerscher Weg (links), Eulerscher Graph (rechts)





Das ist das Haus des Ni-ko-laus!

Eine notwendige Bedingung für Eulersche Graphen

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

  Also ist $\deg(v)$ gerade.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

  Also ist $\deg(v)$ gerade.

  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

  Also ist $\deg(v)$ gerade.

  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

  Also ist $\deg(v)$ gerade.

  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

  Also ist auch $\deg(u)$ gerade.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

  Also ist $\deg(v)$ gerade.

  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

  Also ist auch $\deg(u)$ gerade.

  Somit haben alle Knoten, wie behauptet, einen geraden Grad.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

  Also ist $\deg(v)$ gerade.

  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

  Also ist auch $\deg(u)$ gerade.

  Somit haben alle Knoten, wie behauptet, einen geraden Grad.
* Folgerung: Das Haus-des-Nikolaus und die Brücken von Königsberg sind nicht-Eulersch.

Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

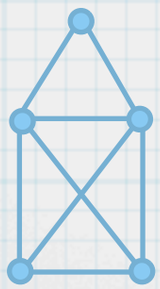
  Also ist $\deg(v)$ gerade.

  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

  Also ist auch $\deg(u)$ gerade.

  Somit haben alle Knoten, wie behauptet, einen geraden Grad.
* Folgerung: Das Haus-des-Nikolaus und die Brücken von Königsberg sind nicht-Eulersch.



Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

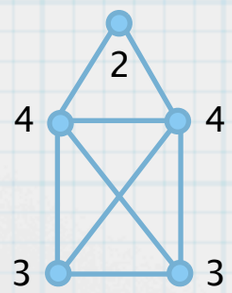
  Also ist $\deg(v)$ gerade.

  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

  Also ist auch $\deg(u)$ gerade.

  Somit haben alle Knoten, wie behauptet, einen geraden Grad.
* Folgerung: Das Haus-des-Nikolaus und die Brücken von Königsberg sind nicht-Eulersch.



Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

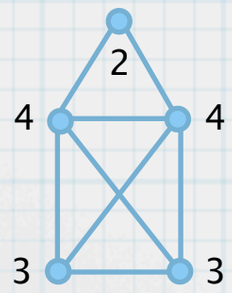
  Also ist $\deg(v)$ gerade.

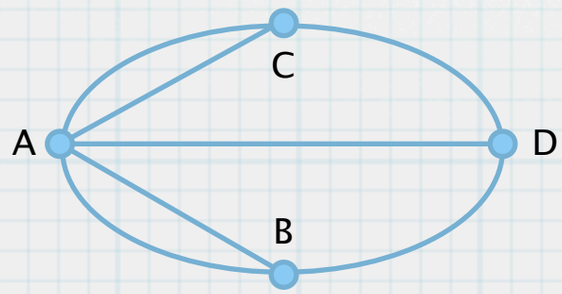
  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

  Also ist auch $\deg(u)$ gerade.

  Somit haben alle Knoten, wie behauptet, einen geraden Grad.
* Folgerung: Das Haus-des-Nikolaus und die Brücken von Königsberg sind nicht-Eulersch.





Eine notwendige Bedingung für Eulersche Graphen

* **Satz 4** (Euler, 1736):
  Sei G ein Eulerscher Graph. Dann ist der Grad jedes Knotens gerade.
* Beweis:

  Sei G ein Eulerscher Graph und C eine Eulersche Tour in G, die bei einem Knoten u startet und endet.

  Sei v ein von u verschiedener Knoten.

  Da G zusammenhängend ist und die Tour jede Kante von G einschließt, ist auch v ein Knoten der Tour.

  v wird jedes Mal, wenn er auf der Tour C durchlaufen wird, von unterschiedlichen Kanten erreicht und verlassen, da jede Kante nur einmal in der Tour vorkommt.

  Somit wird der Grad des Knotens v jedes Mal um 2 erhöht.

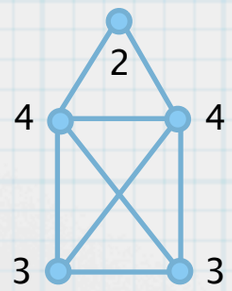
  Also ist $\deg(v)$ gerade.

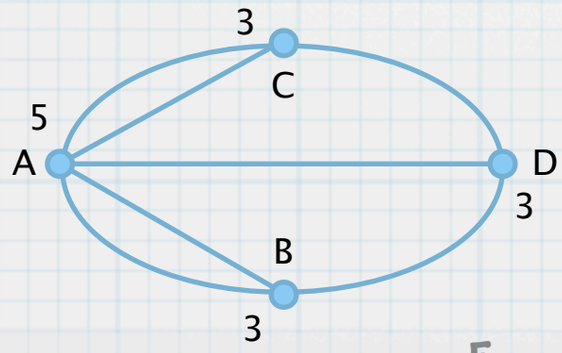
  Da die Tour bei u beginnt und endet (mit zwei unterschiedlichen Kanten), tragen die erste und letzte Kante 2 zu $\deg(u)$ bei.

  Wie für v gilt auch für u, dass jede Wdh. von u innerhalb der Tour den Grad um 2 erhöht.

  Also ist auch $\deg(u)$ gerade.

  Somit haben alle Knoten, wie behauptet, einen geraden Grad.
* Folgerung: Das Haus-des-Nikolaus und die Brücken von Königsberg sind nicht-Eulersch.





Ein einleitendes Lemma vorweg

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

  Da $\deg(v_0) \geq 2$, gibt es eine Kante e_1 mit Endknoten v_0 und einem weiteren mit v_1 bezeichneten Endknoten.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

  Da $\deg(v_0) \geq 2$, gibt es eine Kante e_1 mit Endknoten v_0 und einem weiteren mit v_1 bezeichneten Endknoten.

  Da auch $\deg(v_1) \geq 2$, gibt es eine Kante e_2 mit Endknoten v_1 und v_2, wobei $v_2 \neq v_0$.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

  Da $\deg(v_0) \geq 2$, gibt es eine Kante e_1 mit Endknoten v_0 und einem weiteren mit v_1 bezeichneten Endknoten.

  Da auch $\deg(v_1) \geq 2$, gibt es eine Kante e_2 mit Endknoten v_1 und v_2, wobei $v_2 \neq v_0$.

  Allgemein wird beim $(i+1)$–ten Mal eine Kante $e_i = \{v_i, v_{i+1}\}$ gewählt, wobei $v_{i+1} \neq v_{i-1}$.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

  Da $\deg(v_0) \geq 2$, gibt es eine Kante e_1 mit Endknoten v_0 und einem weiteren mit v_1 bezeichneten Endknoten.

  Da auch $\deg(v_1) \geq 2$, gibt es eine Kante e_2 mit Endknoten v_1 und v_2, wobei $v_2 \neq v_0$.

  Allgemein wird beim $(i+1)$–ten Mal eine Kante $e_i = \{v_i, v_{i+1}\}$ gewählt, wobei $v_{i+1} \neq v_{i-1}$.

  Da G nur endlich viele Knoten hat, müssen wir irgendwann einen Knoten wählen, der bereits vorher durchlaufen wurde.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

  Da $\deg(v_0) \geq 2$, gibt es eine Kante e_1 mit Endknoten v_0 und einem weiteren mit v_1 bezeichneten Endknoten.

  Da auch $\deg(v_1) \geq 2$, gibt es eine Kante e_2 mit Endknoten v_1 und v_2, wobei $v_2 \neq v_0$.

  Allgemein wird beim $(i+1)$–ten Mal eine Kante $e_i = \{v_i, v_{i+1}\}$ gewählt, wobei $v_{i+1} \neq v_{i-1}$.

  Da G nur endlich viele Knoten hat, müssen wir irgendwann einen Knoten wählen, der bereits vorher durchlaufen wurde.

  So lange wird das Verfahren wiederholt.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

  Da $\deg(v_0) \geq 2$, gibt es eine Kante e_1 mit Endknoten v_0 und einem weiteren mit v_1 bezeichneten Endknoten.

  Da auch $\deg(v_1) \geq 2$, gibt es eine Kante e_2 mit Endknoten v_1 und v_2, wobei $v_2 \neq v_0$.

  Allgemein wird beim $(i+1)$–ten Mal eine Kante $e_i = \{v_i, v_{i+1}\}$ gewählt, wobei $v_{i+1} \neq v_{i-1}$.

  Da G nur endlich viele Knoten hat, müssen wir irgendwann einen Knoten wählen, der bereits vorher durchlaufen wurde.

  So lange wird das Verfahren wiederholt.

  Sei v_k der erste derartige Knoten.

Ein einleitendes Lemma vorweg

* **Lemma 5**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

  Sei v_0 ein beliebiger Knoten von G.

  Da $\deg(v_0) \geq 2$, gibt es eine Kante e_1 mit Endknoten v_0 und einem weiteren mit v_1 bezeichneten Endknoten.

  Da auch $\deg(v_1) \geq 2$, gibt es eine Kante e_2 mit Endknoten v_1 und v_2, wobei $v_2 \neq v_0$.

  Allgemein wird beim $(i+1)$–ten Mal eine Kante $e_i = \{v_i, v_{i+1}\}$ gewählt, wobei $v_{i+1} \neq v_{i-1}$.

  Da G nur endlich viele Knoten hat, müssen wir irgendwann einen Knoten wählen, der bereits vorher durchlaufen wurde.

  So lange wird das Verfahren wiederholt.

  Sei v_k der erste derartige Knoten.

  Dann ist $(v_k, v_{k+1}, \ldots, v_{k+s+1}, v_k)$ der gesuchte Kreis.

Chrakterisierung Eulerscher Graphen

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

  Konstruiere eine Eulertour für G wie folgt:

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

  Konstruiere eine Eulertour für G wie folgt:

    Wir starten mit einem beliebigen Knoten in C.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

  Konstruiere eine Eulertour für G wie folgt:

    Wir starten mit einem beliebigen Knoten in C.

    Laufe so lange entlang der Kanten in C, bis man auf eine Komponente von H stößt.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

  Konstruiere eine Eulertour für G wie folgt:

    Wir starten mit einem beliebigen Knoten in C.

    Laufe so lange entlang der Kanten in C, bis man auf eine Komponente von H stößt.

    Laufe nun entlang der Eulertour in der Komponente.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

  Konstruiere eine Eulertour für G wie folgt:

    Wir starten mit einem beliebigen Knoten in C.

    Laufe so lange entlang der Kanten in C, bis man auf eine Komponente von H stößt.

    Laufe nun entlang der Eulertour in der Komponente.

    Am Ende sind wir wieder dort, wo wir C verlassen haben.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

  Konstruiere eine Eulertour für G wie folgt:

    Wir starten mit einem beliebigen Knoten in C.

    Laufe so lange entlang der Kanten in C, bis man auf eine Komponente von H stößt.

    Laufe nun entlang der Eulertour in der Komponente.

    Am Ende sind wir wieder dort, wo wir C verlassen haben.

    Setze dieses fort, bis alle Wege in C und alle Eulertouren in allen Komponenten von H abgelaufen sind.

Chrakterisierung Eulerscher Graphen

* **Satz 6** (Hierholzer, 1873):
  Ein zusammenhängender Graph ist genau dann Eulersch, wenn der Grad jedes Knotens gerade ist.
* Beweis (Induktion über die Kantenanzahl):

  Induktionsanfang, keine Kanten: Ein Graph mit einem einzigen Knoten ist Eulersch.

  Wenn der Graph $G = (V, E)$ Kanten hat, dann gibt es keine isolierten Knoten mit Grad 0, da der Graph zusammenhängend ist.

  Da der Knotengrad gerade ist, ist daher $\deg(v) \geq 2$ für alle $v \in V$.

  Nach Lemma 5 gibt es dann in G einen Kreis C.

  Enthält C bereits jede Kante von G, dann haben wir damit die gesuchte Eulertour.

  Andernfalls sei $H := (V, E \backslash E(C))$. H ist u.U. nicht zusammenhängend.

  Die Knotengrade in H sind immer noch gerade, da der Grad aller Knoten in C um genau 2 reduziert wurde (gegenüber dem jeweiligen Knotengrad in G).

  Nach Induktionsannahme ist jede Zusammenhangskomponente von H Eulersch.

  Außerdem hat jede Komponente mind. einen Knoten mit C gemein.

  Konstruiere eine Eulertour für G wie folgt:

    Wir starten mit einem beliebigen Knoten in C.

    Laufe so lange entlang der Kanten in C, bis man auf eine Komponente von H stößt.

    Laufe nun entlang der Eulertour in der Komponente.

    Am Ende sind wir wieder dort, wo wir C verlassen haben.

    Setze dieses fort, bis alle Wege in C und alle Eulertouren in allen Komponenten von H abgelaufen sind.

  Da die Kanten von G genau die Kanten von H und C sind, erhalten wir eine Eulertour in G.

Charakterisierung von Graphen mit Eulerschen Wegen

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).
  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  1. Fall, es gibt keine Kante $e = \{u, v\}$ zwischen u und v.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  1. Fall, es gibt keine Kante $e = \{u, v\}$ zwischen u und v.

     Wir verbinden u und v mit einer Kante $e = \{u, v\}$.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  1. Fall, es gibt keine Kante $e = \{u, v\}$ zwischen u und v.

     Wir verbinden u und v mit einer Kante $e = \{u, v\}$.

     In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  1. Fall, es gibt keine Kante $e = \{u, v\}$ zwischen u und v.

     Wir verbinden u und v mit einer Kante $e = \{u, v\}$.

     In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

     Jetzt haben alle Knoten einen geraden Grad.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  1. Fall, es gibt keine Kante $e = \{u, v\}$ zwischen u und v.

     Wir verbinden u und v mit einer Kante $e = \{u, v\}$.

     In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

     Jetzt haben alle Knoten einen geraden Grad.

     Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  1. Fall, es gibt keine Kante $e = \{u, v\}$ zwischen u und v.

     Wir verbinden u und v mit einer Kante $e = \{u, v\}$.

     In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

     Jetzt haben alle Knoten einen geraden Grad.

     Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.

     O.B.d.A. startet und endet diese Tour in u, so dass e die letzte Kante ist.

Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).
  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.
  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  1. Fall, es gibt keine Kante $e = \{u, v\}$ zwischen u und v.

     Wir verbinden u und v mit einer Kante $e = \{u, v\}$.
     In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).
     Jetzt haben alle Knoten einen geraden Grad.
     Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.
     O.B.d.A. startet und endet diese Tour in u, so dass e die letzte Kante ist.
     Lösche e aus der Tour und erhalte einen Eulerweg für G mit den Endknoten u, v.

Charakterisierung (Forts.)

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

     Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

  Füge danach Kante e in die Tour ein, so entsteht ein Eulerweg für G.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

  Füge danach Kante e in die Tour ein, so entsteht ein Eulerweg für G.

  2. Fall, der Graph $G - e$ ist nicht mehr zusammenhängend.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

  Füge danach Kante e in die Tour ein, so entsteht ein Eulerweg für G.

  2. Fall, der Graph $G - e$ ist nicht mehr zusammenhängend.

  Dann gibt es in jeder der beiden Zusammenhangskomponenten Eulertouren.

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

  Füge danach Kante e in die Tour ein, so entsteht ein Eulerweg für G.

  2. Fall, der Graph $G - e$ ist nicht mehr zusammenhängend.

  Dann gibt es in jeder der beiden Zusammenhangskomponenten Eulertouren.

  Der Eulerweg für G besteht dann aus

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

  Füge danach Kante e in die Tour ein, so entsteht ein Eulerweg für G.

  2. Fall, der Graph $G - e$ ist nicht mehr zusammenhängend.

  Dann gibt es in jeder der beiden Zusammenhangskomponenten Eulertouren.

  Der Eulerweg für G besteht dann aus

  der Eulertour der einen Zusammenhangskomponente, welche bei u startet und endet,

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

  Füge danach Kante e in die Tour ein, so entsteht ein Eulerweg für G.

  2. Fall, der Graph $G - e$ ist nicht mehr zusammenhängend.

  Dann gibt es in jeder der beiden Zusammenhangskomponenten Eulertouren.

  Der Eulerweg für G besteht dann aus

  der Eulertour der einen Zusammenhangskomponente, welche bei u startet und endet,

  der Kante e von u nach v,

Charakterisierung (Forts.)

* **Satz 7**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis (Forts.):

  2. Fall, es gibt eine Kante $e = \{u, v\}$ zwischen u und v.

  Lösche diese Kante. Dann sind alle Knoten in $G - e$ gerade.

  1. Fall, der Graph $G - e$ ist immer noch zusammenhängend.

  Es gibt eine Eulertour in $G - e$, die o.B.d.A. bei u startet und endet.

  Füge danach Kante e in die Tour ein, so entsteht ein Eulerweg für G.

  2. Fall, der Graph $G - e$ ist nicht mehr zusammenhängend.

  Dann gibt es in jeder der beiden Zusammenhangskomponenten Eulertouren.

  Der Eulerweg für G besteht dann aus

  der Eulertour der einen Zusammenhangskomponente, welche bei u startet und endet,

  der Kante e von u nach v,

  und der Eulertour der anderen Zusammenhangskomponente, welche bei v startet und endet.

Allgemeinere Definition eines Graphen

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**

Allgemeinere Definition eines Graphen

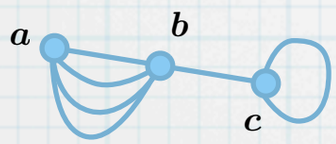
* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:

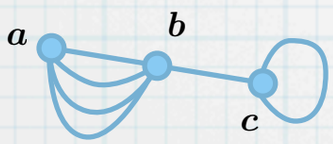
Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:



Allgemeinere Definition eines Graphen

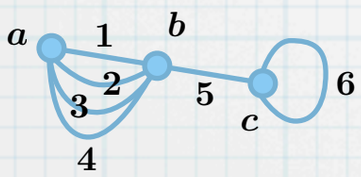
* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:



$$V = \{a, b, c\}$$

Allgemeinere Definition eines Graphen

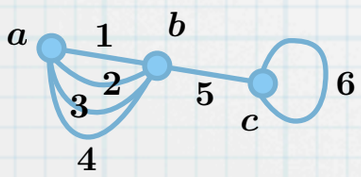
* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:



$$V = \{a, b, c\}$$

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:

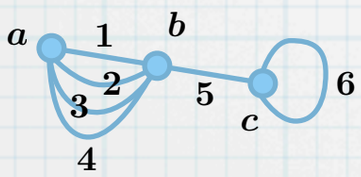


$$V = \{a, b, c\}$$

$$E = \{1, 2, 3, 4, 5, 6\}$$

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:



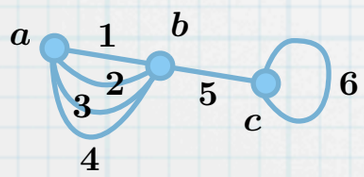
$$V = \{a, b, c\}$$

$$E = \{1, 2, 3, 4, 5, 6\}$$

$$\phi : \begin{cases} 1, 2, 3, 4 & \mapsto & \{a, b\} \\ 5 & \mapsto & \{b, c\} \\ 6 & \mapsto & \{c\} \end{cases}$$

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:



$$V = \{a, b, c\}$$

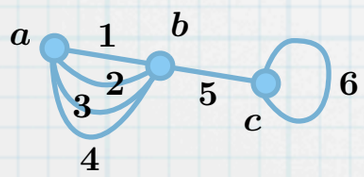
$$E = \{1, 2, 3, 4, 5, 6\}$$

$$\phi : \begin{cases} 1, 2, 3, 4 & \mapsto & \{a, b\} \\ 5 & \mapsto & \{b, c\} \\ 6 & \mapsto & \{c\} \end{cases}$$

* Bemerkungen:

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:



$$V = \{a, b, c\}$$

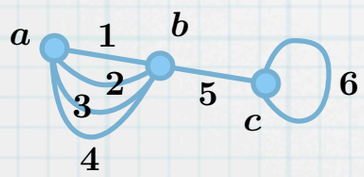
$$E = \{1, 2, 3, 4, 5, 6\}$$

$$\phi : \begin{cases} 1, 2, 3, 4 & \mapsto & \{a, b\} \\ 5 & \mapsto & \{b, c\} \\ 6 & \mapsto & \{c\} \end{cases}$$

* Bemerkungen:
  * Die Abbildung ϕ wird oftmals nicht aufgeschrieben. Auch für Multigraphen schreiben wir weiterhin kurz $G = (V, E)$.

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten.**
* Beispiel:



$$V = \{a, b, c\}$$

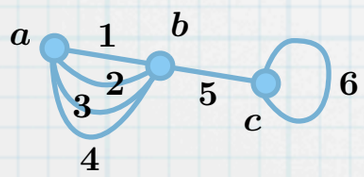
$$E = \{1, 2, 3, 4, 5, 6\}$$

$$\phi : \begin{cases} 1, 2, 3, 4 & \mapsto & \{a, b\} \\ 5 & \mapsto & \{b, c\} \\ 6 & \mapsto & \{c\} \end{cases}$$

* Bemerkungen:
  * Die Abbildung ϕ wird oftmals nicht aufgeschrieben. Auch für Multigraphen schreiben wir weiterhin kurz $G = (V, E)$.
  * Eine analoge Definition gibt es für Digraphen mit gerichteten Mehrfachbögen und Schlingen.

Allgemeinere Definition eines Graphen

* Bisher haben wir Graphen (in der 1. Vorlesung) so definiert, dass Schlingen und Mehrfachkanten ausgeschlossen waren. Wir geben nun eine allgemeinere Definition an, die solches gestattet.
* **Definition 8**:
  Sei V eine endliche, nicht-leere Menge, E eine endliche (möglicherweise leere) Teilmenge der Menge der natürlichen Zahlen und $\phi : E \to (\mathcal{P}_1 \cup \mathcal{P}_2)$ eine Abbildung von E in die Menge der ein- oder zweielementigen Teilmengen von V. Dann heißt das Tripel $G = (V, E, \phi)$ **ungerichteter Multigraph** (oder auch nur **Graph**). Die Elemente von V heißen **Knoten**, **Ecken** oder **Punkte**, die Elemente von E **Kanten**. Für die Knoten- bzw. Kantenmenge eines Graphen schreiben wir auch $V(G)$ bzw. $E(G)$. Ist $\phi(e) \in \mathcal{P}_1$, so wird e auch als **Schlinge** bezeichnet. Gibt es $e, e' \in E$ mit $\phi(e) = \phi(e')$, so heißen e, e' **Mehrfachschlingen** bzw. **Mehrfachkanten**.
* Beispiel:



$$V = \{a, b, c\}$$

$$E = \{1, 2, 3, 4, 5, 6\}$$

$$\phi : \begin{cases} 1, 2, 3, 4 & \mapsto & \{a, b\} \\ 5 & \mapsto & \{b, c\} \\ 6 & \mapsto & \{c\} \end{cases}$$

* Bemerkungen:
  * Die Abbildung ϕ wird oftmals nicht aufgeschrieben. Auch für Multigraphen schreiben wir weiterhin kurz $G = (V, E)$.
  * Eine analoge Definition gibt es für Digraphen mit gerichteten Mehrfachbögen und Schlingen.
  * Bisher gezeigte Sätze (wie z.B. der Satz von Euler oder Hierholzer) gelten auch für Multigraphen. Die Beweise werde mal aufwändiger, mal einfacher.

Nochmals: Ein einleitendes Lemma vorweg

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:
  1. Fall, G ist nicht schlicht.

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:
  1. Fall, G ist nicht schlicht.
     Dann ist jede Schlinge in G ein Kreis der Länge 1.

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:
  1. Fall, G ist nicht schlicht.
     Dann ist jede Schlinge in G ein Kreis der Länge 1.
     Und jede Mehrfachkante in G ist ein Kreis der Länge 2.

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:
  1. Fall, G ist nicht schlicht.
     Dann ist jede Schlinge in G ein Kreis der Länge 1.
     Und jede Mehrfachkante in G ist ein Kreis der Länge 2.
     Also hat G einen Kreis.

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:
  1. Fall, G ist nicht schlicht.
     Dann ist jede Schlinge in G ein Kreis der Länge 1.
     Und jede Mehrfachkante in G ist ein Kreis der Länge 2.
     Also hat G einen Kreis.
  2. Fall, G ist schlicht.

Nochmals: Ein einleitendes Lemma vorweg

* **Lemma 5‘**:
  Sei G ein Graph, in dem der Grad jedes Knotens mindestens 2 beträgt. Dann enthält G einen Kreis.
* Beweis:
  1. Fall, G ist nicht schlicht.
     Dann ist jede Schlinge in G ein Kreis der Länge 1.
     Und jede Mehrfachkante in G ist ein Kreis der Länge 2.
     Also hat G einen Kreis.
  2. Fall, G ist schlicht.
     Siehe Beweis von Lemma 5.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7'**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).

  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).

  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

  Jetzt haben alle Knoten einen geraden Grad.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).

  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

  Jetzt haben alle Knoten einen geraden Grad.

  Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).

  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

  Jetzt haben alle Knoten einen geraden Grad.

  Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.

  O.B.d.A. startet und endet diese Tour in u, so dass e die letzte Kante ist.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).

  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

  Jetzt haben alle Knoten einen geraden Grad.

  Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.

  O.B.d.A. startet und endet diese Tour in u, so dass e die letzte Kante ist.

  Lösche e aus der Tour und erhalte einen Eulerweg für G mit den Endknoten u, v.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:
  ($\Rightarrow$): G habe einen Eulerschen Weg.
  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).
  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.
  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.
  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.
  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.
  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).
  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).
  Jetzt haben alle Knoten einen geraden Grad.
  Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.
  O.B.d.A. startet und endet diese Tour in u, so dass e die letzte Kante ist.
  Lösche e aus der Tour und erhalte einen Eulerweg für G mit den Endknoten u, v.
* Folgerung: Im Königsberger Brückenproblem gibt es keinen Eulerweg.

Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:

  ($\Rightarrow$): G habe einen Eulerschen Weg.

  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).

  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.

  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.

  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.

  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.

  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).

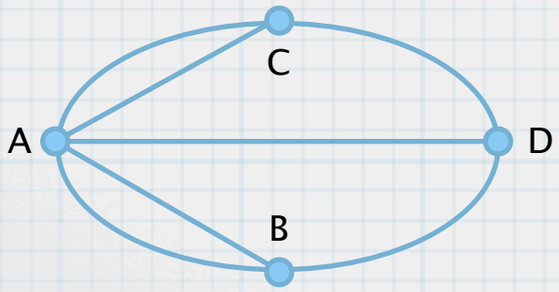
  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).

  Jetzt haben alle Knoten einen geraden Grad.

  Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.

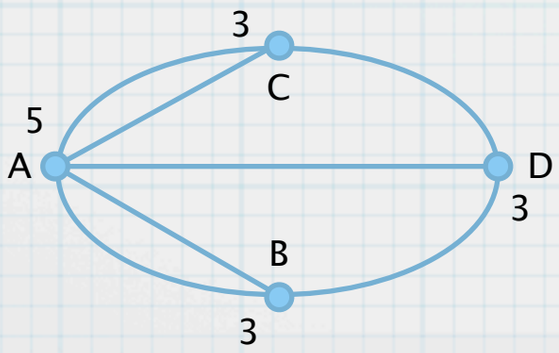
  O.B.d.A. startet und endet diese Tour in u, so dass e die letzte Kante ist.

  Lösche e aus der Tour und erhalte einen Eulerweg für G mit den Endknoten u, v.
* Folgerung: Im Königsberger Brückenproblem gibt es keinen Eulerweg.



Nochmals: Charakterisierung von Graphen mit Eulerschen Wegen

* **Satz 7‘**:
  Ein zusammenhängender Graph G hat genau dann einen Eulerschen Weg, wenn er höchstens zwei ungerade Knoten enthält.
* Beweis:
  ($\Rightarrow$): G habe einen Eulerschen Weg.
  Wenn v kein Endknoten des Weges ist, dann ist $\deg(v)$ gerade (vgl. Beweis von Satz 4).
  Somit sind die einzig möglichen Knoten mit ungeradem Grad die beiden Endknoten.
  ($\Leftarrow$): Sei G zusammenhängend mit höchstens zwei ungeraden Knoten.
  Wenn G keinen ungeraden Knoten hat, dann gibt es nach Satz 6 sogar eine Eulertour.
  G kann nicht genau einen ungeraden Knoten haben, da die Anzahl ungerader Knoten stets gerade ist. Bleibt der Fall, dass G genau zwei ungerade Knoten u und v hat.
  Wir verbinden u und v mit einer Kante $e = \{u, v\}$ (**auch wenn da schon eine ist!**).
  In $G + e$ hat sich der Grad von u und v damit um 1 erhöht (gegenüber G).
  Jetzt haben alle Knoten einen geraden Grad.
  Es gibt dann eine Eulertour, die jede Kante von $G + e$ genau ein Mal einbezieht.
  O.B.d.A. startet und endet diese Tour in u, so dass e die letzte Kante ist.
  Lösche e aus der Tour und erhalte einen Eulerweg für G mit den Endknoten u, v.
* Folgerung: Im Königsberger Brückenproblem gibt es keinen Eulerweg.

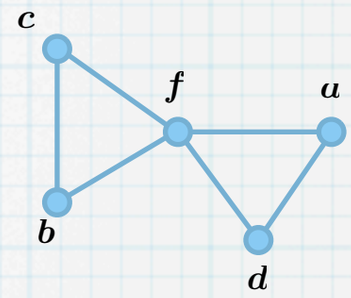


Algorithmus von Fleury

Algorithmus von Fleury

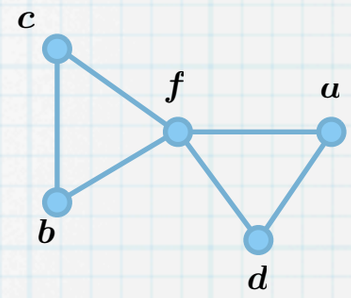
* Eingabe: Eulerscher Graph $G = (V, E)$

Algorithmus von Fleury



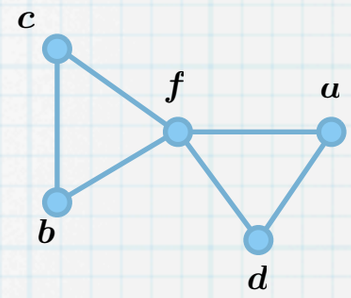
* Eingabe: Eulerscher Graph $G = (V, E)$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

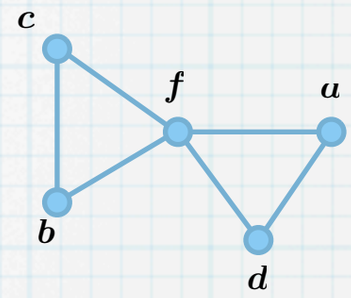
Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury

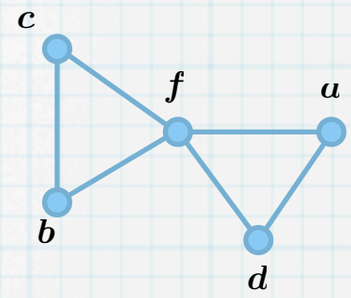
Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury

Algorithmus von Fleury

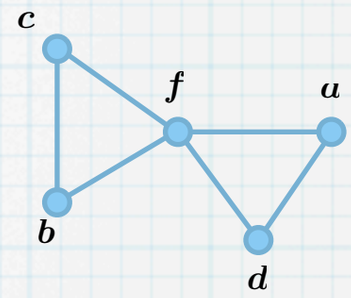


* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury

(2) $m := |E|, v_0 \in V$

Algorithmus von Fleury

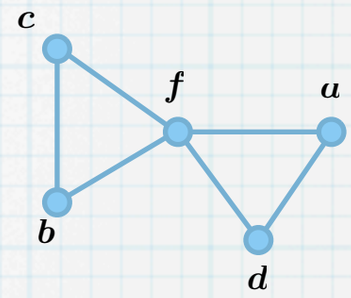


* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury

(2) $m := |E|, v_0 \in V$

Algorithmus von Fleury



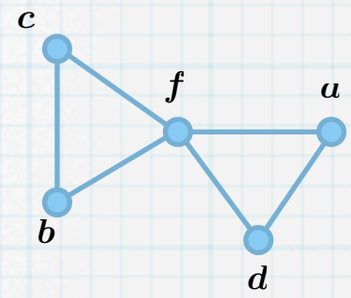
* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury

(2) $m := |E|, v_0 \in V$

$m = 6, v_0 = a$

Algorithmus von Fleury



- Eingabe: Eulerscher Graph $G = (V, E)$
- Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

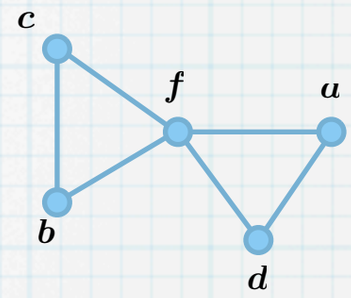
(1) **algorithm** fleury

(2) $m := |E|, v_0 \in V$

(3) **for** i **from** 1 **to** m **do**

$m = 6, v_0 = a$

Algorithmus von Fleury

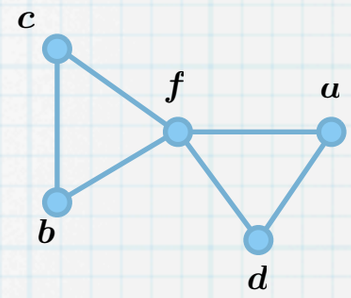


- Eingabe: Eulerscher Graph $G = (V, E)$
- Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**

$m = 6, v_0 = a$

Algorithmus von Fleury



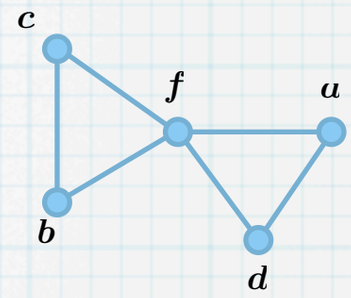
* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**

$m = 6, v_0 = a$

$i = 1$

Algorithmus von Fleury



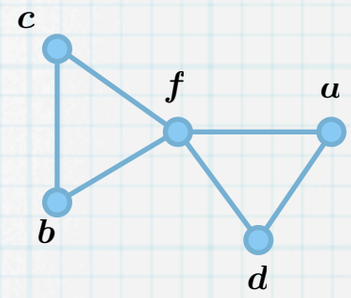
* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$

$m = 6, v_0 = a$

$i = 1$

Algorithmus von Fleury



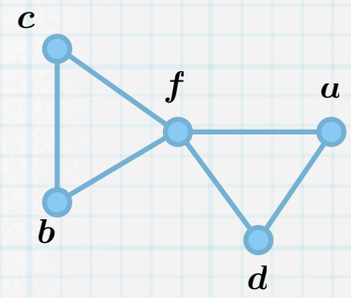
* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$

$m = 6, v_0 = a$

$i = 1$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

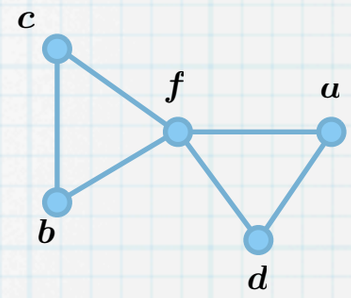
(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$

$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

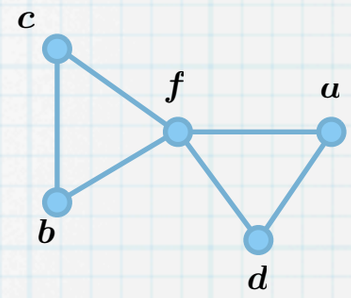
(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**

$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

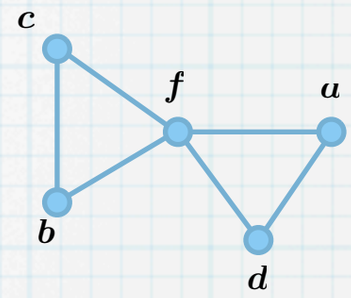
(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**

$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

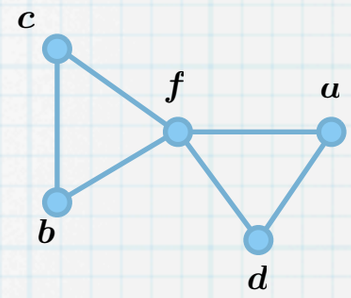
(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$

$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

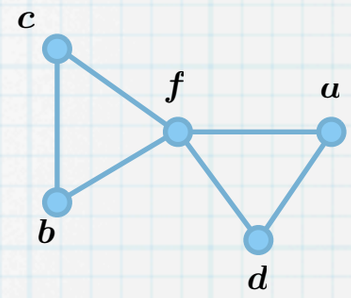
(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$

$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$

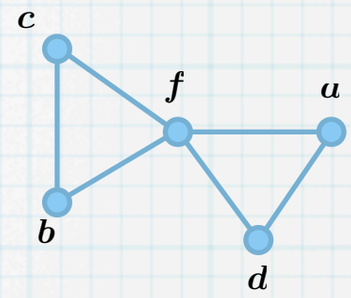
$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

$e = ad$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$

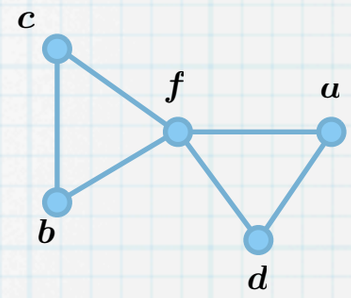
$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

$e = ad$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$

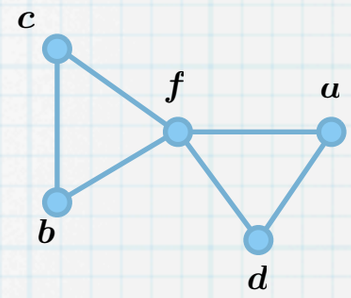
$m = 6, v_0 = a$

$i = 1$

$Z = X = \{af, ad\}$

$e = ad$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$

$m = 6, v_0 = a$

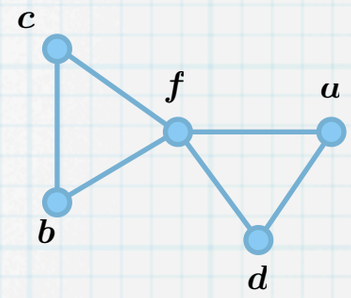
$i = 1$

$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$

Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$

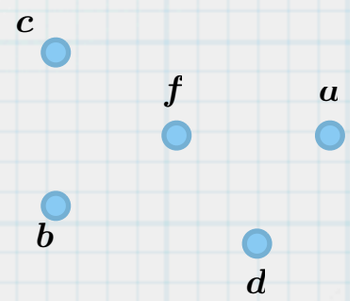
$m = 6, v_0 = a$

$i = 1$

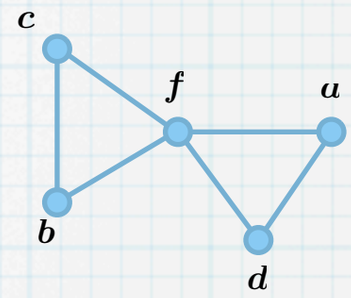
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$

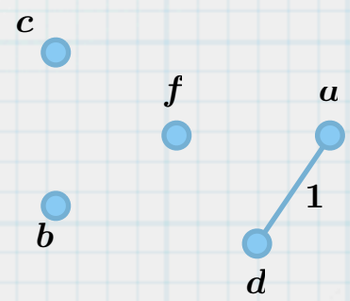
$m = 6, v_0 = a$

$i = 1$

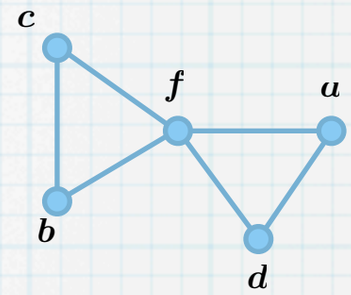
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**

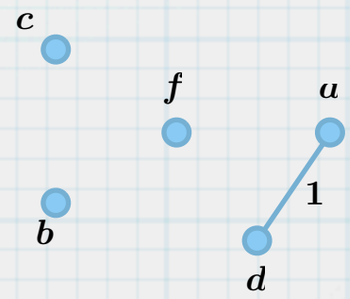
$m = 6, v_0 = a$

$i = 1$

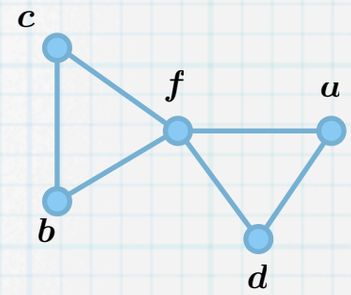
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



- Eingabe: Eulerscher Graph $G = (V, E)$
- Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$

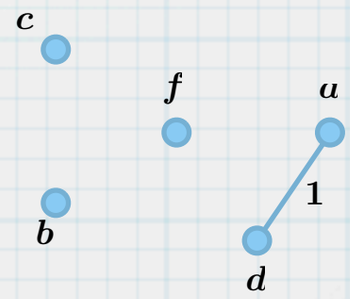
$m = 6, v_0 = a$

$i = 1$

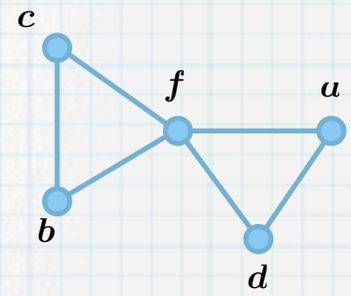
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$

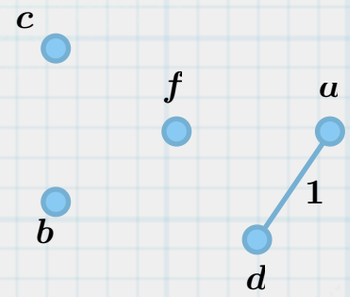
$m = 6, v_0 = a$

$i = 1$

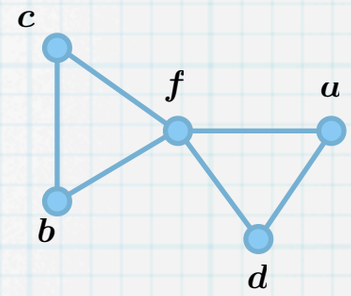
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**

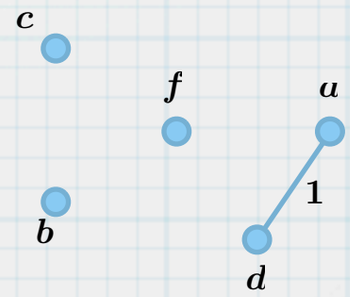
$m = 6, v_0 = a$

$i = 1$

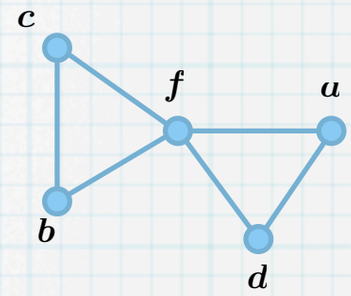
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**

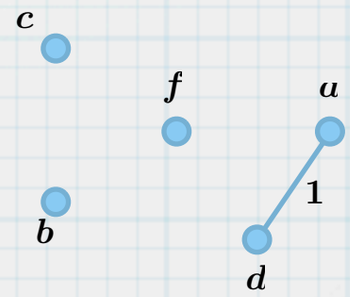
$m = 6, v_0 = a$

$i = 1$

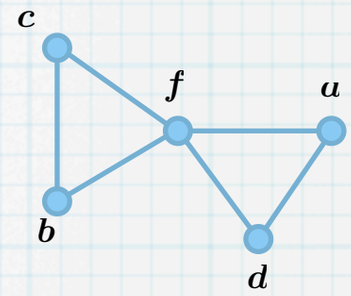
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$

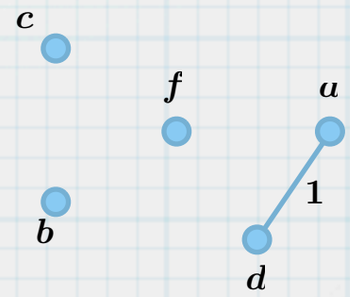
$m = 6, v_0 = a$

$i = 1$

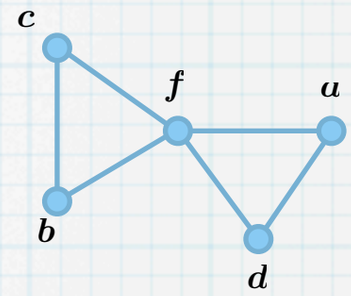
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$

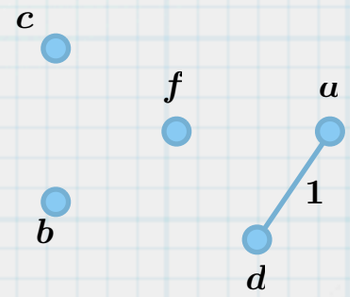
$m = 6, v_0 = a$

$i = 1$

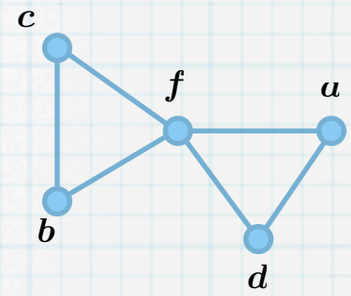
$Z = X = \{af, ad\}$

$e = ad$

$e_1 = ad$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$

$m = 6, v_0 = a$

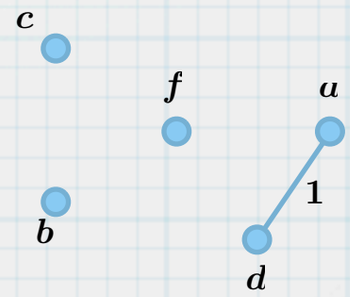
$i = 1$

$Z = X = \{af, ad\}$

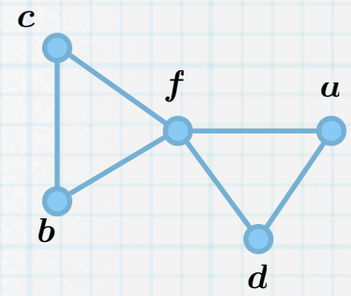
$e = ad$

$e_1 = ad$

$v_1 = d$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$

$m = 6, v_0 = a$

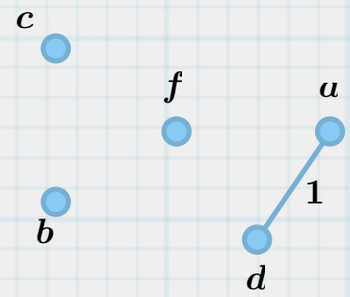
$i = 1$

$Z = X = \{af, ad\}$

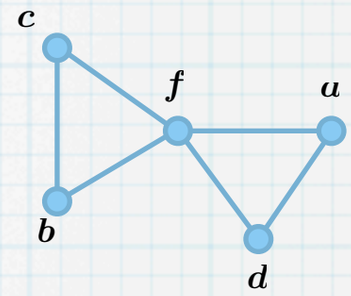
$e = ad$

$e_1 = ad$

$v_1 = d$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$

$m = 6, v_0 = a$

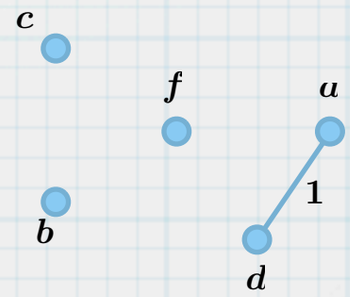
$i = 1$

$Z = X = \{af, ad\}$

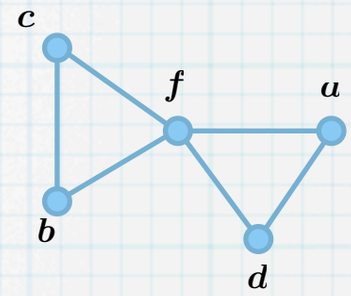
$e = ad$

$e_1 = ad$

$v_1 = d$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$

$m = 6, v_0 = a$

$i = 1$

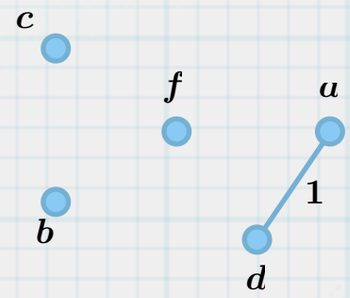
$Z = X = \{af, ad\}$

$e = ad$

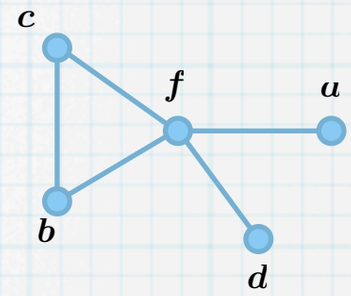
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$

$m = 6, v_0 = a$

$i = 1$

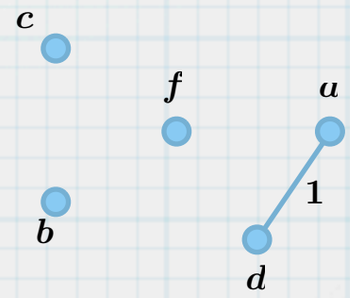
$Z = X = \{af, ad\}$

$e = ad$

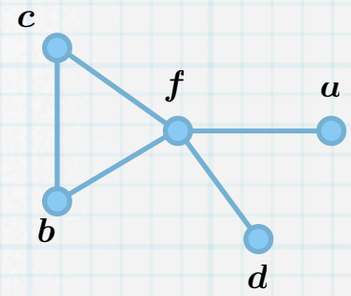
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**

$m = 6, v_0 = a$

$i = 1$

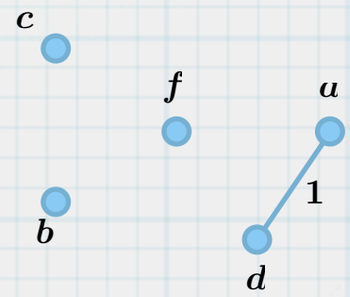
$Z = X = \{af, ad\}$

$e = ad$

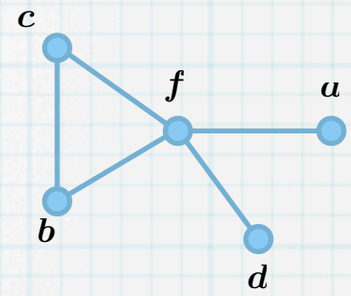
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**

$m = 6, v_0 = a$

$i = 1$

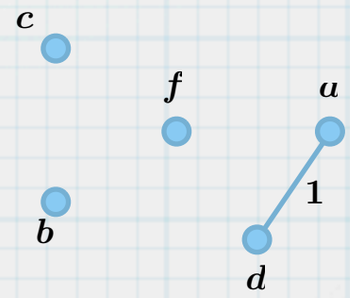
$Z = X = \{af, ad\}$

$e = ad$

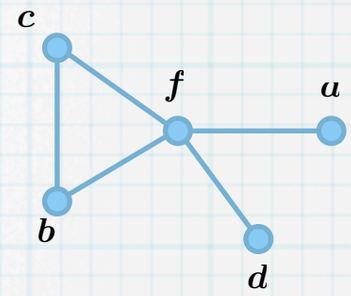
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 1$

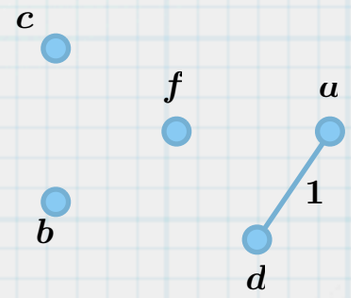
$Z = X = \{af, ad\}$

$e = ad$

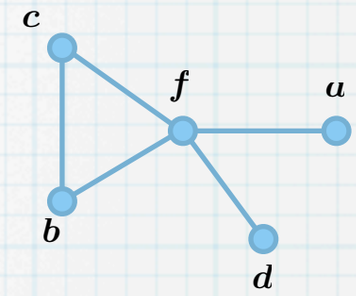
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 1$

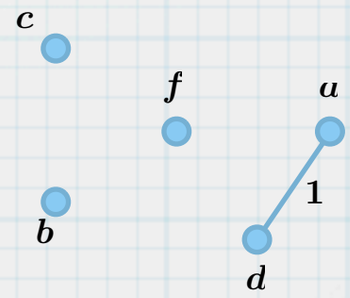
$Z = X = \{af, ad\}$

$e = ad$

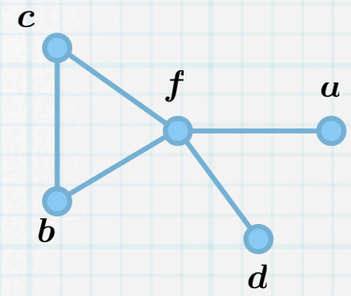
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 1$

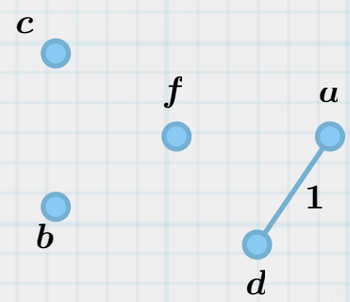
$Z = X = \{af, ad\}$

$e = ad$

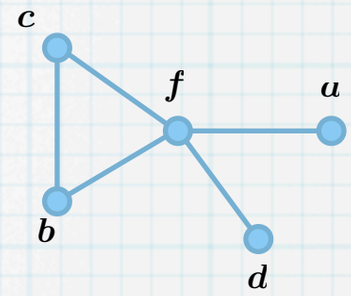
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

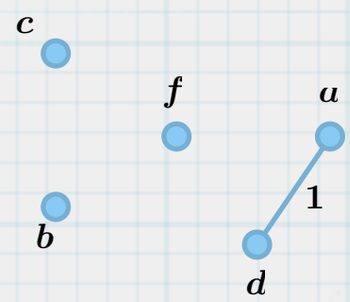
$Z = X = \{af, ad\}$

$e = ad$

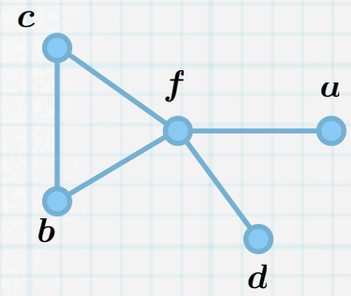
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

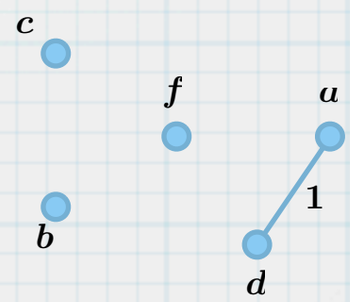
$Z = X = \{af, ad\}$

$e = ad$

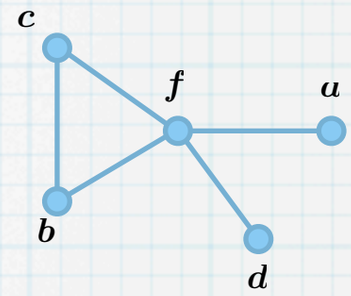
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

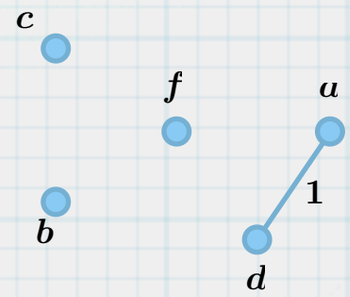
$Z = X = \{df\}$

$e = ad$

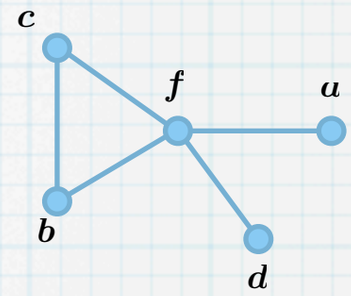
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

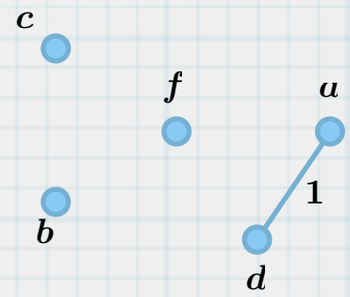
$Z = X = \{df\}$

$e = ad$

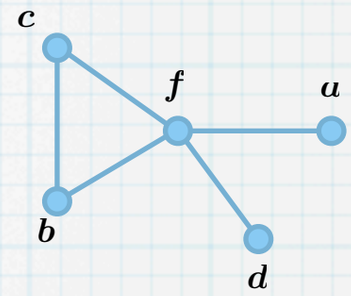
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

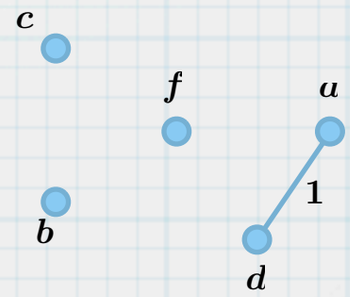
$Z = X = \{df\}$

$e = ad$

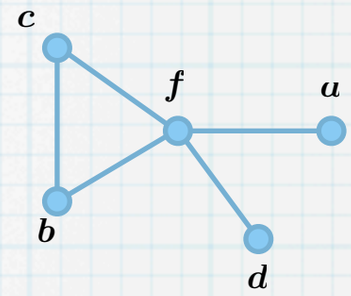
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

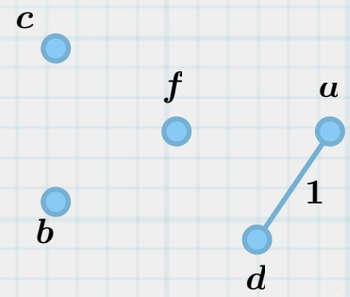
$Z = X = \{df\}$

$e = df$

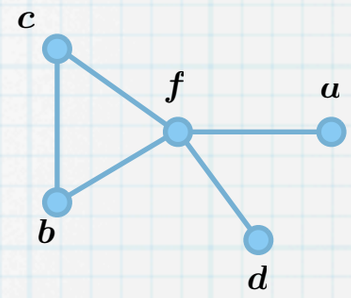
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

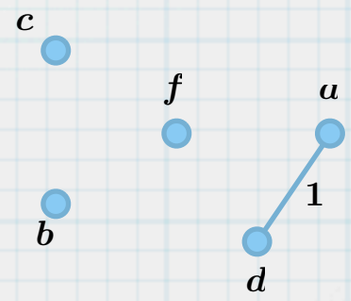
$Z = X = \{df\}$

$e = df$

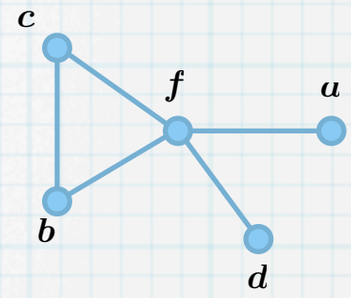
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

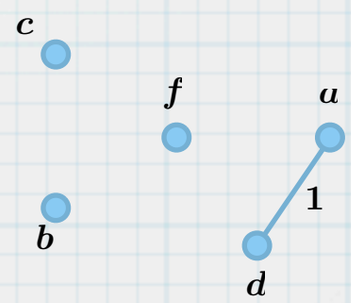
$Z = X = \{df\}$

$e = df$

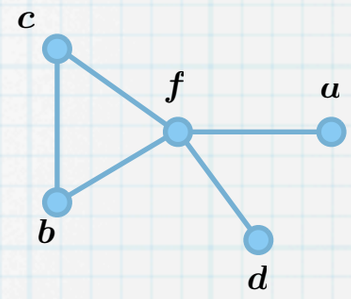
$e_1 = ad$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

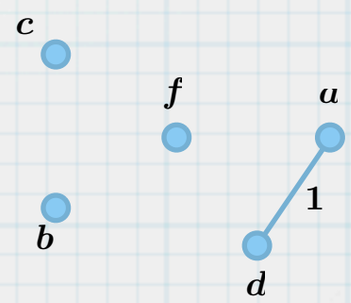
$e = df$

$e_1 = ad$

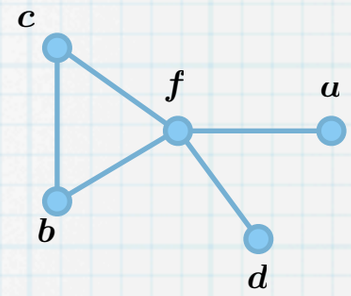
$X = \{\}$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

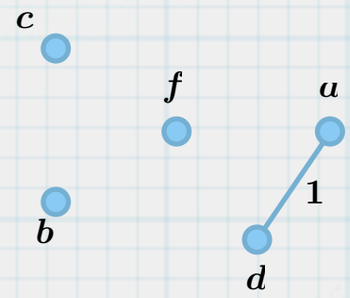
$e = df$

$e_1 = ad$

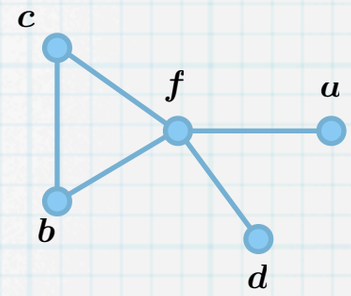
$X = \{\}$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

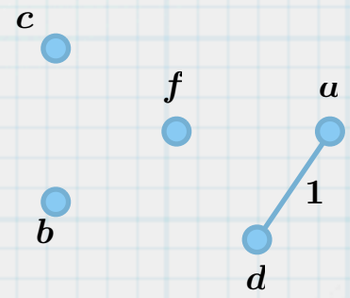
$e = df$

$e_1 = ad$

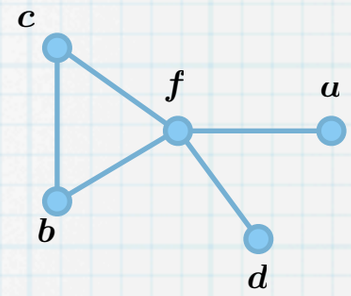
$X = \{\}$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

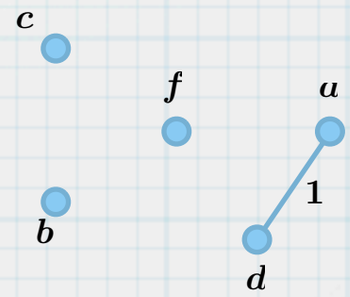
$e_1 = ad$

$X = \{\}$

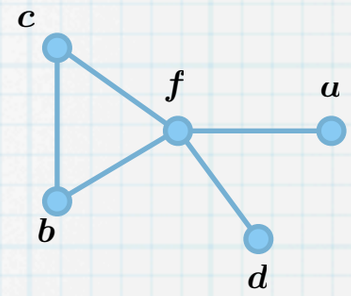
$e_2 = df$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

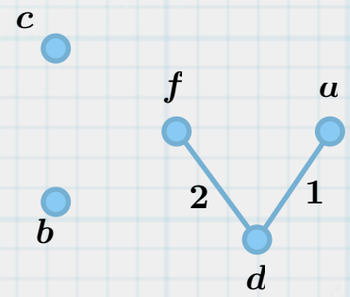
$e_1 = ad$

$X = \{\}$

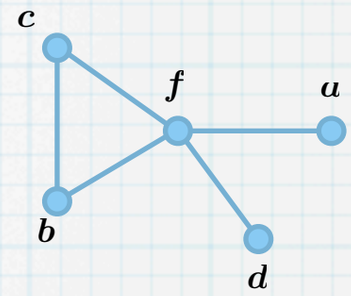
$e_2 = df$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

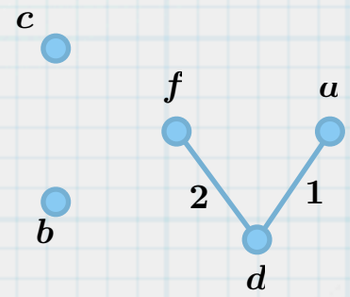
$e_1 = ad$

$X = \{\}$

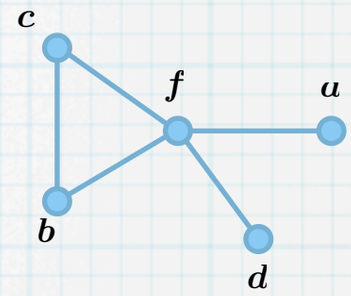
$e_2 = df$

$v_1 = d$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

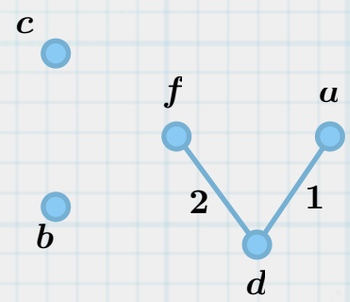
$e_1 = ad$

$X = \{\}$

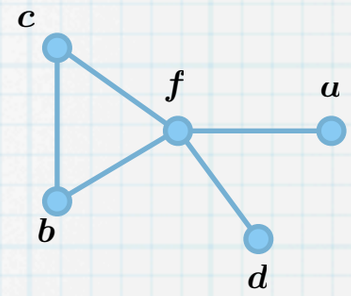
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

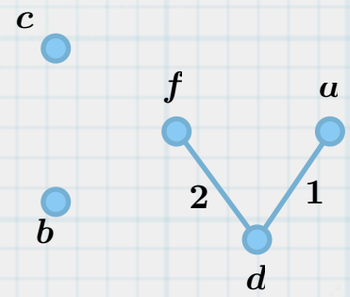
$e_1 = ad$

$X = \{\}$

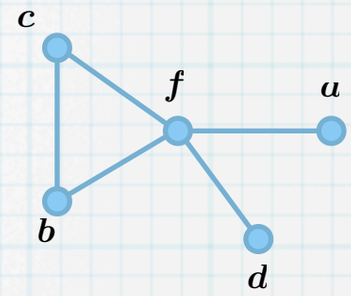
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf, df\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

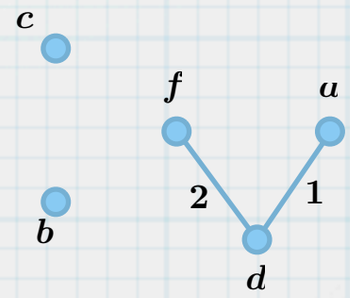
$e_1 = ad$

$X = \{\}$

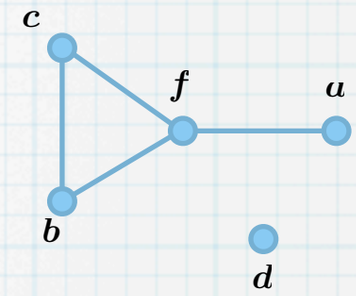
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

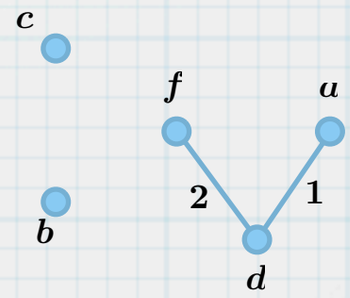
$e_1 = ad$

$X = \{\}$

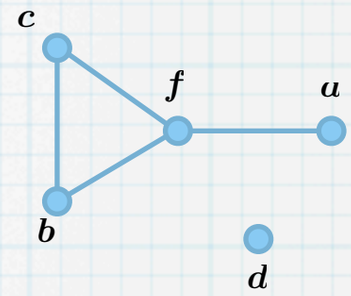
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

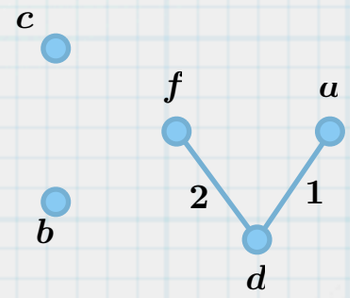
$e_1 = ad$

$X = \{\}$

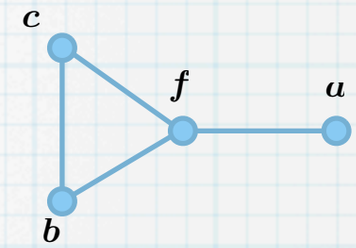
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

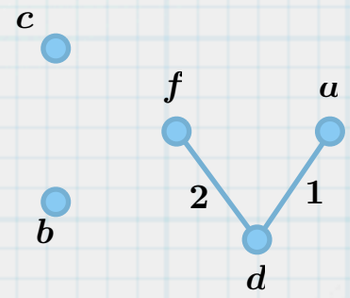
$e_1 = ad$

$X = \{\}$

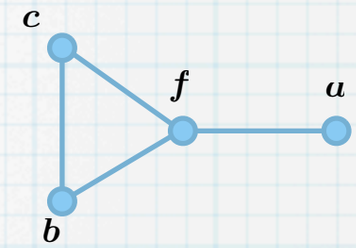
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

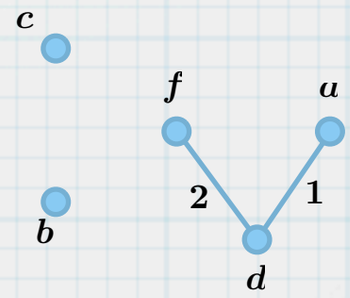
$e_1 = ad$

$X = \{\}$

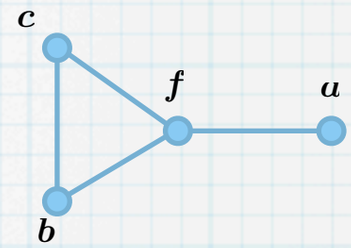
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 2$

$Z = X = \{df\}$

$e = df$

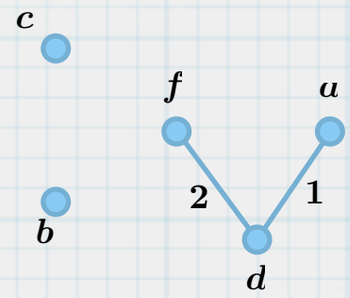
$e_1 = ad$

$X = \{\}$

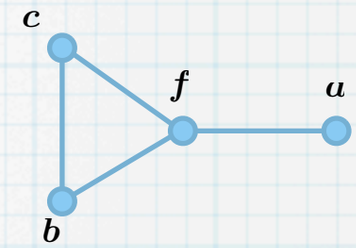
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{df\}$

$e = df$

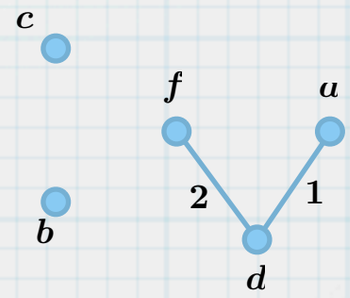
$e_1 = ad$

$X = \{\}$

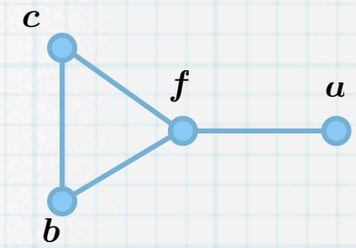
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{df\}$

$e = df$

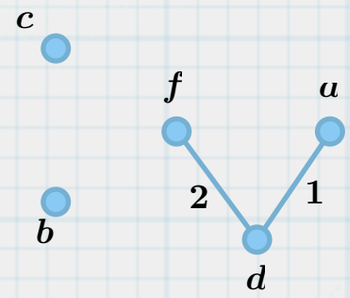
$e_1 = ad$

$X = \{\}$

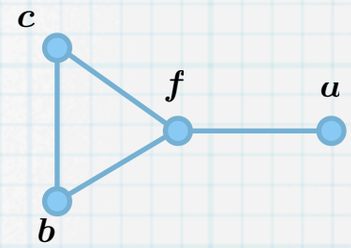
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = df$

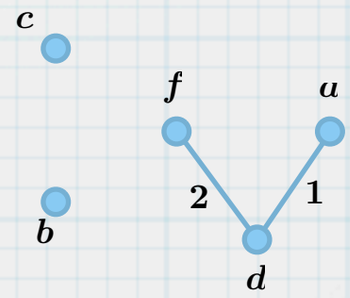
$e_1 = ad$

$X = \{\}$

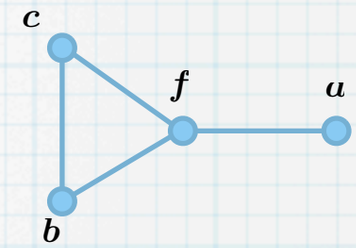
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = df$

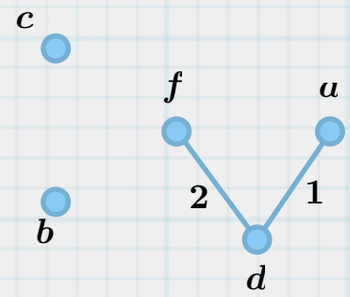
$e_1 = ad$

$X = \{\}$

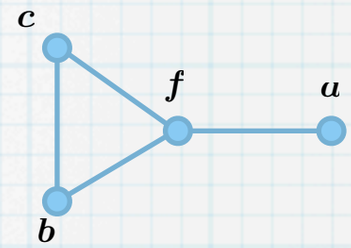
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = df$

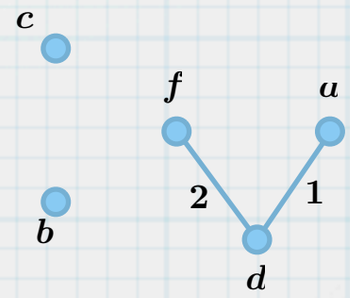
$e_1 = ad$

$X = \{\}$

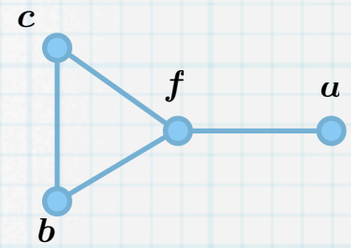
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = af$

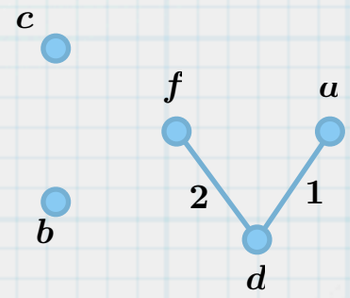
$e_1 = ad$

$X = \{\}$

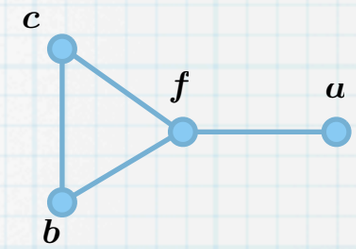
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = af$

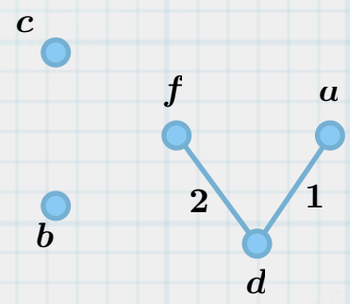
$e_1 = ad$

$X = \{\}$

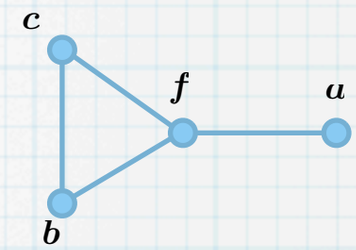
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = af$

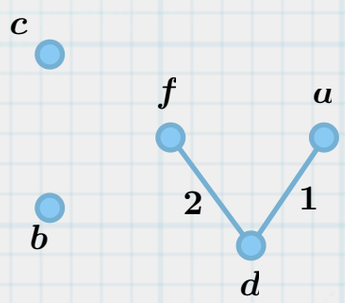
$e_1 = ad$

$X = \{\}$

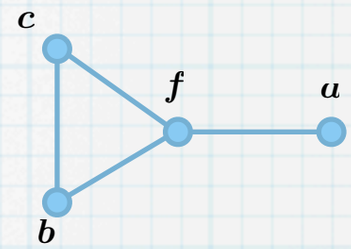
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = af$

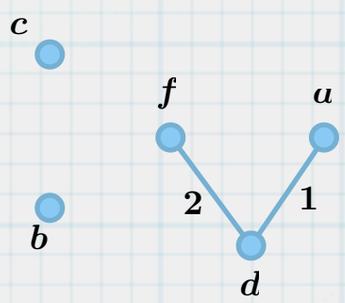
$e_1 = ad$

$X = \{bf, cf\}$

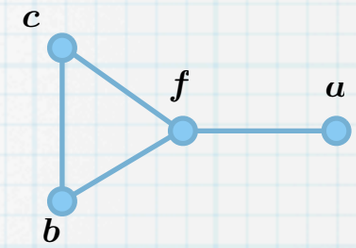
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$$m = 6, v_0 = a$$

$$i = 3$$

$$Z = X = \{af, bf, cf\}$$

$$e = af$$

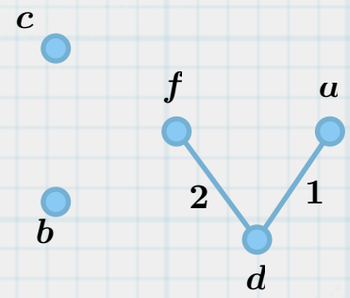
$$e_1 = ad$$

$$X = \{bf, cf\}$$

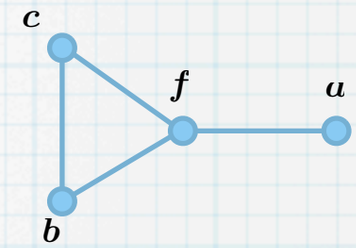
$$e_2 = df$$

$$v_2 = f$$

$$E = \{af, bc, bf, cf\}$$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$$m = 6, v_0 = a$$

$$i = 3$$

$$Z = X = \{af, bf, cf\}$$

$$e = af$$

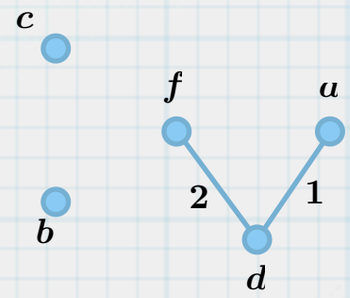
$$e_1 = ad$$

$$X = \{bf, cf\}$$

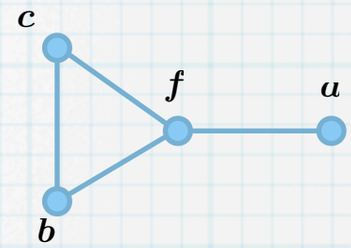
$$e_2 = df$$

$$v_2 = f$$

$$E = \{af, bc, bf, cf\}$$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = af$

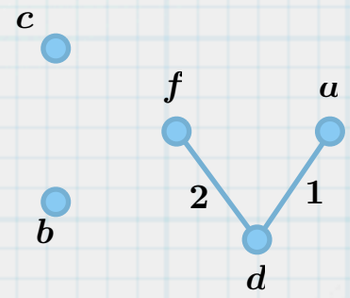
$e_1 = ad$

$X = \{bf, cf\}$

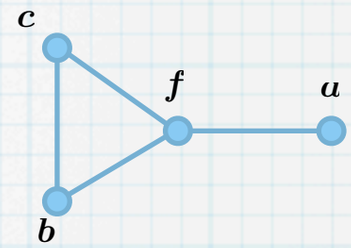
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

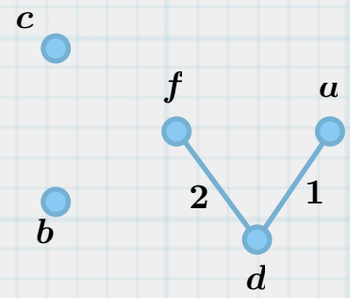
$e_1 = ad$

$X = \{bf, cf\}$

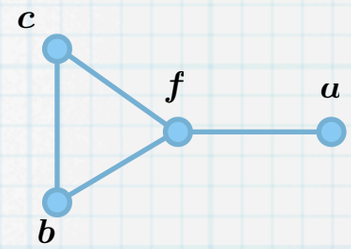
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$$m = 6, v_0 = a$$

$$i = 3$$

$$Z = X = \{af, bf, cf\}$$

$$e = bf$$

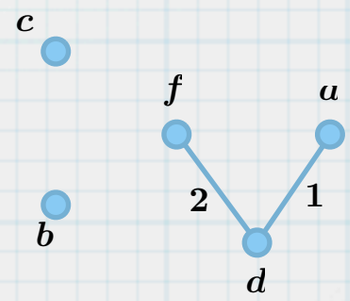
$$e_1 = ad$$

$$X = \{bf, cf\}$$

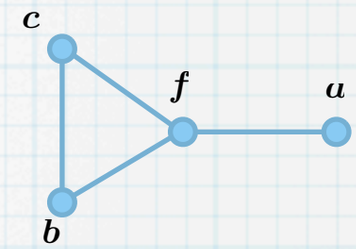
$$e_2 = df$$

$$v_2 = f$$

$$E = \{af, bc, bf, cf\}$$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

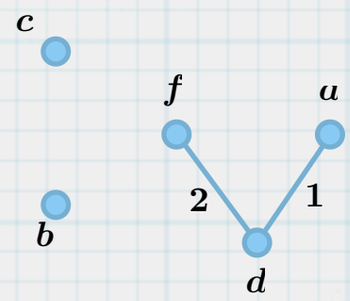
$e_3 = bf$

$X = \{bf, cf\}$

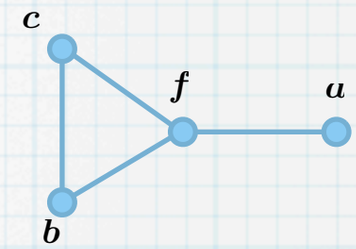
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

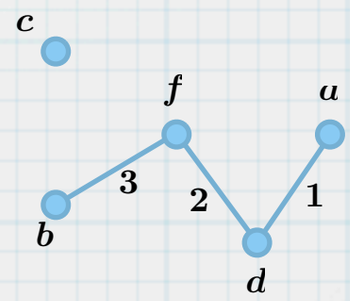
$e_3 = bf$

$X = \{bf, cf\}$

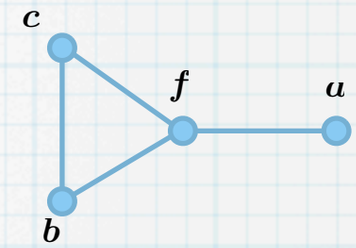
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

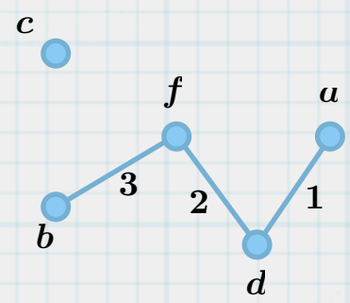
$e_3 = bf$

$X = \{bf, cf\}$

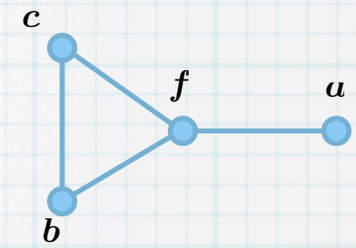
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

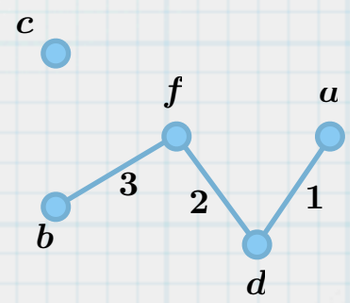
$e_3 = bf$

$X = \{bf, cf\}$

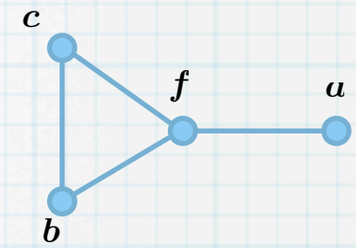
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

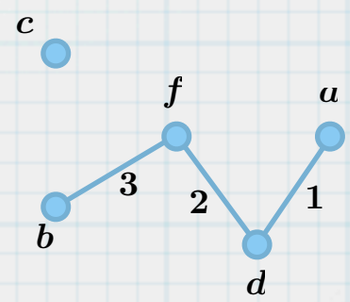
$e_3 = bf$

$X = \{bf, cf\}$

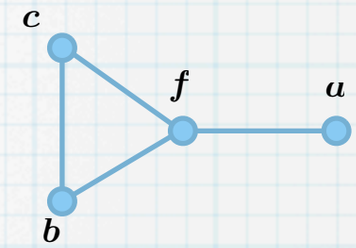
$e_2 = df$

$v_2 = f$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

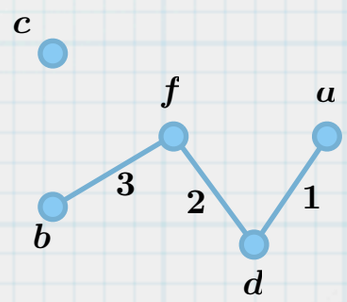
$e_3 = bf$

$X = \{bf, cf\}$

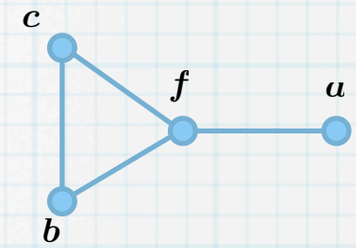
$e_2 = df$

$v_3 = b$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

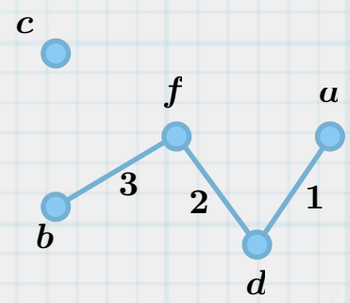
$e_3 = bf$

$X = \{bf, cf\}$

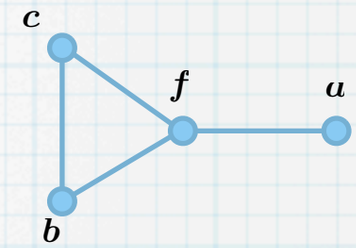
$e_2 = df$

$v_3 = b$

$E = \{af, bc, bf, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

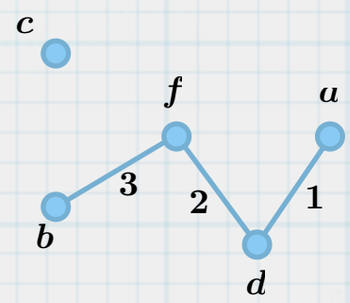
$e_3 = bf$

$X = \{bf, cf\}$

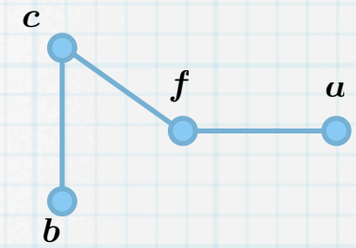
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

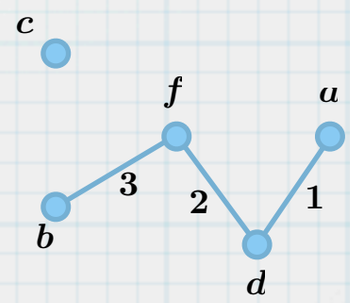
$e_3 = bf$

$X = \{bf, cf\}$

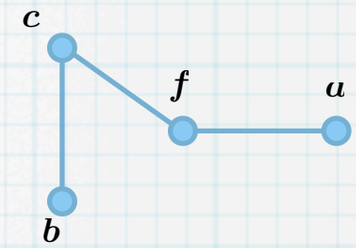
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

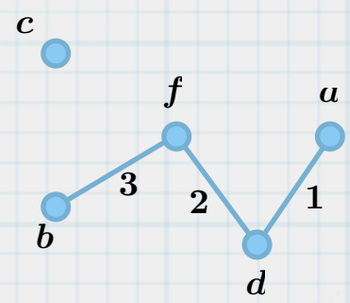
$e_3 = bf$

$X = \{bf, cf\}$

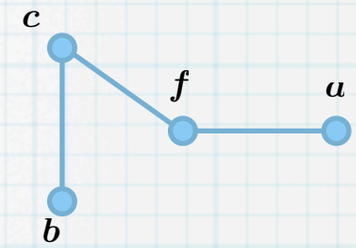
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

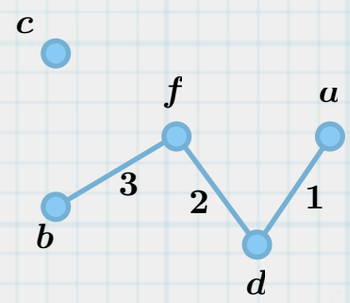
$e_3 = bf$

$X = \{bf, cf\}$

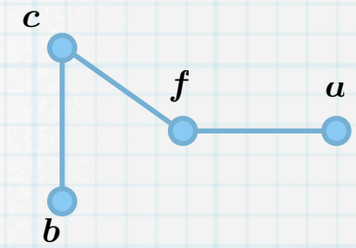
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 3$

$Z = X = \{af, bf, cf\}$

$e = bf$

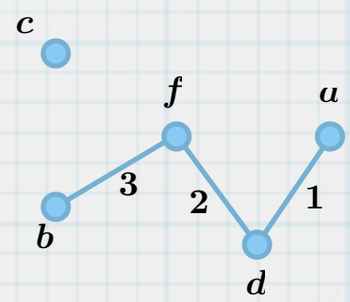
$e_3 = bf$

$X = \{bf, cf\}$

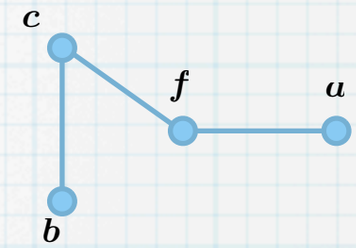
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{af, bf, cf\}$

$e = bf$

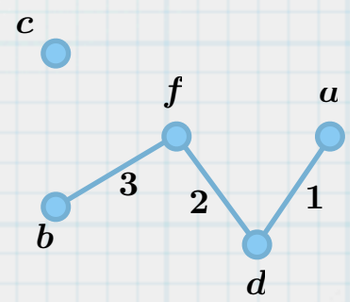
$e_3 = bf$

$X = \{bf, cf\}$

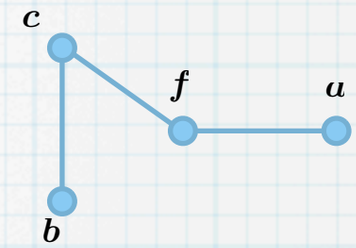
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{af, bf, cf\}$

$e = bf$

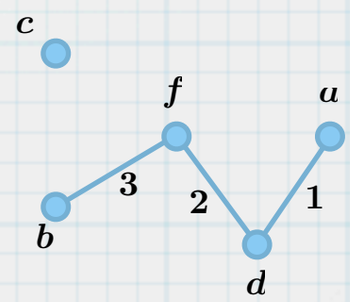
$e_3 = bf$

$X = \{bf, cf\}$

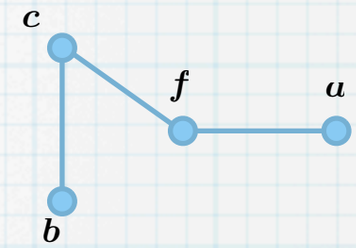
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bf$

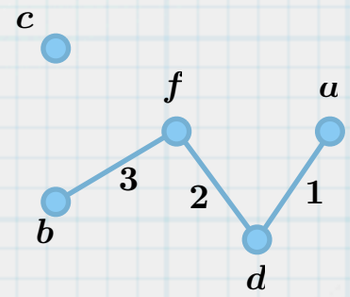
$e_3 = bf$

$X = \{bf, cf\}$

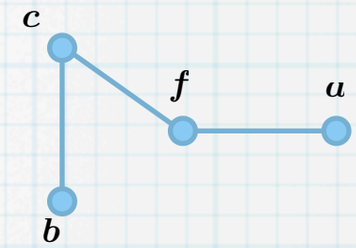
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bf$

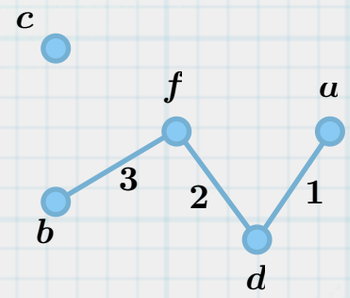
$e_3 = bf$

$X = \{bf, cf\}$

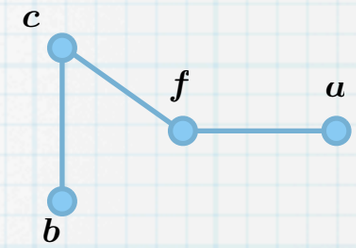
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bf$

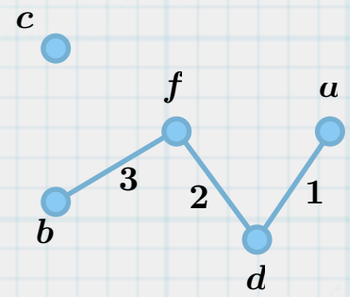
$e_3 = bf$

$X = \{bf, cf\}$

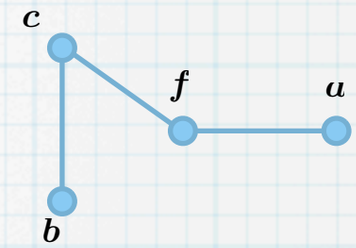
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

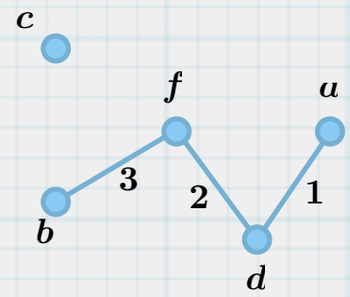
$e_3 = bf$

$X = \{bf, cf\}$

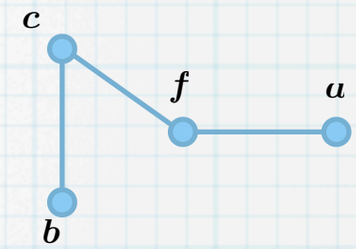
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

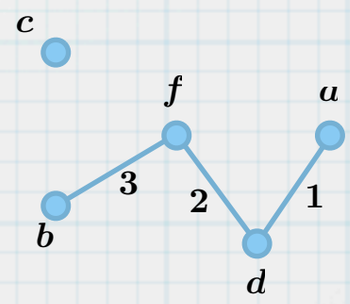
$e_3 = bf$

$X = \{bf, cf\}$

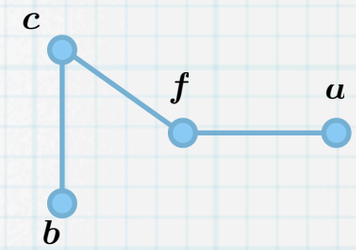
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

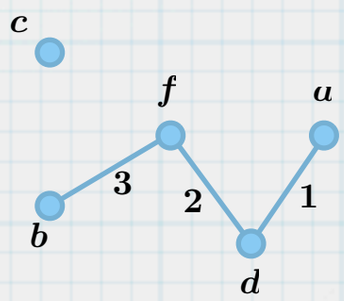
$e_3 = bf$

$X = \{bf, cf\}$

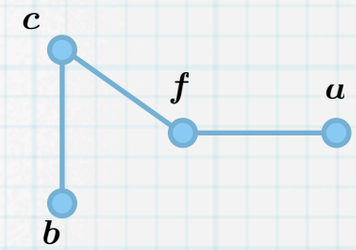
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

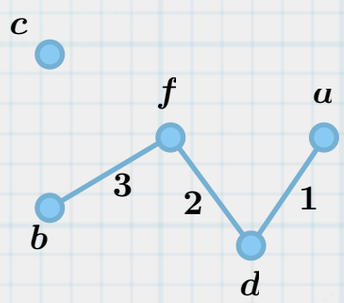
$e_3 = bf$

$X = \{\}$

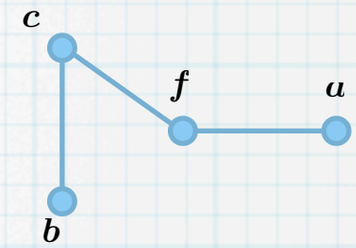
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

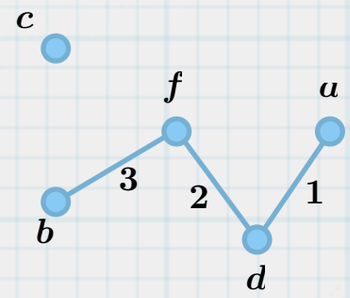
$e_3 = bf$

$X = \{\}$

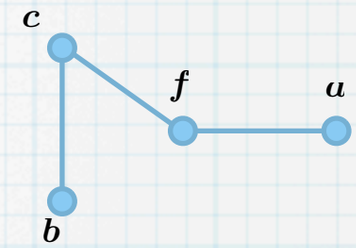
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

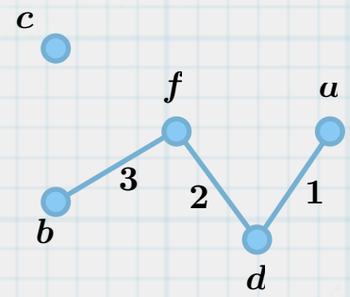
$e_3 = bf$

$X = \{\}$

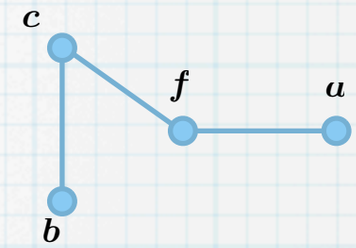
$e_2 = df$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

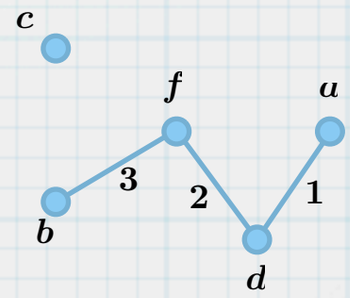
$e_3 = bf$

$X = \{\}$

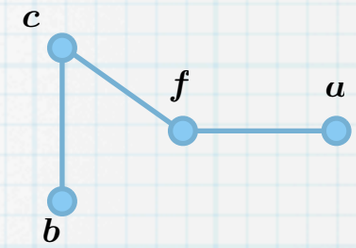
$e_4 = bc$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

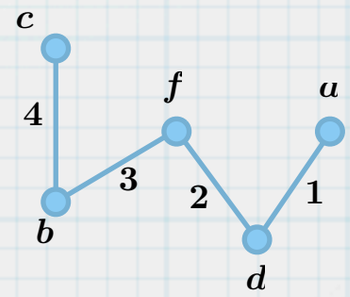
$e_3 = bf$

$X = \{\}$

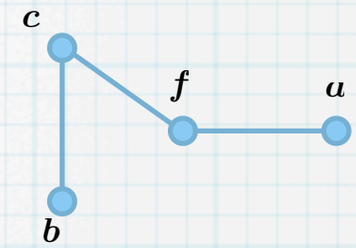
$e_4 = bc$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

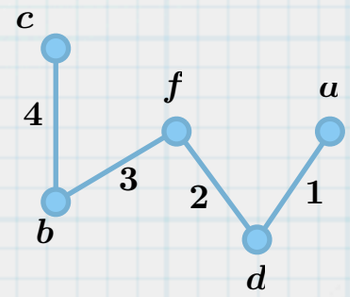
$e_3 = bf$

$X = \{\}$

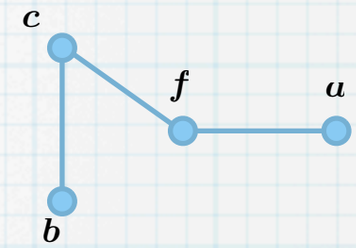
$e_4 = bc$

$v_3 = b$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

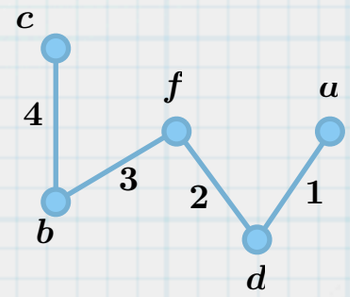
$e_3 = bf$

$X = \{\}$

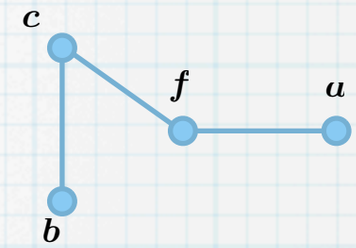
$e_4 = bc$

$v_4 = c$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

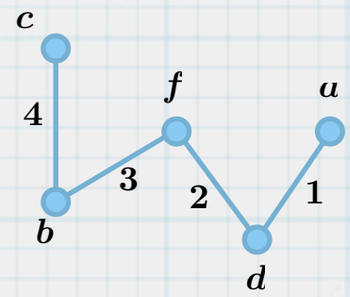
$e_3 = bf$

$X = \{\}$

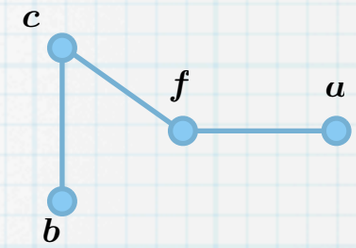
$e_4 = bc$

$v_4 = c$

$E = \{af, bc, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

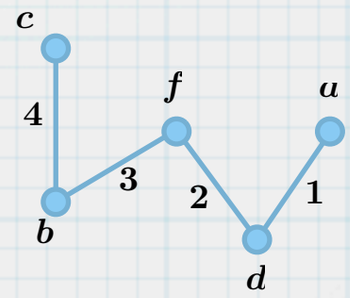
$e_3 = bf$

$X = \{\}$

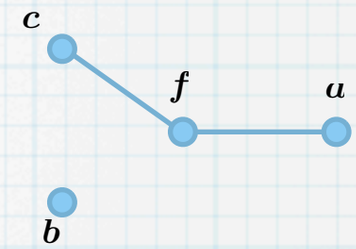
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

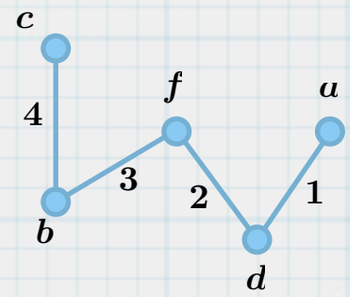
$e_3 = bf$

$X = \{\}$

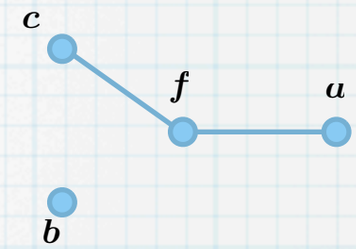
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

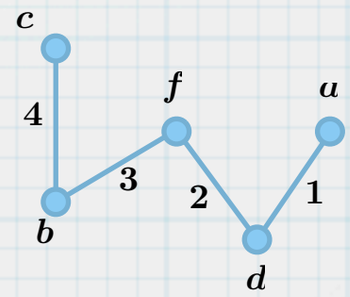
$e_3 = bf$

$X = \{\}$

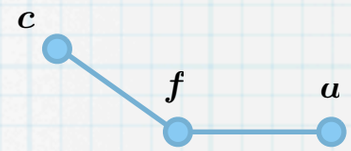
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

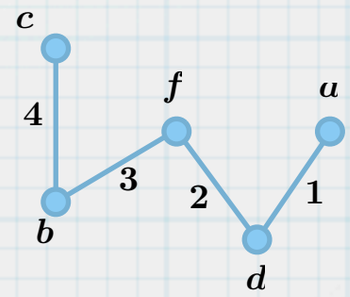
$e_3 = bf$

$X = \{\}$

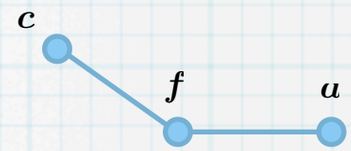
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

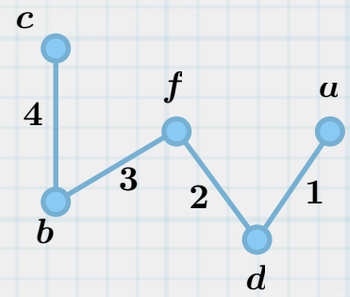
$e_3 = bf$

$X = \{\}$

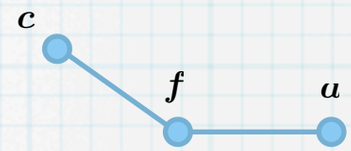
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 4$

$Z = X = \{bc\}$

$e = bc$

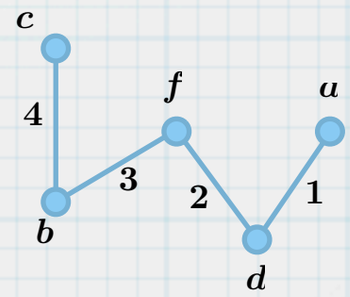
$e_3 = bf$

$X = \{\}$

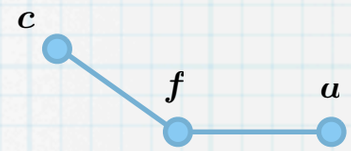
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{bc\}$

$e = bc$

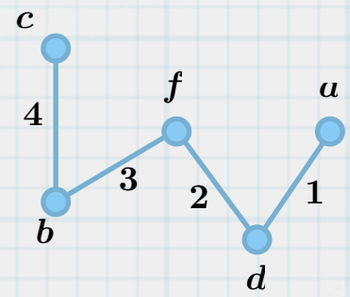
$e_3 = bf$

$X = \{\}$

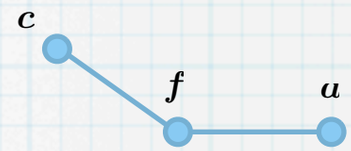
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{bc\}$

$e = bc$

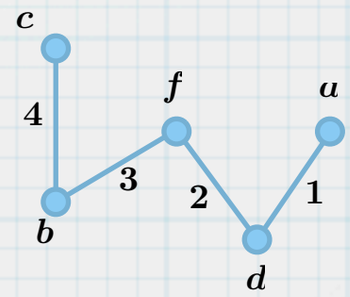
$e_3 = bf$

$X = \{\}$

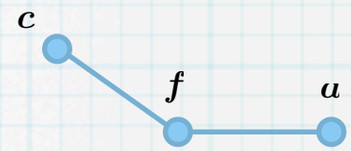
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = bc$

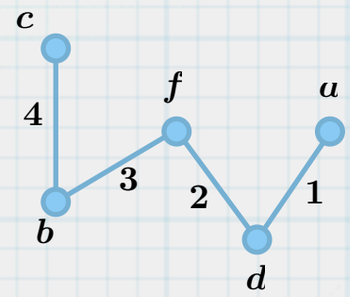
$e_3 = bf$

$X = \{\}$

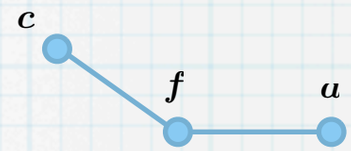
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$

* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury

(2) $m := |E|, v_0 \in V$

(3) **for** i **from** 1 **to** m **do**

(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$

(5) **repeat**

(6) $e \in X$

(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$

(8) **else** $X := X \backslash \{e\}$ **end if**

(9) **until** e_i definiert **or** $X = \emptyset$

(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**

(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$

(12) $E := E \backslash \{e_i\}$

(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**

(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = bc$

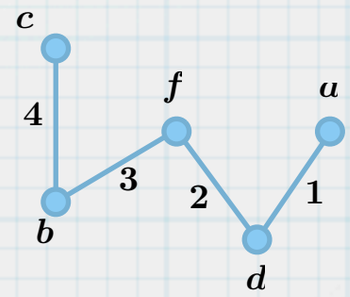
$e_3 = bf$

$X = \{\}$

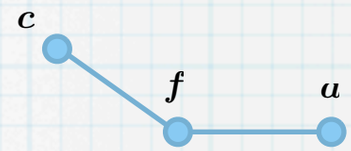
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = bc$

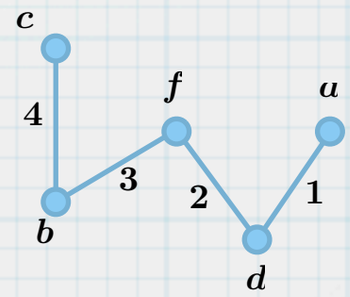
$e_3 = bf$

$X = \{\}$

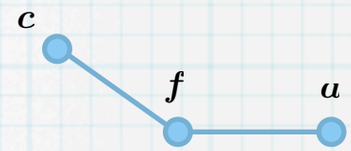
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

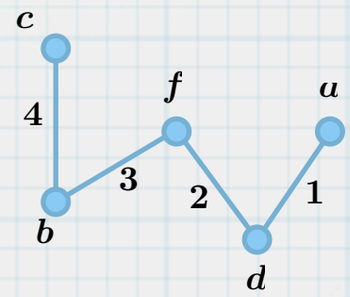
$e_3 = bf$

$X = \{\}$

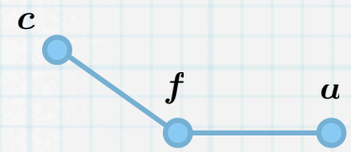
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

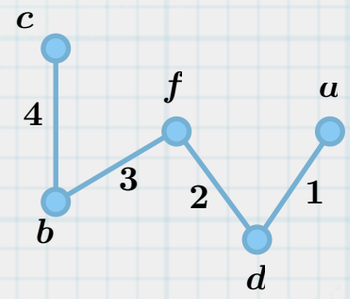
$e_3 = bf$

$X = \{\}$

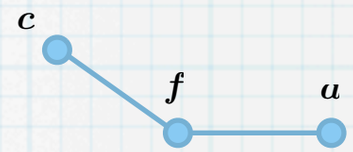
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

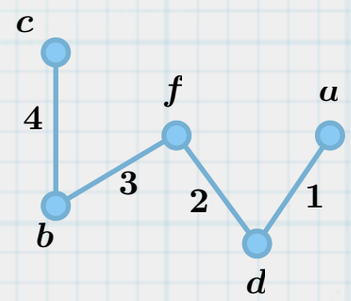
$e_3 = bf$

$X = \{\}$

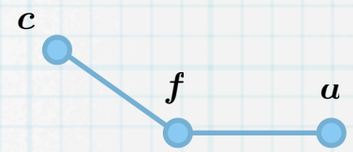
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

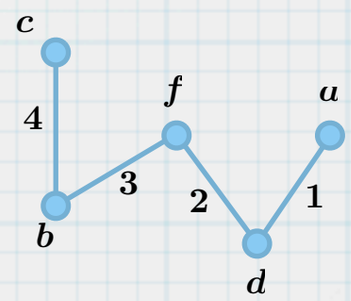
$e_3 = bf$

$X = \{\}$

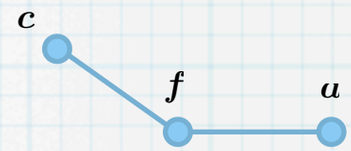
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

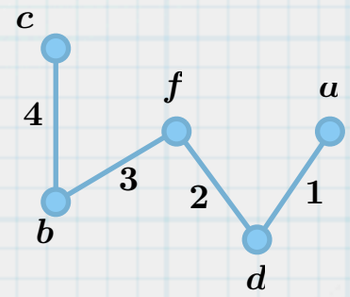
$e_3 = bf$

$X = \{\}$

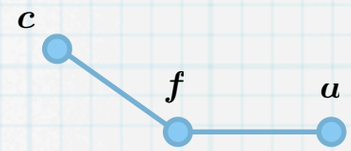
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

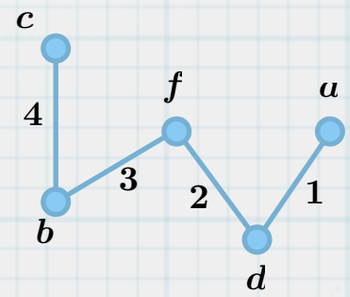
$e_3 = bf$

$X = \{\}$

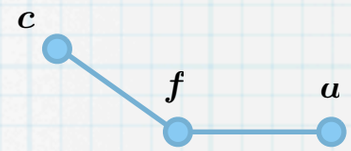
$e_4 = bc$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

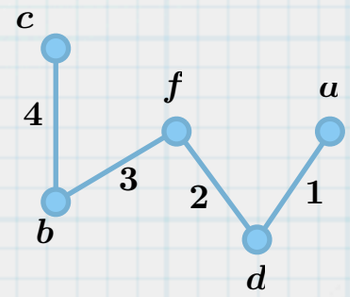
$e_3 = bf$

$X = \{\}$

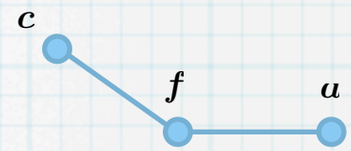
$e_5 = cf$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

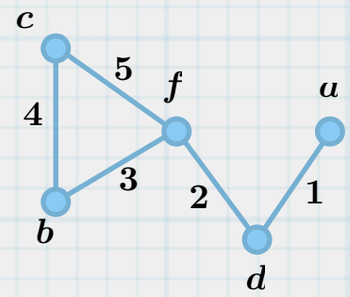
$e_3 = bf$

$X = \{\}$

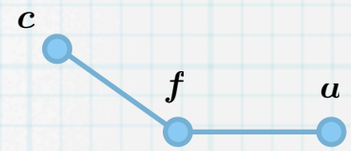
$e_5 = cf$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

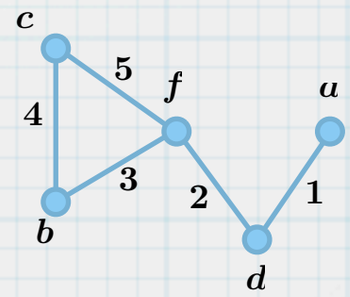
$e_3 = bf$

$X = \{\}$

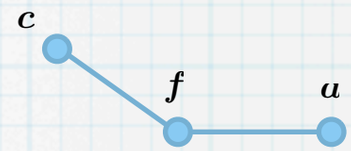
$e_5 = cf$

$v_4 = c$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

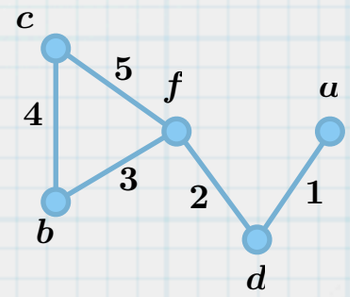
$e_3 = bf$

$X = \{\}$

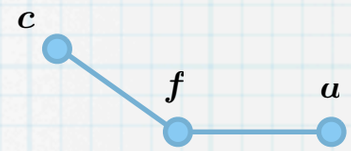
$e_5 = cf$

$v_5 = f$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

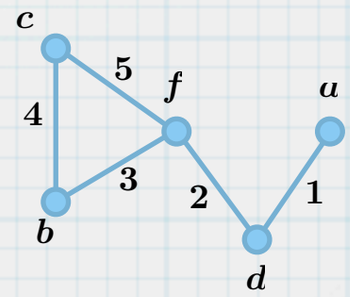
$e_3 = bf$

$X = \{\}$

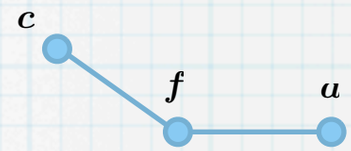
$e_5 = cf$

$v_5 = f$

$E = \{af, cf\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

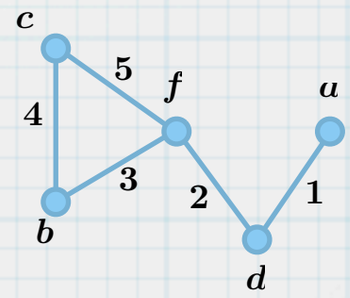
$e_3 = bf$

$X = \{\}$

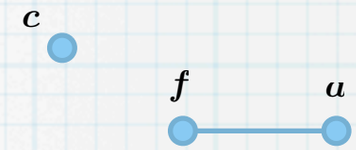
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

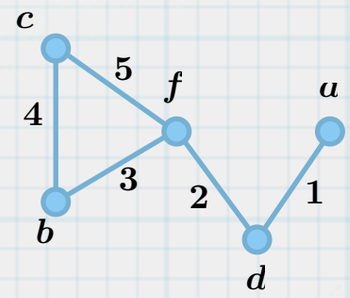
$e_3 = bf$

$X = \{\}$

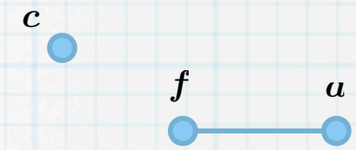
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

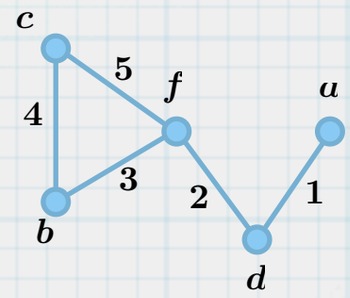
$e_3 = bf$

$X = \{\}$

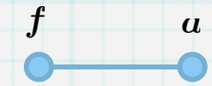
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

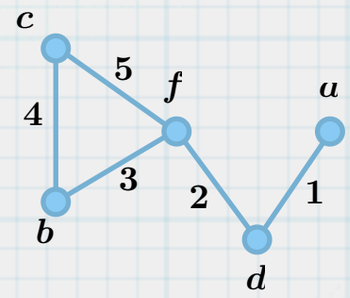
$e_3 = bf$

$X = \{\}$

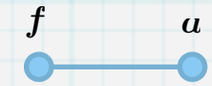
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

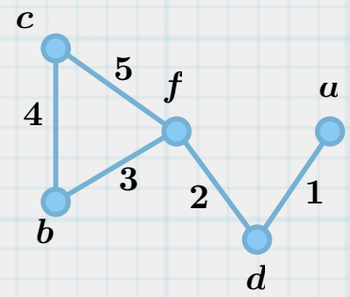
$e_3 = bf$

$X = \{\}$

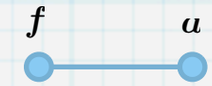
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 5$

$Z = X = \{cf\}$

$e = cf$

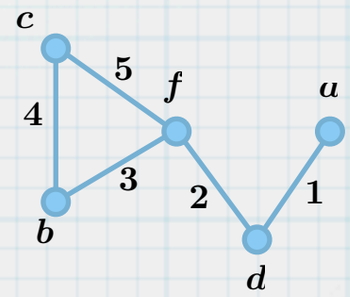
$e_3 = bf$

$X = \{\}$

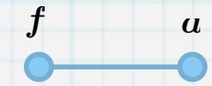
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{cf\}$

$e = cf$

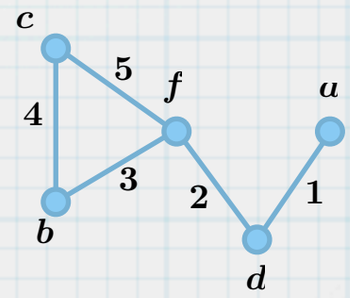
$e_3 = bf$

$X = \{\}$

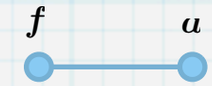
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{cf\}$

$e = cf$

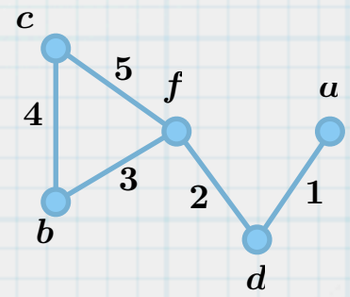
$e_3 = bf$

$X = \{\}$

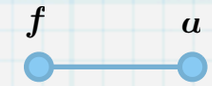
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = cf$

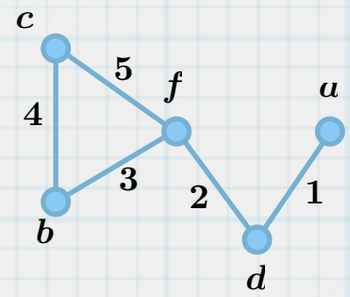
$e_3 = bf$

$X = \{\}$

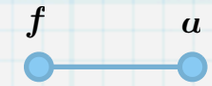
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = cf$

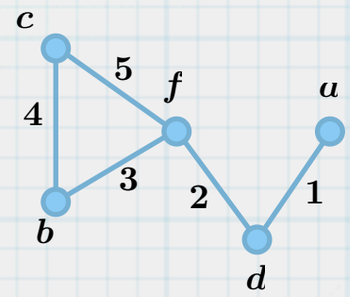
$e_3 = bf$

$X = \{\}$

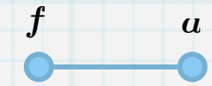
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = cf$

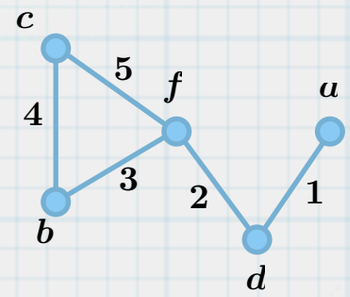
$e_3 = bf$

$X = \{\}$

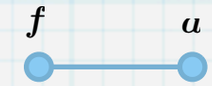
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

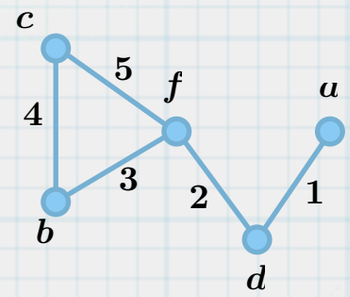
$e_3 = bf$

$X = \{\}$

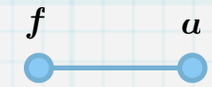
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

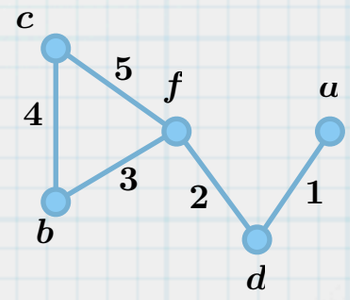
$e_3 = bf$

$X = \{\}$

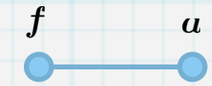
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

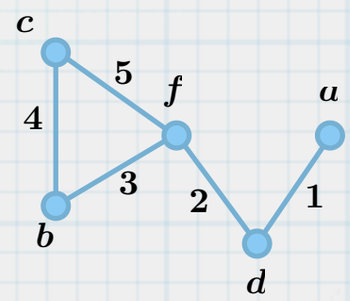
$e_3 = bf$

$X = \{\}$

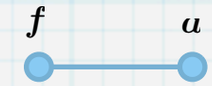
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

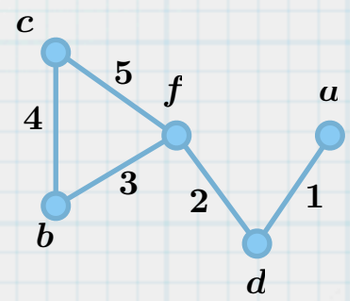
$e_3 = bf$

$X = \{\}$

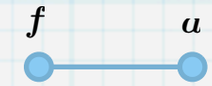
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

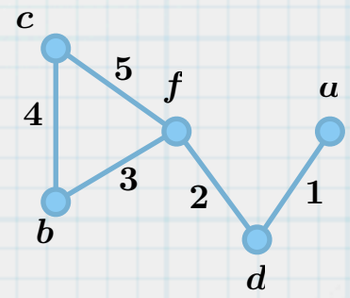
$e_3 = bf$

$X = \{\}$

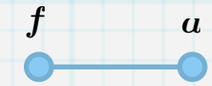
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

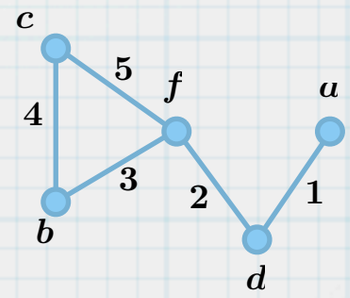
$e_3 = bf$

$X = \{\}$

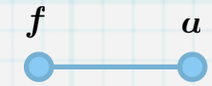
$e_5 = cf$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

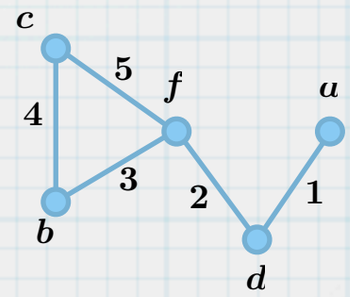
$e_3 = bf$

$X = \{\}$

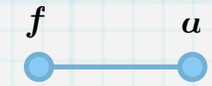
$e_6 = af$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

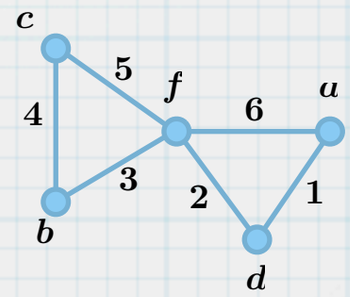
$e_3 = bf$

$X = \{\}$

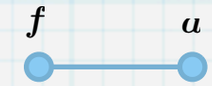
$e_6 = af$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

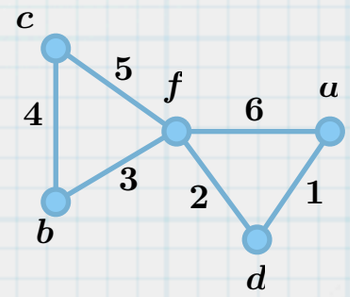
$e_3 = bf$

$X = \{\}$

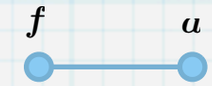
$e_6 = af$

$v_5 = f$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

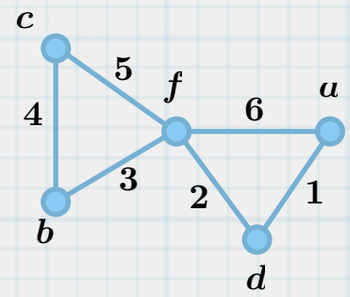
$e_3 = bf$

$X = \{\}$

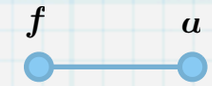
$e_6 = af$

$v_6 = a$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

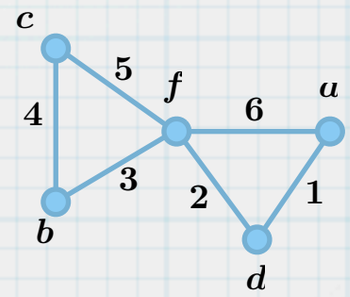
$e_3 = bf$

$X = \{\}$

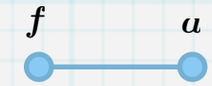
$e_6 = af$

$v_6 = a$

$E = \{af\}$



Algorithmus von Fleury



* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

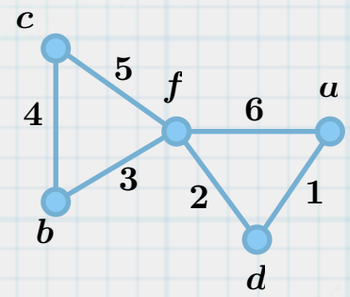
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Algorithmus von Fleury

* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

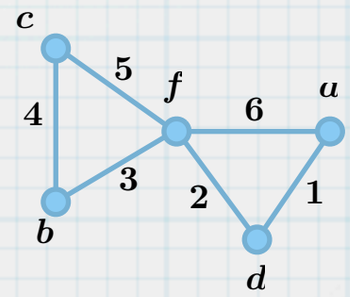
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Algorithmus von Fleury

* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

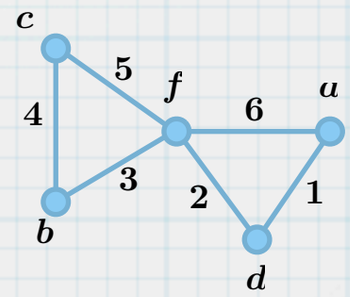
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Algorithmus von Fleury

- Eingabe: Eulerscher Graph $G = (V, E)$
- Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

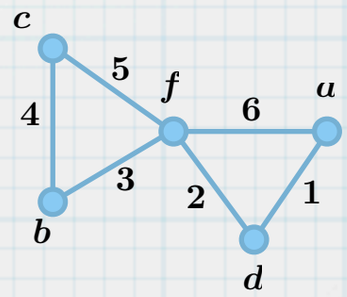
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Algorithmus von Fleury

- Eingabe: Eulerscher Graph $G = (V, E)$
- Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

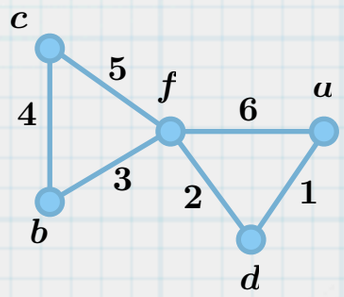
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Algorithmus von Fleury

- Eingabe: Eulerscher Graph $G = (V, E)$
- Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

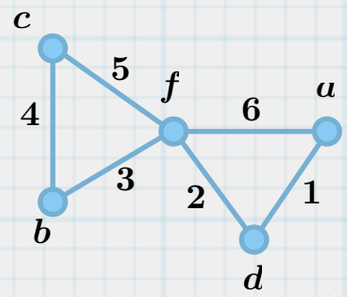
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Algorithmus von Fleury

- Eingabe: Eulerscher Graph $G = (V, E)$
- Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**
(15) **end algorithm**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

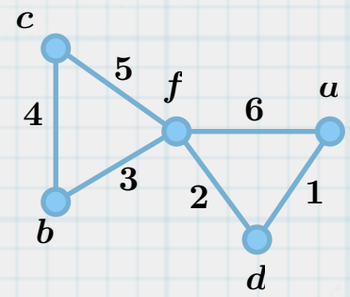
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Algorithmus von Fleury

* Eingabe: Eulerscher Graph $G = (V, E)$
* Ausgabe: Eulertour $(e_1, \ldots, e_{|E|})$ für G

(1) **algorithm** fleury
(2) $m := |E|, v_0 \in V$
(3) **for** i **from** 1 **to** m **do**
(4) $Z := X := \{\{v, w\} \in E : \{v, w\} \cap \{v_{i-1}\} \neq \emptyset\}$
(5) **repeat**
(6) $e \in X$
(7) **if** $(V, E \backslash e)$ zusammenhängend **then** $e_i := e$
(8) **else** $X := X \backslash \{e\}$ **end if**
(9) **until** e_i definiert **or** $X = \emptyset$
(10) **if** $X = \emptyset$ **then** $e_i \in Z$ **end if**
(11) sei $v_i \in V$ so, dass $e_i = \{v_{i-1}, v_i\}$
(12) $E := E \backslash \{e_i\}$
(13) **if** $\deg(v_{i-1}) = 0$ **then** $V := V \backslash \{v_{i-1}\}$ **end if**
(14) **end for**
(15) **end algorithm**

$m = 6, v_0 = a$

$i = 6$

$Z = X = \{af\}$

$e = af$

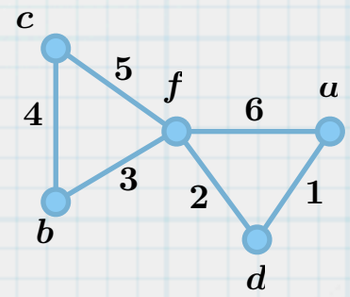
$e_3 = bf$

$X = \{\}$

$e_6 = af$

$v_6 = a$

$E = \{\}$



Korrektheitsbeweis zum Algorithmus von Fleury

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V | \exists\, e \in E(G_i) : \{v\} \subset e\}$.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

  1. Fall, $v_0 \neq v_i$.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

  1. Fall, $v_0 \neq v_i$.

     In diesem Fall gibt es neben e_i noch eine gerade Anzahl Kanten, die mit v_0 bzw. v_i inzidieren (je zwei für jedes weitere Vorkommen von v_0 bzw. v_i in P_i).

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

  1. Fall, $v_0 \neq v_i$.

     In diesem Fall gibt es neben e_i noch eine gerade Anzahl Kanten, die mit v_0 bzw. v_i inzidieren (je zwei für jedes weitere Vorkommen von v_0 bzw. v_i in P_i).

     Also sind $\deg(v_0)$ bzw. $\deg(v_i)$ ungerade.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

  1. Fall, $v_0 \neq v_i$.

     In diesem Fall gibt es neben e_i noch eine gerade Anzahl Kanten, die mit v_0 bzw. v_i inzidieren (je zwei für jedes weitere Vorkommen von v_0 bzw. v_i in P_i).

     Also sind $\deg(v_0)$ bzw. $\deg(v_i)$ ungerade.

     1. Fall, $\deg(v_i) = 1$.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

  1. Fall, $v_0 \neq v_i$.

     In diesem Fall gibt es neben e_i noch eine gerade Anzahl Kanten, die mit v_0 bzw. v_i inzidieren (je zwei für jedes weitere Vorkommen von v_0 bzw. v_i in P_i).

     Also sind $\deg(v_0)$ bzw. $\deg(v_i)$ ungerade.

     1. Fall, $\deg(v_i) = 1$.

        Dann gibt es nur eine mit v_i inzidente Kante e in G_i.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

  1. Fall, $v_0 \neq v_i$.

     In diesem Fall gibt es neben e_i noch eine gerade Anzahl Kanten, die mit v_0 bzw. v_i inzidieren (je zwei für jedes weitere Vorkommen von v_0 bzw. v_i in P_i).

     Also sind $\deg(v_0)$ bzw. $\deg(v_i)$ ungerade.

     1. Fall, $\deg(v_i) = 1$.

        Dann gibt es nur eine mit v_i inzidente Kante e in G_i.

        In Schritt (10) wird diese Kante e gewählt, um P_i zu P_{i+1} zu verlängern.

Korrektheitsbeweis zum Algorithmus von Fleury

* **Satz 9**:
  Der Algorithmus von Fleury liefert eine Eulertour, falls der Graph Eulersch ist.
* Beweis:

  Mit $W_i = (V, E_i)$ bezeichnen wir den Graphen, der aus allen Knoten und den Kanten des Kantenzugs $P_i = (v_0, e_1, v_1, e_2, v_2, \ldots, e_i, v_i)$ besteht.

  Mit G_i bezeichnen wir den Restgraphen am Ende der Iteration, also $E(G_i) = E \backslash \{e_1, \ldots, e_i\}$ und $V(G_i) = \{v \in V \mid \exists\, e \in E(G_i) : \{v\} \subset e\}$.

  Jeder Knoten $v \notin \{v_0, v_i\}$ im Kantenzug von P_i hat eine Kante, die zu ihm hinführt, und eine, die von ihm wegführt.

  Also ist der Grad von v in W_i stets gerade. Ebenso ist der Grad von v in G_i stets gerade.

  1. Fall, $v_0 \neq v_i$.

     In diesem Fall gibt es neben e_i noch eine gerade Anzahl Kanten, die mit v_0 bzw. v_i inzidieren (je zwei für jedes weitere Vorkommen von v_0 bzw. v_i in P_i).

     Also sind $\deg(v_0)$ bzw. $\deg(v_i)$ ungerade.

     1. Fall, $\deg(v_i) = 1$.

        Dann gibt es nur eine mit v_i inzidente Kante e in G_i.

        In Schritt (10) wird diese Kante e gewählt, um P_i zu P_{i+1} zu verlängern.

        Der Graph $G_i - e$ wird dadurch unzusammenhängend, bzw. v_i wird in G_{i+1} entfernt.

Fleury: Korrektheitsbeweis (Forts.)

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
    2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
    2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
        Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.

   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.

      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.

      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.
         Dann hat $G_i - \{e', f'\}$ genau vier ungerade Knoten: v_0, v_i, u, w.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.
         Dann hat $G_i - \{e', f'\}$ genau vier ungerade Knoten: v_0, v_i, u, w.
         Also hat (mind.) eine der Komponenten C_1, C_2, C_3 genau einen ungeraden Knoten. Widerspruch, da die Anzahl ungerader Knoten pro Komponente gerade sein muss.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.
         Dann hat $G_i - \{e', f'\}$ genau vier ungerade Knoten: v_0, v_i, u, w.
         Also hat (mind.) eine der Komponenten C_1, C_2, C_3 genau einen ungeraden Knoten. Widerspruch, da die Anzahl ungerader Knoten pro Komponente gerade sein muss.
      2. Fall, $v_0 \in \{u, w\}$. O.B.d.A. $v_0 = u$.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.
         Dann hat $G_i - \{e', f'\}$ genau vier ungerade Knoten: v_0, v_i, u, w.
         Also hat (mind.) eine der Komponenten C_1, C_2, C_3 genau einen ungeraden Knoten. Widerspruch, da die Anzahl ungerader Knoten pro Komponente gerade sein muss.
      2. Fall, $v_0 \in \{u, w\}$. O.B.d.A. $v_0 = u$.
         Dann hat $G_i - \{e', f'\}$ genau zwei ungerade Knoten: v_i, w.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.
         Dann hat $G_i - \{e', f'\}$ genau vier ungerade Knoten: v_0, v_i, u, w.
         Also hat (mind.) eine der Komponenten C_1, C_2, C_3 genau einen ungeraden Knoten. Widerspruch, da die Anzahl ungerader Knoten pro Komponente gerade sein muss.
      2. Fall, $v_0 \in \{u, w\}$. O.B.d.A. $v_0 = u$.
         Dann hat $G_i - \{e', f'\}$ genau zwei ungerade Knoten: v_i, w.
         Also gibt es eine Zusammenhangskomponente, die genau einen ungeraden Knoten hat. Widerspruch.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.
         Dann hat $G_i - \{e', f'\}$ genau vier ungerade Knoten: v_0, v_i, u, w.
         Also hat (mind.) eine der Komponenten C_1, C_2, C_3 genau einen ungeraden Knoten. Widerspruch, da die Anzahl ungerader Knoten pro Komponente gerade sein muss.
      2. Fall, $v_0 \in \{u, w\}$. O.B.d.A. $v_0 = u$.
         Dann hat $G_i - \{e', f'\}$ genau zwei ungerade Knoten: v_i, w.
         Also gibt es eine Zusammenhangskomponente, die genau einen ungeraden Knoten hat. Widerspruch.

Damit ist die Behauptung gezeigt.

Fleury: Korrektheitsbeweis (Forts.)

1. Fall, $v_0 \neq v_i$.
   2. Fall, $\deg(v_i) > 1$, d.h. $\deg(v_i) \geq 3$.
      Behauptung: es gibt eine mit v_i inzidente Kante e, so dass $G_i - e$ zusammenhängt.
      Angenommen, es gibt keine solche Kante. D.h. für alle noch mit v_i inzidenten Kanten e ist $G_i - e$ unzusammenhängend.
      Seien $e' = \{v_i, u\}$ und $f' = \{v_i, w\}$ zwei solche Kanten.
      In $G_i - \{e', f'\}$ ist der Grad von v_i um 2 verringert.
      Somit ist $\deg(v_i)$ in $G_i - \{e', f'\}$ immer noch ungerade.
      Der Untergraph $G_i - \{e', f'\}$ besteht aus mind. 3 Komponenten C_1, C_2, C_3.
      O.B.d.A. sei $v_i \in C_1, u \in C_2$ und $w \in C_3$.
      Die einzigen ungeraden Knoten in G_i sind v_0 und v_i (siehe oben).
      1. Fall, $v_0 \notin \{u, w\}$.
         Dann hat $G_i - \{e', f'\}$ genau vier ungerade Knoten: v_0, v_i, u, w.
         Also hat (mind.) eine der Komponenten C_1, C_2, C_3 genau einen ungeraden Knoten. Widerspruch, da die Anzahl ungerader Knoten pro Komponente gerade sein muss.
      2. Fall, $v_0 \in \{u, w\}$. O.B.d.A. $v_0 = u$.
         Dann hat $G_i - \{e', f'\}$ genau zwei ungerade Knoten: v_i, w.
         Also gibt es eine Zusammenhangskomponente, die genau einen ungeraden Knoten hat. Widerspruch.
   Damit ist die Behauptung gezeigt.

Zusammen ist also gezeigt: wenn e so gewählt wird, dass $G - e$ unzusammenhängend ist, dann ist v_i in $G - e$ ein isolierter Knoten. Insbesondere ist v_i nicht mehr in G_{i+1}.

Fleury: Korrektheitsbeweis (Ende)

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

Dann inzidieren e_1, e_i mit v_i.

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

   Dann inzidieren e_1, e_i mit v_i.

   Ferner gibt es für jedes weitere Durchlaufen von v_i stets zwei inzidente Kanten.

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

   Dann inzidieren e_1, e_i mit v_i.

   Ferner gibt es für jedes weitere Durchlaufen von v_i stets zwei inzidente Kanten.

   Also ist $\deg(v_0) = \deg(v_i)$ in W_i bzw. G_i stets gerade.

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

Dann inzidieren e_1, e_i mit v_i.

Ferner gibt es für jedes weitere Durchlaufen von v_i stets zwei inzidente Kanten.

Also ist $\deg(v_0) = \deg(v_i)$ in W_i bzw. G_i stets gerade.

1. Fall, $\deg(v_i) \geq 2$.

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

Dann inzidieren e_1, e_i mit v_i.

Ferner gibt es für jedes weitere Durchlaufen von v_i stets zwei inzidente Kanten.

Also ist $\deg(v_0) = \deg(v_i)$ in W_i bzw. G_i stets gerade.

1. Fall, $\deg(v_i) \geq 2$.

Analog zu Fall 1, Unterfall 2 kann gezeigt werden, dass keine der mit v_i inzidenten Kanten eine Brücke ist.

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

Dann inzidieren e_1, e_i mit v_i.

Ferner gibt es für jedes weitere Durchlaufen von v_i stets zwei inzidente Kanten.

Also ist $\deg(v_0) = \deg(v_i)$ in W_i bzw. G_i stets gerade.

1. Fall, $\deg(v_i) \geq 2$.

Analog zu Fall 1, Unterfall 2 kann gezeigt werden, dass keine der mit v_i inzidenten Kanten eine Brücke ist.

2. Fall, $\deg(v_i) = 0$.

Fleury: Korrektheitsbeweis (Ende)

2. Fall, $v_0 = v_i$.

Dann inzidieren e_1, e_i mit v_i.

Ferner gibt es für jedes weitere Durchlaufen von v_i stets zwei inzidente Kanten.

Also ist $\deg(v_0) = \deg(v_i)$ in W_i bzw. G_i stets gerade.

1. Fall, $\deg(v_i) \geq 2$.

Analog zu Fall 1, Unterfall 2 kann gezeigt werden, dass keine der mit v_i inzidenten Kanten eine Brücke ist.

2. Fall, $\deg(v_i) = 0$.

Wenn es keine Kanten gibt, die noch mit v_i inzidieren, dann sind – nach den obigen Ausführungen – alle Kanten in den Kantenzug einbezogen worden. Da wir ferner wieder am Ausgangspunkt angekommen sind, wurde eine Eulertour konstruiert.

Das Problem des chinesischen Briefträgers (CPP)

Das Problem des chinesischen Briefträgers (CPP)

Das Problem des chinesischen Briefträgers (CPP)

* Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.

Das Problem des chinesischen Briefträgers (CPP)

* Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.
* Die Tourlänge soll so gering wie möglich sein.

Das Problem des chinesischen Briefträgers (CPP)

* Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.
* Die Tourlänge soll so gering wie möglich sein.
* Dieses Problem wird als das „Problem des chinesischen Briefträgers“ bezeichnet, da es zuerst vom chinesischem Mathematiker Kuan (1962) beschrieben wurde.

Das Problem des chinesischen Briefträgers (CPP)

* Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.
* Die Tourlänge soll so gering wie möglich sein.
* Dieses Problem wird als das „Problem des chinesischen Briefträgers“ bezeichnet, da es zuerst vom chinesischem Mathematiker Kuan (1962) beschrieben wurde.
* Modell: In einem (positiv) bewerteten, zusammenhängenden Graphen $G = (V, E, c)$ entsprechen die Kanten den Straßen, die Knoten den Kreuzungen und die Gewichte den Längen der Straße zwischen den Kreuzungen.

Das Problem des chinesischen Briefträgers (CPP)

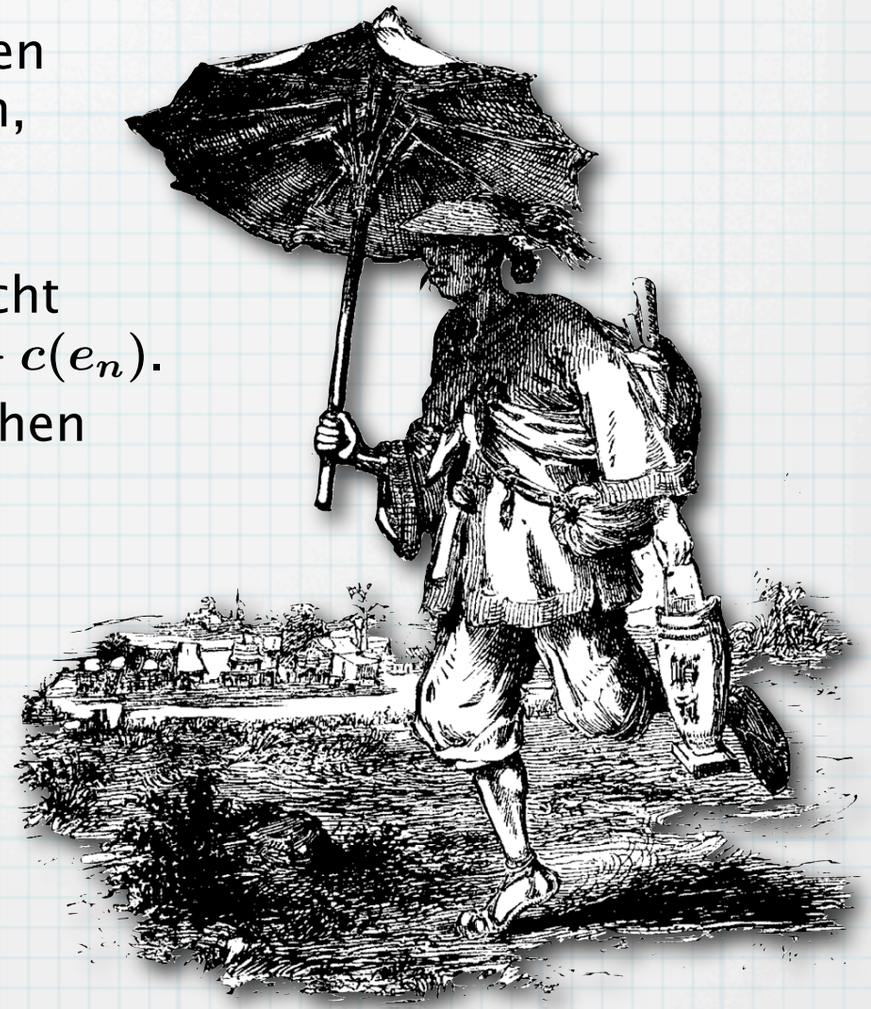
* Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.
* Die Tourlänge soll so gering wie möglich sein.
* Dieses Problem wird als das „Problem des chinesischen Briefträgers“ bezeichnet, da es zuerst vom chinesischem Mathematiker Kuan (1962) beschrieben wurde.
* Modell: In einem (positiv) bewerteten, zusammenhängenden Graphen $G = (V, E, c)$ entsprechen die Kanten den Straßen, die Knoten den Kreuzungen und die Gewichte den Längen der Straße zwischen den Kreuzungen.
* Die Bewertung einer Tour $(v_0, e_1, v_1, e_2, \ldots, e_n, v_0)$ entspricht der Bewertung der in ihr enthaltenen Kanten: $c(e_1) + \ldots + c(e_n)$.

Das Problem des chinesischen Briefträgers (CPP)

* Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.
* Die Tourlänge soll so gering wie möglich sein.
* Dieses Problem wird als das „Problem des chinesischen Briefträgers“ bezeichnet, da es zuerst vom chinesischem Mathematiker Kuan (1962) beschrieben wurde.
* Modell: In einem (positiv) bewerteten, zusammenhängenden Graphen $G = (V, E, c)$ entsprechen die Kanten den Straßen, die Knoten den Kreuzungen und die Gewichte den Längen der Straße zwischen den Kreuzungen.
* Die Bewertung einer Tour $(v_0, e_1, v_1, e_2, \ldots, e_n, v_0)$ entspricht der Bewertung der in ihr enthaltenen Kanten: $c(e_1) + \ldots + c(e_n)$.
* Das Briefträgerproblem besteht also darin, in diesem Graphen eine Tour minimaler Bewertung zu finden (per Definition enthält eine Tour jede Kante mindestens einmal).

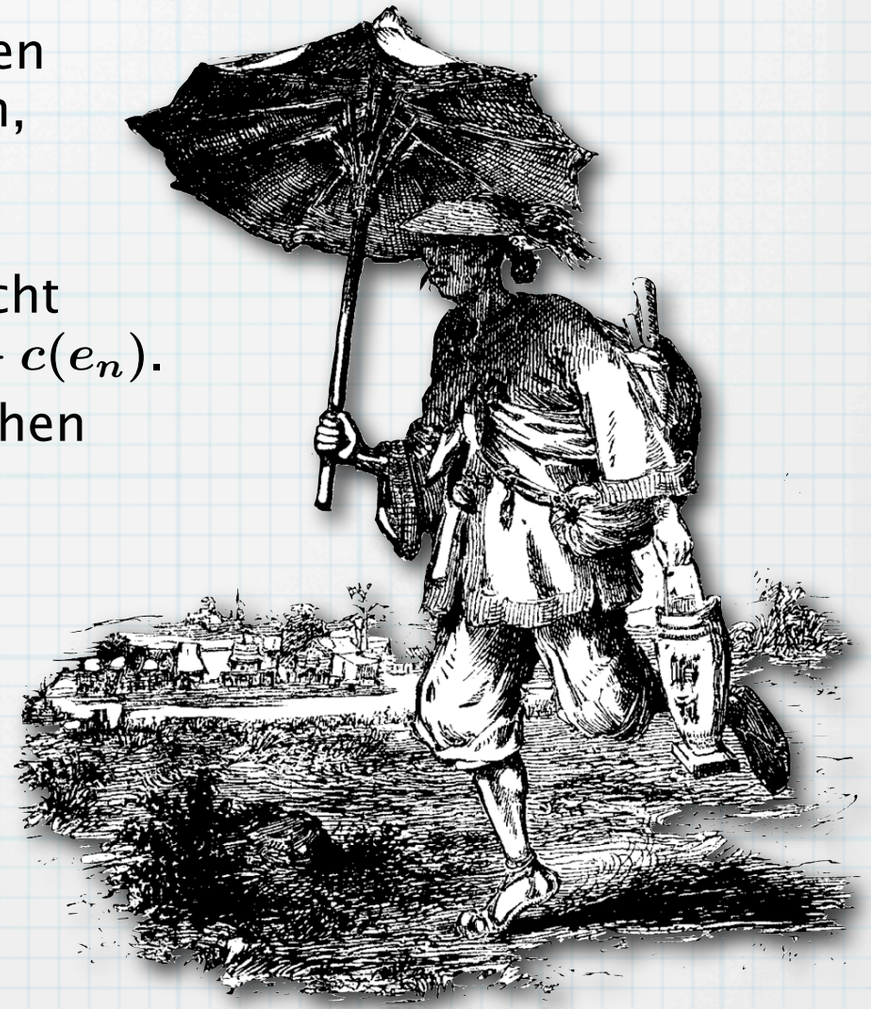
Das Problem des chinesischen Briefträgers (CPP)

* Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.
* Die Tourlänge soll so gering wie möglich sein.
* Dieses Problem wird als das „Problem des chinesischen Briefträgers“ bezeichnet, da es zuerst vom chinesischem Mathematiker Kuan (1962) beschrieben wurde.
* Modell: In einem (positiv) bewerteten, zusammenhängenden Graphen $G = (V, E, c)$ entsprechen die Kanten den Straßen, die Knoten den Kreuzungen und die Gewichte den Längen der Straße zwischen den Kreuzungen.
* Die Bewertung einer Tour $(v_0, e_1, v_1, e_2, \ldots, e_n, v_0)$ entspricht der Bewertung der in ihr enthaltenen Kanten: $c(e_1) + \ldots + c(e_n)$.
* Das Briefträgerproblem besteht also darin, in diesem Graphen eine Tour minimaler Bewertung zu finden (per Definition enthält eine Tour jede Kante mindestens einmal).
* Ist der Graph G Eulersch, dann ist jede Eulertour in G eine Tour mit minimaler Bewertung, da jede Kante in ihr genau einmal vorkommt. Die Tour kann praktisch mit dem Algorithmus von Fleury bestimmt werden.



Das Problem des chinesischen Briefträgers (CPP)

- Ein Briefträger startet seine Tour im Postamt. Danach läuft er alle Straßen seines Zustellbezirkes ab. Am Ende der Tour liefert er die nichtausgestellten Briefe wieder im Postamt ab.
- Die Tourlänge soll so gering wie möglich sein.
- Dieses Problem wird als das „Problem des chinesischen Briefträgers“ bezeichnet, da es zuerst vom chinesischem Mathematiker Kuan (1962) beschrieben wurde.
- Modell: In einem (positiv) bewerteten, zusammenhängenden Graphen $G = (V, E, c)$ entsprechen die Kanten den Straßen, die Knoten den Kreuzungen und die Gewichte den Längen der Straße zwischen den Kreuzungen.
- Die Bewertung einer Tour $(v_0, e_1, v_1, e_2, \ldots, e_n, v_0)$ entspricht der Bewertung der in ihr enthaltenen Kanten: $c(e_1) + \ldots + c(e_n)$.
- Das Briefträgerproblem besteht also darin, in diesem Graphen eine Tour minimaler Bewertung zu finden (per Definition enthält eine Tour jede Kante mindestens einmal).
- Ist der Graph G Eulersch, dann ist jede Eulertour in G eine Tour mit minimaler Bewertung, da jede Kante in ihr genau einmal vorkommt. Die Tour kann praktisch mit dem Algorithmus von Fleury bestimmt werden.
- Ist G nicht Eulersch, dann enthält jede Tour in G einige Kanten mehrfach. Diesen Fall werden wir im Folgenden genauer betrachten.



Der nicht-Eulersche Fall

Der nicht-Eulersche Fall

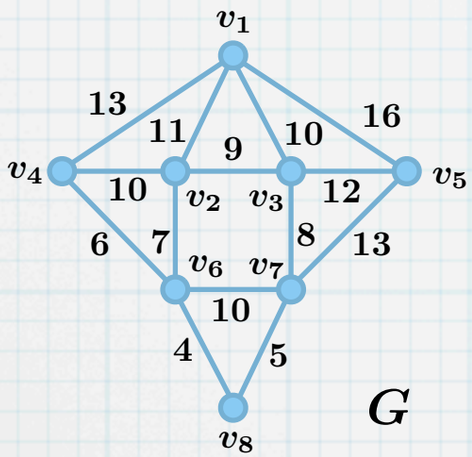
* **Definition 10**:
Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.

Der nicht-Eulersche Fall

* **Definition 10**:
  Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:

Der nicht-Eulersche Fall

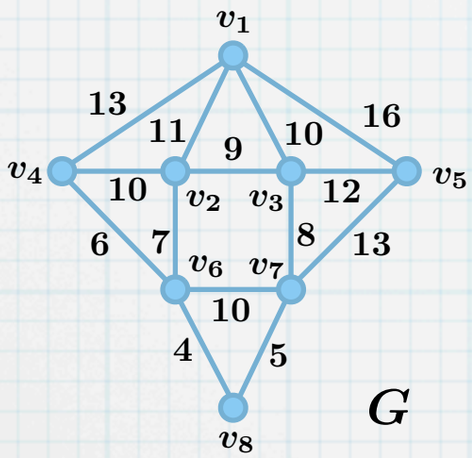
* **Definition 10**:
Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



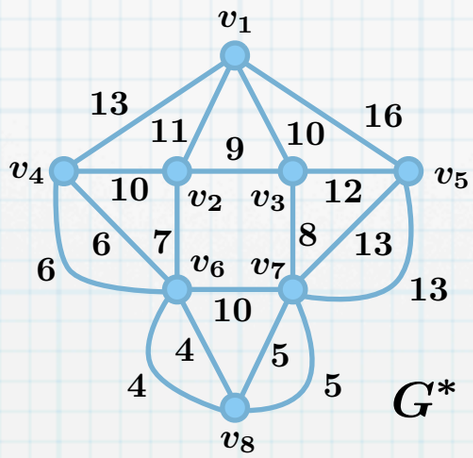
G

Der nicht-Eulersche Fall

* **Definition 10**:
  Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



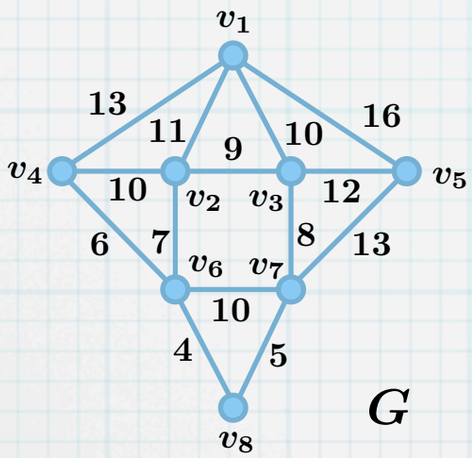
G



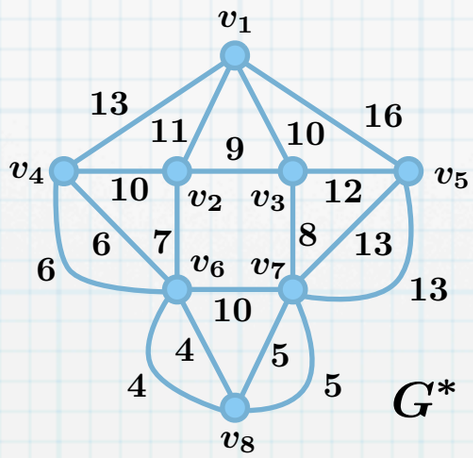
G^*

Der nicht-Eulersche Fall

* **Definition 10**:
Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



G

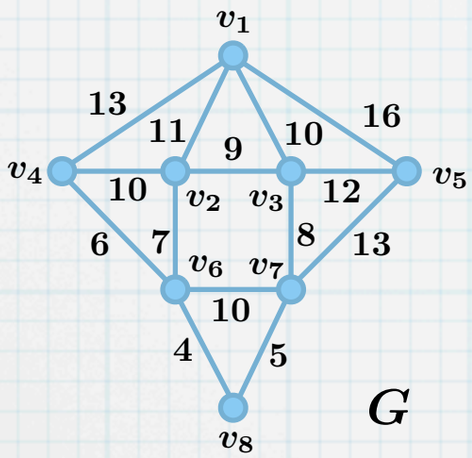


G^*

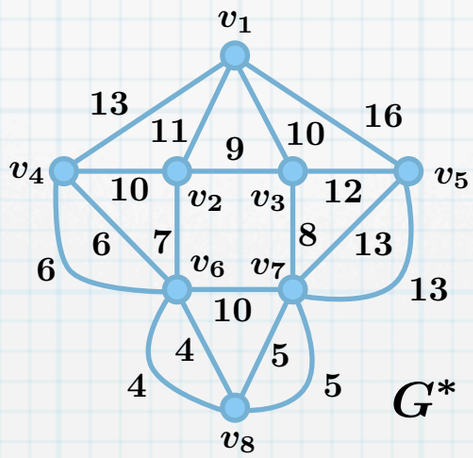
* Bemerkung: Eine optimale Tour von G, in der eine Kante n Mal wiederholt wird, entspricht einer Tour in G^*, bei der diese Kante $(n-1)$ Mal vervielfacht wurde.

Der nicht-Eulersche Fall

* **Definition 10**:
  Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



G

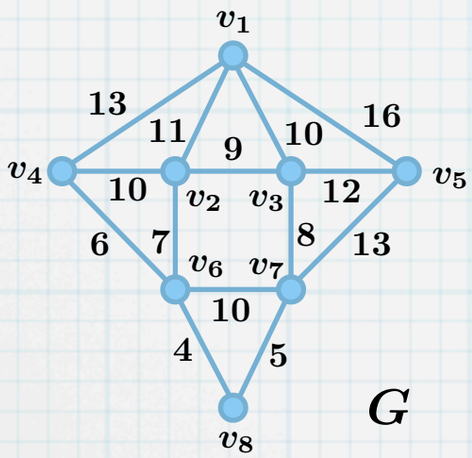


G^*

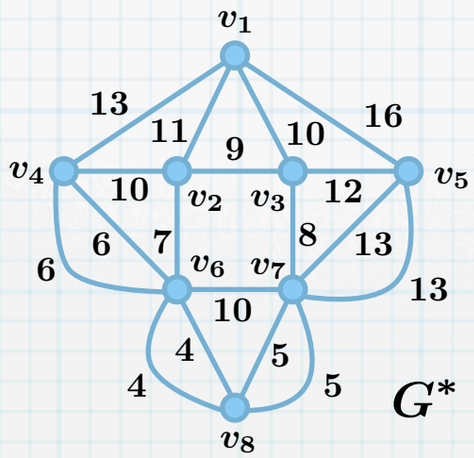
* Bemerkung: Eine optimale Tour von G, in der eine Kante n Mal wiederholt wird, entspricht einer Tour in G^*, bei der diese Kante $(n-1)$ Mal vervielfacht wurde.
* Damit kann das Problem des chinesischen Briefträgers wie folgt gelöst werden:

Der nicht-Eulersche Fall

* **Definition 10**:
  Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



G



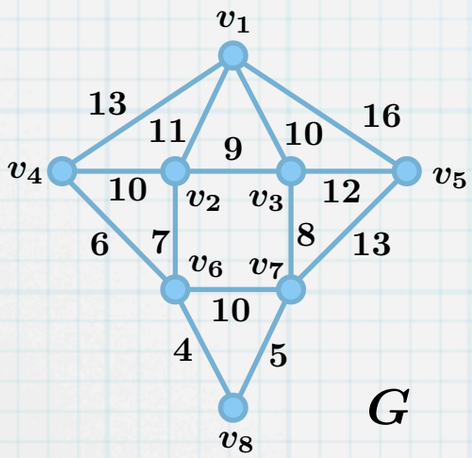
G^*

* Bemerkung: Eine optimale Tour von G, in der eine Kante n Mal wiederholt wird, entspricht einer Tour in G^*, bei der diese Kante $(n-1)$ Mal vervielfacht wurde.
* Damit kann das Problem des chinesischen Briefträgers wie folgt gelöst werden:

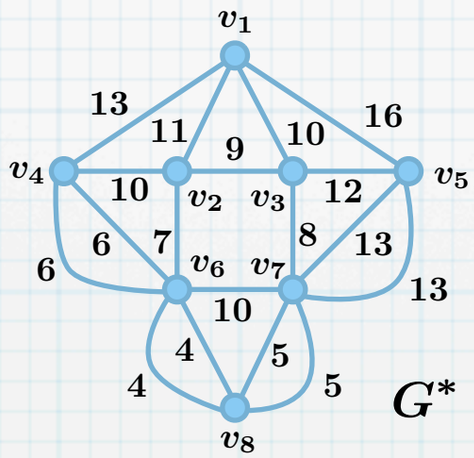
  Gegeben sei ein zusammenhängender, bewerteter Graph $G = (V, E, c)$ mit $c \geq 0$.

Der nicht-Eulersche Fall

* **Definition 10**:
  Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



G



G^*

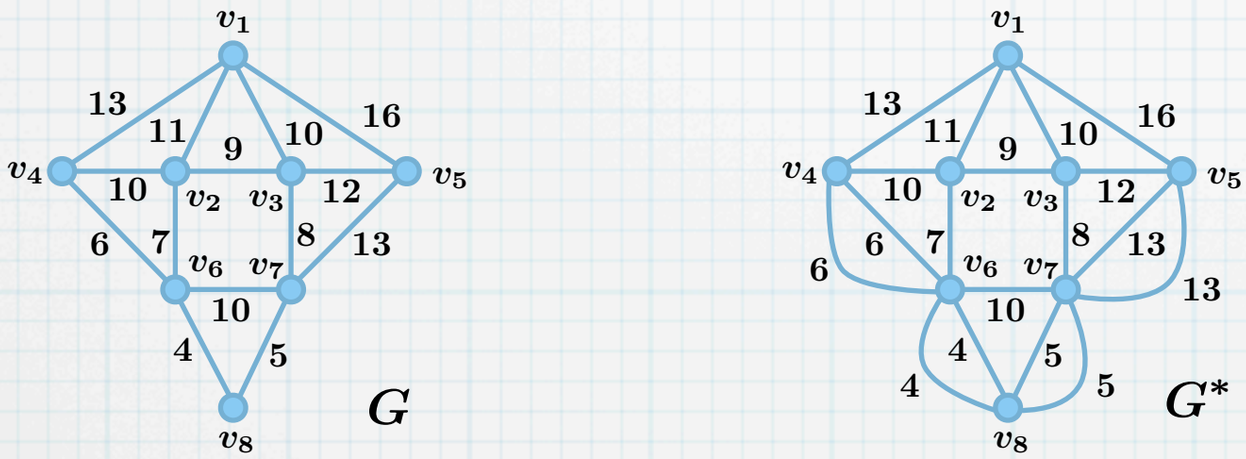
* Bemerkung: Eine optimale Tour von G, in der eine Kante n Mal wiederholt wird, entspricht einer Tour in G^*, bei der diese Kante $(n-1)$ Mal vervielfacht wurde.
* Damit kann das Problem des chinesischen Briefträgers wie folgt gelöst werden:

  Gegeben sei ein zusammenhängender, bewerteter Graph $G = (V, E, c)$ mit $c \geq 0$.

  Man ermittle, wenn nötig durch Kantenvervielfachung, einen bewerteten Eulerschen Obergraphen G^* von G, so dass die Summe der Bewertungen der vervielfachten Kanten minimal ist, d.h. $\sum_{e \in E(G^*) \setminus E(G)} c(e) \rightarrow \min$.

Der nicht-Eulersche Fall

* **Definition 10**:
  Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



* Bemerkung: Eine optimale Tour von G, in der eine Kante n Mal wiederholt wird, entspricht einer Tour in G^*, bei der diese Kante $(n-1)$ Mal vervielfacht wurde.
* Damit kann das Problem des chinesischen Briefträgers wie folgt gelöst werden:

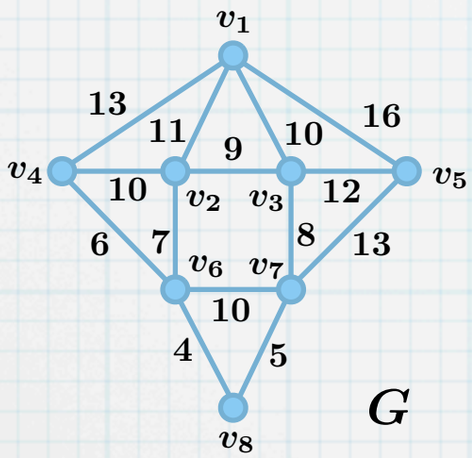
  Gegeben sei ein zusammenhängender, bewerteter Graph $G = (V, E, c)$ mit $c \geq 0$.

  Man ermittle, wenn nötig durch Kantenvervielfachung, einen bewerteten Eulerschen Obergraphen G^* von G, so dass die Summe der Bewertungen der vervielfachten Kanten minimal ist, d.h. $\sum_{e \in E(G^*) \setminus E(G)} c(e) \rightarrow \mathbf{min}$.

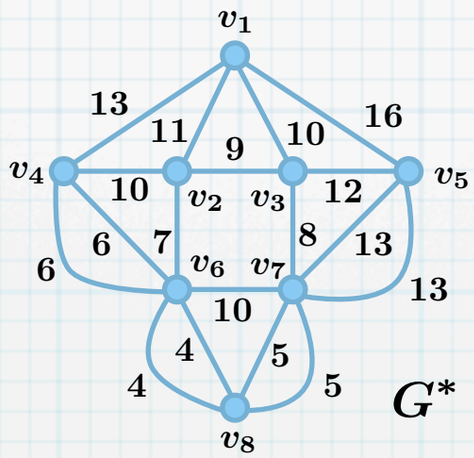
  In G^* bestimme man eine Eulertour.

Der nicht-Eulersche Fall

* **Definition 10**:
  Gegeben sei ein bewerteter Graph $G = (V, E, c)$. Wir erzeugen aus G einen Obergraphen G^*, indem wir einige (oder alle) Kanten **vervielfachen.** Das heißt: ist $e \in E$, so führen wir eine (oder mehrere) Kopien von e ein, welche die gleichen Endknoten und die gleiche Bewertung $c(e)$ wie e haben.
* Beispiel:



G



G^*

* Bemerkung: Eine optimale Tour von G, in der eine Kante n Mal wiederholt wird, entspricht einer Tour in G^*, bei der diese Kante $(n-1)$ Mal vervielfacht wurde.
* Damit kann das Problem des chinesischen Briefträgers wie folgt gelöst werden:

  Gegeben sei ein zusammenhängender, bewerteter Graph $G = (V, E, c)$ mit $c \geq 0$.

  Man ermittle, wenn nötig durch Kantenvervielfachung, einen bewerteten Eulerschen Obergraphen G^* von G, so dass die Summe der Bewertungen der vervielfachten Kanten minimal ist, d.h. $\sum_{e \in E(G^*) \setminus E(G)} c(e) \rightarrow \mathbf{min}$.

  In G^* bestimme man eine Eulertour.
* Nachfolgend diskutieren wir einen Spezialfall.

Touren in Graphen mit genau zwei ungeraden Knoten

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
* Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
* Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
* Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
* Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
* Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
* Dieses wiederhole man nun für $v_2, v_3, \ldots$

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
* Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
* Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
* Dieses wiederhole man nun für $v_2, v_3, \ldots$
* Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.

Touren in Graphen mit genau zwei ungeraden Knoten

- Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
- Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
- Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
- Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
- Dieses wiederhole man nun für $v_2, v_3, \ldots$
- Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
- Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
* Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
* Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
* Dieses wiederhole man nun für $v_2, v_3, \ldots$
* Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
* Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
* Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \to \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
* Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
* Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
* Dieses wiederhole man nun für $v_2, v_3, \ldots$
* Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
* Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
* Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \to \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.
* Also ergibt sich folgender Algorithmus:

Touren in Graphen mit genau zwei ungeraden Knoten

* Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
* Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
* Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
* Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
* Dieses wiederhole man nun für $v_2, v_3, \ldots$
* Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
* Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
* Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \to \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.
* Also ergibt sich folgender Algorithmus:

  Bestimme einen kürzesten u–w–Weg in G (z.B. mittels Dijkstras Algorithmus).

Touren in Graphen mit genau zwei ungeraden Knoten

- Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
- Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
- Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
- Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
- Dieses wiederhole man nun für $v_2, v_3, \ldots$
- Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
- Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
- Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \to \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.
- Also ergibt sich folgender Algorithmus:

  Bestimme einen kürzesten u–w–Weg in G (z.B. mittels Dijkstras Algorithmus).

  Verdopple die Kanten in diesem u–w–Weg. Der so erhaltene Graph G^* ist Eulersch.

Touren in Graphen mit genau zwei ungeraden Knoten

- Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
- Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
- Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
- Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
- Dieses wiederhole man nun für $v_2, v_3, \ldots$
- Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
- Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
- Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \to \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.
- Also ergibt sich folgender Algorithmus:

  Bestimme einen kürzesten u–w–Weg in G (z.B. mittels Dijkstras Algorithmus).

  Verdopple die Kanten in diesem u–w–Weg. Der so erhaltene Graph G^* ist Eulersch.

  Bestimme eine Eulertour in G^* (z.B. mittels des Algorithmus von Fleury).

Touren in Graphen mit genau zwei ungeraden Knoten

- Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
- Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
- Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
- Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
- Dieses wiederhole man nun für $v_2, v_3, \ldots$
- Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
- Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
- Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \to \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.
- Also ergibt sich folgender Algorithmus:

  Bestimme einen kürzesten u–w–Weg in G (z.B. mittels Dijkstras Algorithmus).

  Verdopple die Kanten in diesem u–w–Weg. Der so erhaltene Graph G^* ist Eulersch.

  Bestimme eine Eulertour in G^* (z.B. mittels des Algorithmus von Fleury).
- Beispiel:

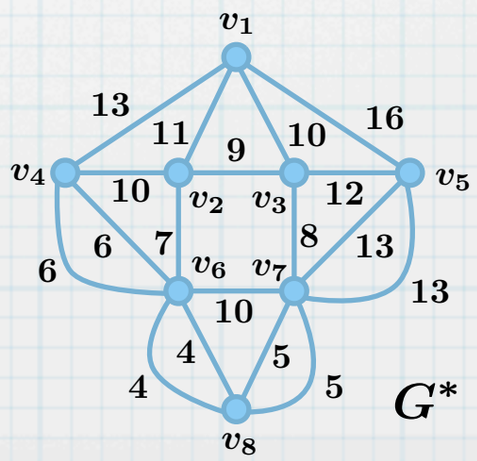
Touren in Graphen mit genau zwei ungeraden Knoten

- Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
- Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
- Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
- Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
- Dieses wiederhole man nun für $v_2, v_3, \ldots$
- Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
- Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
- Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \to \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.
- Also ergibt sich folgender Algorithmus:

  Bestimme einen kürzesten u–w–Weg in G (z.B. mittels Dijkstras Algorithmus).

  Verdopple die Kanten in diesem u–w–Weg. Der so erhaltene Graph G^* ist Eulersch.

  Bestimme eine Eulertour in G^* (z.B. mittels des Algorithmus von Fleury).
- Beispiel:



Touren in Graphen mit genau zwei ungeraden Knoten

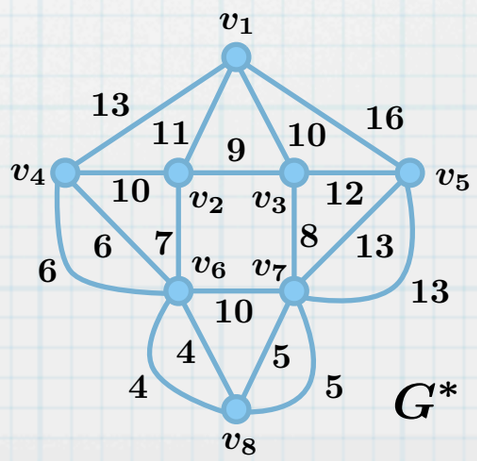
- Der Graph $G = (V, E, c)$ habe genau zwei Knoten u und w mit ungeradem Grad.
- Damit G^* Eulersch ist, muss es eine Kante $\{u, v_1\}$ geben, die eine verdoppelte Kante von G ist (denn nur so kann der Grad von u in G^* gerade sein).
- Ist $v_1 \neq w$, so ist durch diese Verdopplung der Grad von v_1 zunächst ungerade (da er vorher in G gerade war).
- Es muss also eine verdoppelte Kante $\{v_1, v_2\}$ geben, damit der Grad von v_1 in G^* gerade ist.
- Dieses wiederhole man nun für $v_2, v_3, \ldots$
- Nach endlich vielen Schritten erreicht man $v_k = w$, welches durch die verdoppelte Kante $\{v_{k-1}, v_k\}$ jetzt einen geraden Grad in G^* hat.
- Die Knotenfolge $(u, v_1, v_2, \ldots, v_{k-1}, w)$ ist ein u–w–Kantenzug.
- Damit dieser Weg der Bedingung $\sum_{e \in E(G^*) \setminus E(G)} c(e) \rightarrow \min$ genügt, muss dieser Kantenzug ein kürzester u–w–Weg sein.
- Also ergibt sich folgender Algorithmus:

  Bestimme einen kürzesten u–w–Weg in G (z.B. mittels Dijkstras Algorithmus).

  Verdopple die Kanten in diesem u–w–Weg. Der so erhaltene Graph G^* ist Eulersch.

  Bestimme eine Eulertour in G^* (z.B. mittels des Algorithmus von Fleury).
- Beispiel:

  Die Knotenfolge
  (1,4,6,8,7,5,1,2,4,6,8,7,5,3,2,6,7,3,1)
  ist eine Eulertour in G^* und
  damit eine Lösung des Briefträgerproblems
  in G mit der Tourlänge 162.



Hamiltonsche Wege und Kreise

Hamiltonsche Wege und Kreise

* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.

Hamiltonsche Wege und Kreise

* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.

Hamiltonsche Wege und Kreise

* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.

Hamiltonsche Wege und Kreise

* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
* Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.

Hamiltonsche Wege und Kreise

* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
* Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
* Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).

Hamiltonsche Wege und Kreise

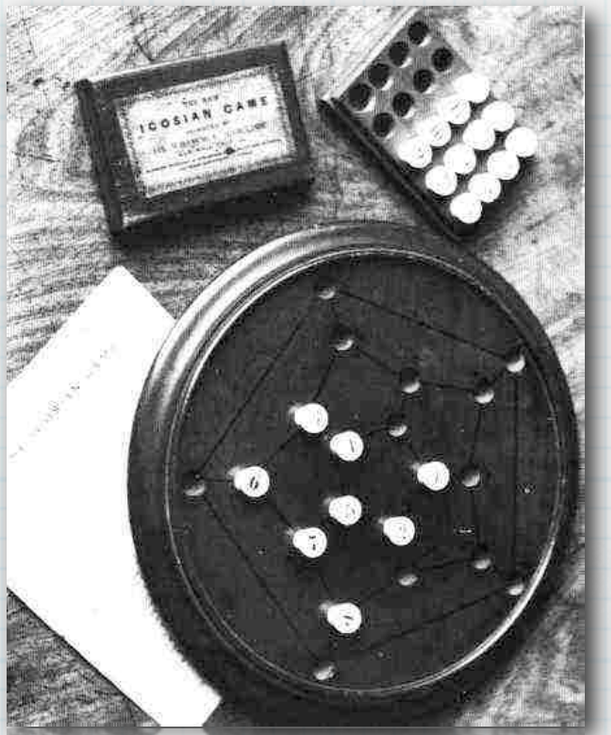
* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
* Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
* Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).

Hamiltonsche Wege und Kreise

* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
* Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
* Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).
* Er erfand ein Spiel, bei dem 20 Knoten eines Dodekaeders zu einer Rundreise verbunden werden sollten, die Tour jeden Punkt genau einmal beinhaltet,
  und das Tourende dem Ausgangspunkt entspricht.

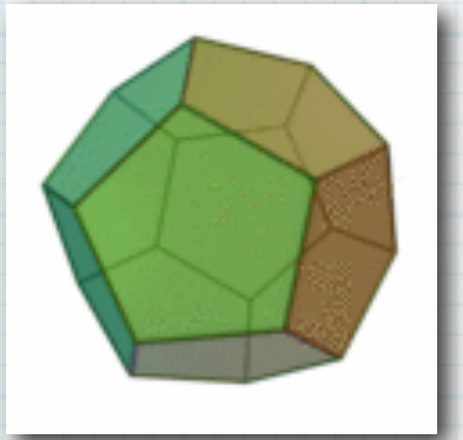
Hamiltonsche Wege und Kreise

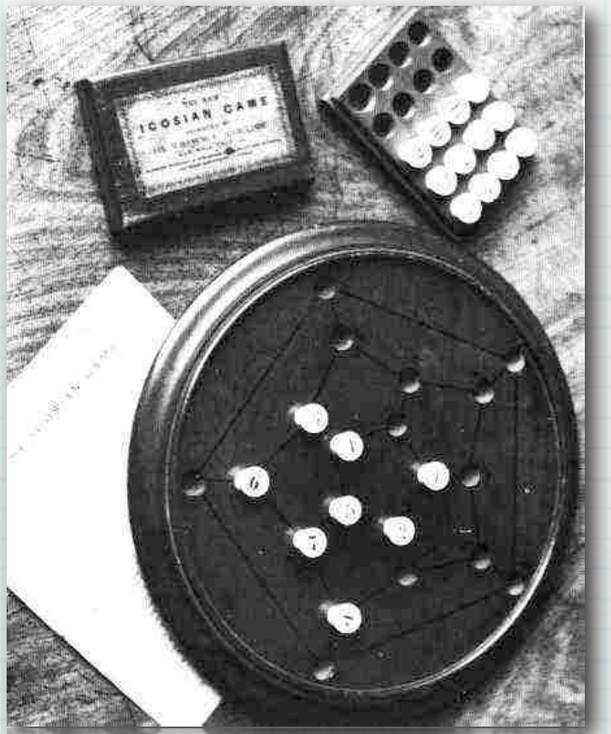
- **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
- Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
- **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
- Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
- Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).
- Er erfand ein Spiel, bei dem 20 Knoten eines Dodekaeders zu einer Rundreise verbunden werden sollten, die Tour jeden Punkt genau einmal beinhaltet, und das Tourende dem Ausgangspunkt entspricht.



Hamiltonsche Wege und Kreise

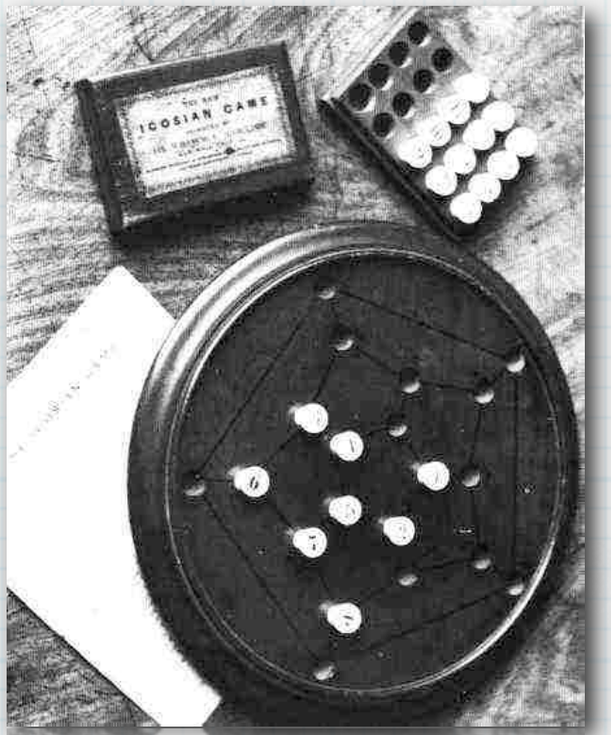
* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
* Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
* Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).
* Er erfand ein Spiel, bei dem 20 Knoten eines Dodekaeders zu einer Rundreise verbunden werden sollten, die Tour jeden Punkt genau einmal beinhaltet, und das Tourende dem Ausgangspunkt entspricht.





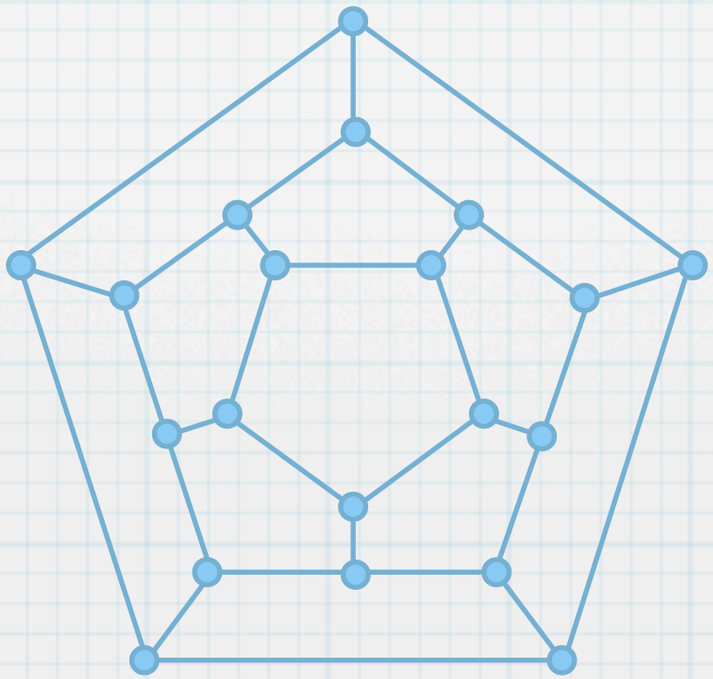
Hamiltonsche Wege und Kreise

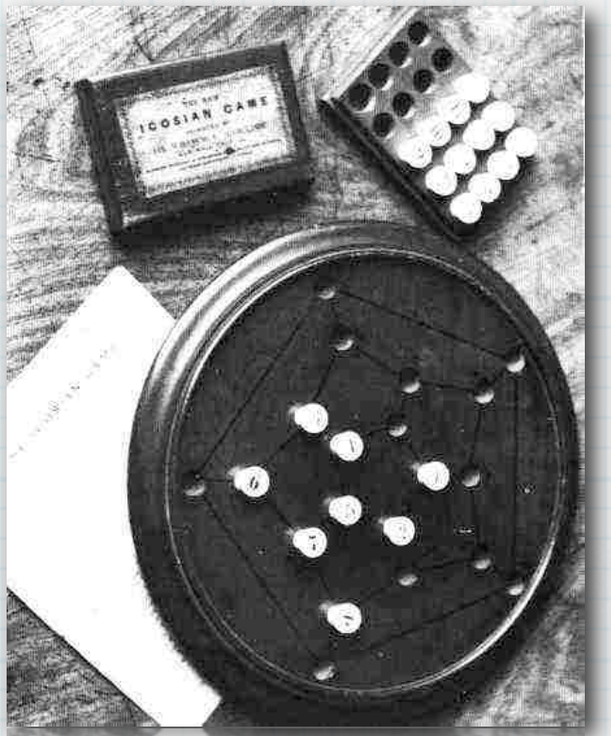
- **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
- Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
- **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
- Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
- Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).
- Er erfand ein Spiel, bei dem 20 Knoten eines Dodekaeders zu einer Rundreise verbunden werden sollten, die Tour jeden Punkt genau einmal beinhaltet, und das Tourende dem Ausgangspunkt entspricht.



Hamiltonsche Wege und Kreise

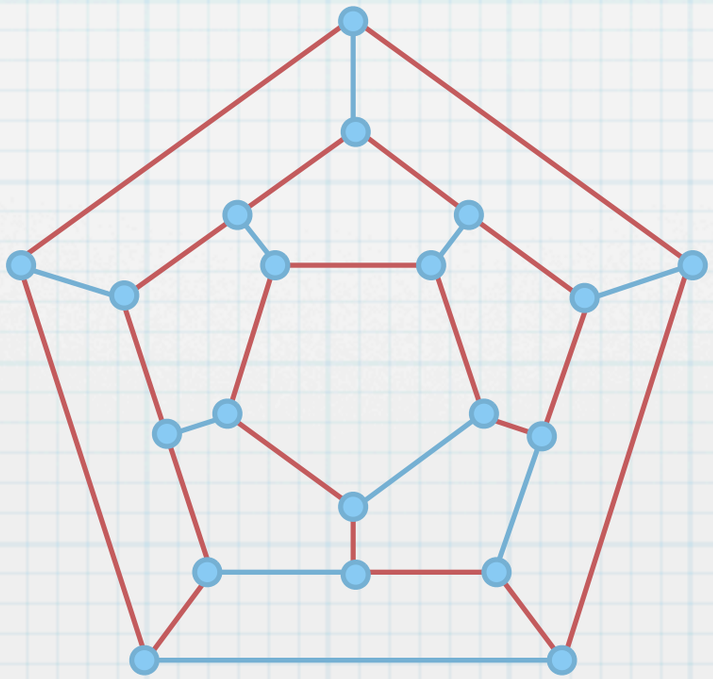
* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
* Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
* Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).
* Er erfand ein Spiel, bei dem 20 Knoten eines Dodekaeders zu einer Rundreise verbunden werden sollten, die Tour jeden Punkt genau einmal beinhaltet, und das Tourende dem Ausgangspunkt entspricht.

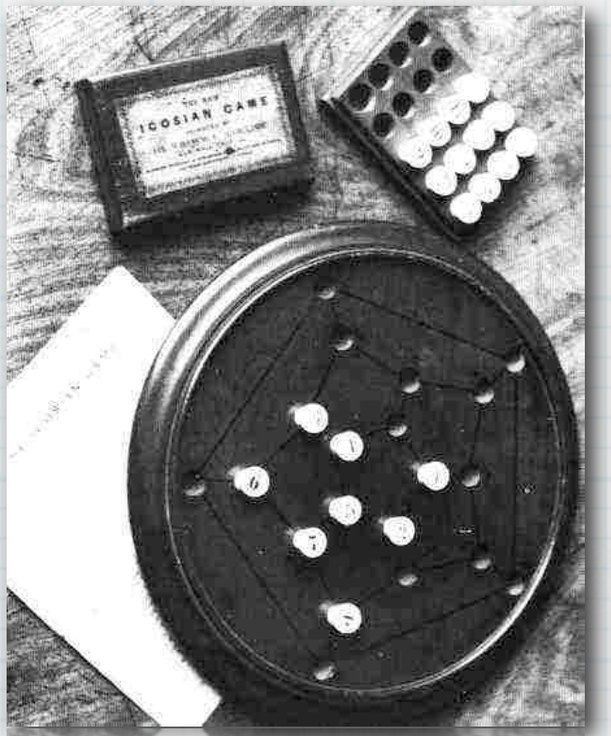




Hamiltonsche Wege und Kreise

* **Definition 11**:
  Ein **Hamiltonscher Weg** im Graphen G ist ein einfacher Weg, der jeden Knoten von G enthält.
* Bemerkung: Gemäß der Definition des einfachen Weges werden keine Knoten wiederholt. Also enthält ein Hamiltonscher Weg jeden Knoten von G genau einmal.
* **Definition 12**:
  Ein **Hamiltonscher Zyklus** (oder **Hamiltonscher Kreis**) in einem Graphen G ist ein Kreis, der jeden Knoten von G enthält.
* Bemerkung: Da abgesehen vom Endknoten kein Knoten wiederholt wird, enthält auch ein Hamiltonscher Zyklus jeden Knoten von G genau einmal.
* Hamiltonsche Graphen sind benannt nach dem Iren Sir William Hamilton (1805–1865).
* Er erfand ein Spiel, bei dem 20 Knoten eines Dodekaeders zu einer Rundreise verbunden werden sollten, die Tour jeden Punkt genau einmal beinhaltet, und das Tourende dem Ausgangspunkt entspricht.





Maximale nicht-Hamiltonsche Graphen

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

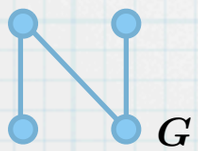
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

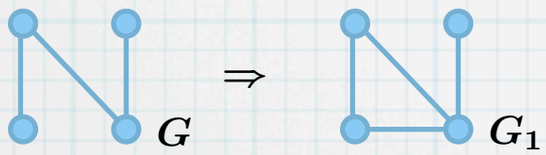
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

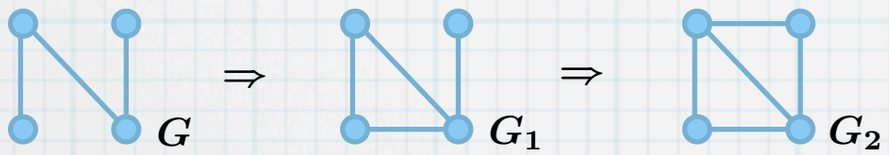
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

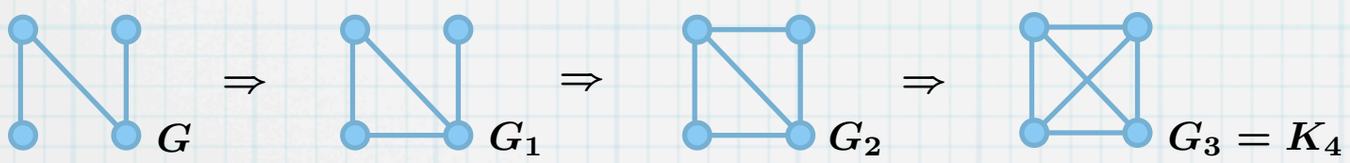
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

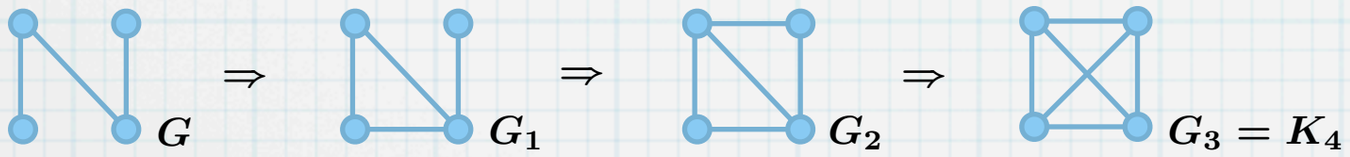
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



* Ist G nicht-Hamiltonsch, dann wird, da K_n Hamiltonsch ist, in irgendeinem Schritt aus einem nicht-Hamiltonschen Graphen ein Hamiltonscher. Alle weiteren Graphen bleiben Hamiltonsch.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

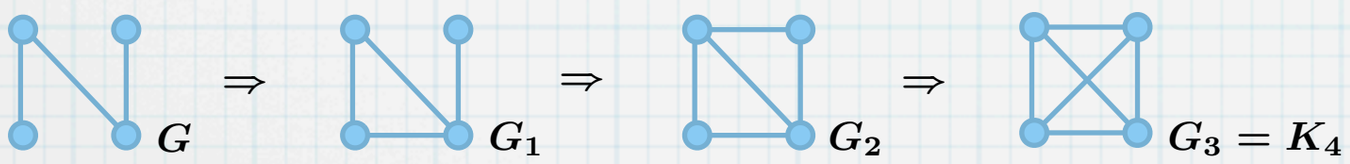
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



* Ist G nicht-Hamiltonsch, dann wird, da K_n Hamiltonsch ist, in irgendeinem Schritt aus einem nicht-Hamiltonschen Graphen ein Hamiltonscher. Alle weiteren Graphen bleiben Hamiltonsch.

  Im obigen Beispiel ist G_1 nicht-Hamiltonsch, G_2 ist Hamiltonsch.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

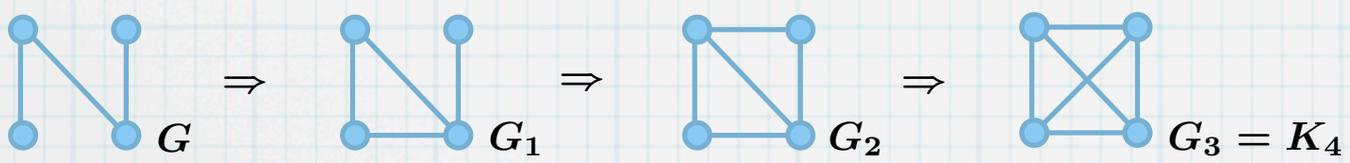
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



* Ist G nicht-Hamiltonsch, dann wird, da K_n Hamiltonsch ist, in irgendeinem Schritt aus einem nicht-Hamiltonschen Graphen ein Hamiltonscher. Alle weiteren Graphen bleiben Hamiltonsch.

  Im obigen Beispiel ist G_1 nicht-Hamiltonsch, G_2 ist Hamiltonsch.
* **Definition 13**:
  Ein schlichter Graph heißt **maximal nicht-Hamiltonsch**, wenn er nicht Hamiltonsch ist, aber durch Hinzufügen einer beliebigen Kante, die zwei nichtadjazente Knoten verbindet, ein Hamiltonscher Graph wird.

Maximale nicht-Hamiltonsche Graphen

* Beispiel: Für alle n ist der K_n ein Hamiltonscher Graph.

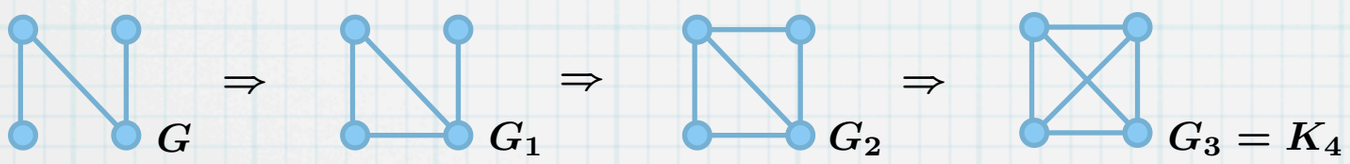
  Beweis: Offensichtlich ist ein Zyklus auf n Knoten (und n Kanten) Hamiltonsch.

  Ein Obergraph eines Hamiltonschen Graphen ist stets Hamiltonsch, da der Hamiltonsche Zyklus auch im Obergraphen existiert.

  Da der K_n ein Obergraph eines Kreises auf n Knoten ist, folgt die Behauptung.
* Bemerkung: Ein nicht-schlichter Graph mit mind. 3 Knoten ist genau dann Hamiltonsch, wenn ein unterliegender schlichter Graph Hamiltonsch ist.

  Daher betrachtet man üblicherweise nur die Hamiltonschen Eigenschaften schlichter Graphen.
* Gegeben sei ein schlichter Hamiltonscher Graph G mit n Knoten.

  Da G Untergraph des K_n ist, können wir durch schrittweises Hinzufügen von Kanten G in den K_n überführen.
* Beispiel:



* Ist G nicht-Hamiltonsch, dann wird, da K_n Hamiltonsch ist, in irgendeinem Schritt aus einem nicht-Hamiltonschen Graphen ein Hamiltonscher. Alle weiteren Graphen bleiben Hamiltonsch.

  Im obigen Beispiel ist G_1 nicht-Hamiltonsch, G_2 ist Hamiltonsch.
* **Definition 13**:
  Ein schlichter Graph heißt **maximal nicht-Hamiltonsch**, wenn er nicht Hamiltonsch ist, aber durch Hinzufügen einer beliebigen Kante, die zwei nichtadjazente Knoten verbindet, ein Hamiltonscher Graph wird.
* Beispiel: G_1 ist maximal nicht-Hamiltonsch.

Satz von Dirac

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

  H ist Hamiltonsch, da N maximal nicht-Hamiltonsch ist.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

  H ist Hamiltonsch, da N maximal nicht-Hamiltonsch ist.

  Sei $C = (v_1, v_2, \ldots, v_n, v_1)$ mit $v_1 = u$ und $v_n = v$ ein Hamiltonscher Zyklus in H.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

  H ist Hamiltonsch, da N maximal nicht-Hamiltonsch ist.

  Sei $C = (v_1, v_2, \ldots, v_n, v_1)$ mit $v_1 = u$ und $v_n = v$ ein Hamiltonscher Zyklus in H.

  Die Kante $\{u, v\}$ ist Teil von C (da es andernfalls bereits einen Hamiltonschen Zyklus in N gegeben hätte).

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

  H ist Hamiltonsch, da N maximal nicht-Hamiltonsch ist.

  Sei $C = (v_1, v_2, \ldots, v_n, v_1)$ mit $v_1 = u$ und $v_n = v$ ein Hamiltonscher Zyklus in H.

  Die Kante $\{u, v\}$ ist Teil von C (da es andernfalls bereits einen Hamiltonschen Zyklus in N gegeben hätte).

  Sei $S := \{v_i \in C : \{u, v_{i+1}\} \in E\}, T := \{v_i \in C : \{v, v_i\} \in E\}$.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

  H ist Hamiltonsch, da N maximal nicht-Hamiltonsch ist.

  Sei $C = (v_1, v_2, \ldots, v_n, v_1)$ mit $v_1 = u$ und $v_n = v$ ein Hamiltonscher Zyklus in H.

  Die Kante $\{u, v\}$ ist Teil von C (da es andernfalls bereits einen Hamiltonschen Zyklus in N gegeben hätte).

  Sei $S := \{v_i \in C : \{u, v_{i+1}\} \in E\}, T := \{v_i \in C : \{v, v_i\} \in E\}$.

  Dann ist $v_n \notin T$, denn andernfalls gibt es eine Kante von v nach $v_n = v$, also eine Schlinge. Aber N ist schlicht.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

  H ist Hamiltonsch, da N maximal nicht-Hamiltonsch ist.

  Sei $C = (v_1, v_2, \ldots, v_n, v_1)$ mit $v_1 = u$ und $v_n = v$ ein Hamiltonscher Zyklus in H.

  Die Kante $\{u, v\}$ ist Teil von C (da es andernfalls bereits einen Hamiltonschen Zyklus in N gegeben hätte).

  Sei $S := \{v_i \in C : \{u, v_{i+1}\} \in E\}, T := \{v_i \in C : \{v, v_i\} \in E\}$.

  Dann ist $v_n \notin T$, denn andernfalls gibt es eine Kante von v nach $v_n = v$, also eine Schlinge. Aber N ist schlicht.

  Ferner ist $v_n \notin S$ (wobei v_{n+1} als v_1 zu lesen ist), denn anderfalls gibt es eine Kante von u nach $v_{n+1} = v_1 = u$, also wiederum eine Schlinge. Also ist $v_n \notin S \cup T$.

Satz von Dirac

* **Satz 14** (Dirac, 1952):
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten mit $\deg(v) \geq n/2$ für alle $v \in V(G)$. Dann ist G Hamiltonsch.
* Beweis:

  Angenommen, die Aussage ist falsch.

  Dann gibt es für alle $n \geq 3$ einen nicht-Hamiltonschen Graphen mit Knotengraden von mindestens $n/2$ für alle Knoten.

  In jedem Obergraphen, der durch Hinzufügen weiterer Kanten auf der selben Knotenmenge entsteht, ist der Knotengrad ebenfalls mindestens $n/2$. Insbesondere gilt das für den maximalen nicht-Hamiltonschen Obergraphen $N = (V, E)$.

  N ist nicht vollständig, da K_V (der vollständige Graph auf V) Hamiltonsch ist.

  Daher gibt es zwei nichtadjazente Knoten $u, v \in V$.

  Sei $H := N + \{u, v\}$ der Obergraph, der durch Hinzufügen der Kante $\{u, v\}$ zu N entsteht.

  H ist Hamiltonsch, da N maximal nicht-Hamiltonsch ist.

  Sei $C = (v_1, v_2, \ldots, v_n, v_1)$ mit $v_1 = u$ und $v_n = v$ ein Hamiltonscher Zyklus in H.

  Die Kante $\{u, v\}$ ist Teil von C (da es andernfalls bereits einen Hamiltonschen Zyklus in N gegeben hätte).

  Sei $S := \{v_i \in C : \{u, v_{i+1}\} \in E\}, T := \{v_i \in C : \{v, v_i\} \in E\}$.

  Dann ist $v_n \notin T$, denn andernfalls gibt es eine Kante von v nach $v_n = v$, also eine Schlinge. Aber N ist schlicht.

  Ferner ist $v_n \notin S$ (wobei v_{n+1} als v_1 zu lesen ist), denn anderfalls gibt es eine Kante von u nach $v_{n+1} = v_1 = u$, also wiederum eine Schlinge. Also ist $v_n \notin S \cup T$.

  Daher gilt $|S \cup T| < n$.

Satz von Dirac (Forts.)

Satz von Dirac (Forts.)

* Beweis (Forts.):

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

    Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

  Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

  Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

  Widerspruch, da N nicht-Hamiltonsch ist.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

  Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

  Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

    Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

    Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

    Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

    Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Folglich ist $|S \cup T| = |S| + |T| - |S \cap T|$

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

  Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

  Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Folglich ist $|S \cup T| = |S| + |T| - |S \cap T| = |S| + |T|$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

  Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

  Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Folglich ist $|S \cup T| = |S| + |T| - |S \cap T| = |S| + |T|$.

Deshalb können wir abschätzen: $\deg(u) + \deg(v) = |S| + |T|$

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

    Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

    Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

    Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

    Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Folglich ist $|S \cup T| = |S| + |T| - |S \cap T| = |S| + |T|$.

Deshalb können wir abschätzen: $\deg(u) + \deg(v) = |S| + |T| = |S \cup T|$

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

  Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

  Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Folglich ist $|S \cup T| = |S| + |T| - |S \cap T| = |S| + |T|$.

Deshalb können wir abschätzen: $\deg(u) + \deg(v) = |S| + |T| = |S \cup T| < n$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

  Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

  Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

  Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

  Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Folglich ist $|S \cup T| = |S| + |T| - |S \cap T| = |S| + |T|$.

Deshalb können wir abschätzen: $\deg(u) + \deg(v) = |S| + |T| = |S \cup T| < n$.

Widerspruch, da $\deg(u) \geq n/2$ und $\deg(v) \geq n/2$, also $\deg(u) + \deg(v) \geq n$.

Satz von Dirac (Forts.)

* Beweis (Forts.):

  Zu jeder mit u inzidenten Kante gehört genau ein Knoten $v_i \in S$.

  Also gilt $|S| = \deg(u)$.

  Zu jeder mit v inzidenten Kante gehört genau ein Knoten $v_i \in T$.

  Also gilt analog $|T| = \deg(v)$.

  Angenommen, es gibt einen Knoten $v_k \in S \cap T$.

    Da $v_k \in S$, gibt es eine Kante $e = \{u, v_{k+1}\} = \{v_1, v_{k+1}\}$.

    Da $v_k \in T$, gibt es eine Kante $f = \{v, v_k\} = \{v_n, v_k\}$.

    Dann ist $C' = (v_1, v_{k+1}, v_{k+2}, \ldots, v_n, v_k, v_{k-1}, \ldots, v_2, v_1\}$ ein Hamiltonscher Zyklus in N.

    Widerspruch, da N nicht-Hamiltonsch ist.

Also ist $S \cap T = \emptyset$.

Folglich ist $|S \cup T| = |S| + |T| - |S \cap T| = |S| + |T|$.

Deshalb können wir abschätzen: $\deg(u) + \deg(v) = |S| + |T| = |S \cup T| < n$.

Widerspruch, da $\deg(u) \geq n/2$ und $\deg(v) \geq n/2$, also $\deg(u) + \deg(v) \geq n$.

Also war die eingangs getroffene Annahme falsch, und der Satz ist richtig.

Hülle eines Graphen

Hülle eines Graphen

* **Korollar 15**:
  Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.

Hülle eines Graphen

* **Korollar 15**:
  Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:

Hülle eines Graphen

* **Korollar 15**:
  Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

Hülle eines Graphen

* **Korollar 15**:
  Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

Hülle eines Graphen

* **Korollar 15**:
  Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.

Hülle eines Graphen

* **Korollar 15**:
  Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.

  Angenommen, G ist nicht Hamiltonsch.

Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.

  Angenommen, G ist nicht Hamiltonsch.

  Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.

Hülle eines Graphen

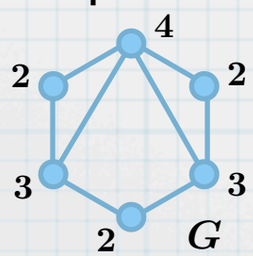
* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
  ($\Rightarrow$): Sei G Hamiltonsch.
    Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
  ($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
    Angenommen, G ist nicht Hamiltonsch.
    Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.

Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
  ($\Rightarrow$): Sei G Hamiltonsch.
    Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
  ($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
    Angenommen, G ist nicht Hamiltonsch.
    Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:

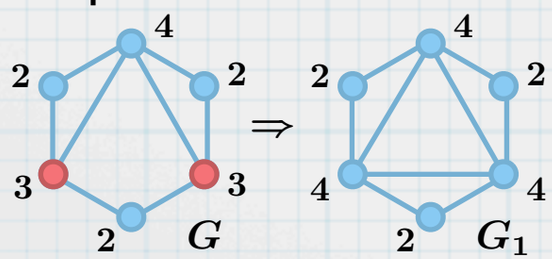
Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
  ($\Rightarrow$): Sei G Hamiltonsch.
    Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
  ($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
    Angenommen, G ist nicht Hamiltonsch.
    Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



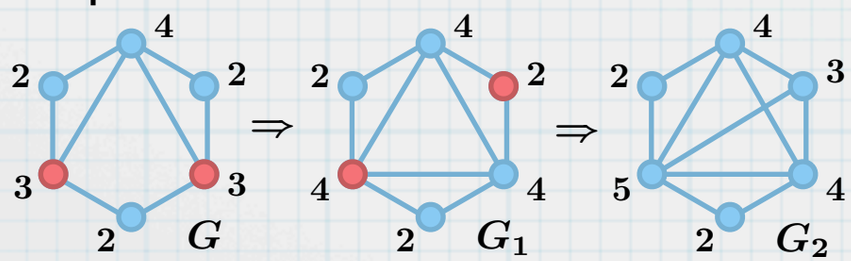
Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
  ($\Rightarrow$): Sei G Hamiltonsch.
    Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
  ($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
    Angenommen, G ist nicht Hamiltonsch.
    Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



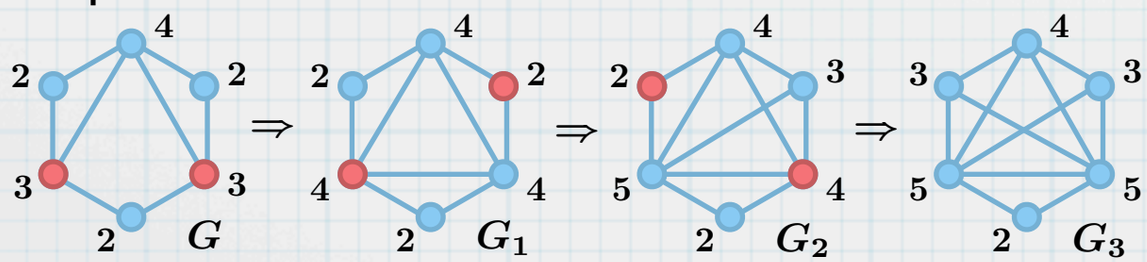
Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
($\Rightarrow$): Sei G Hamiltonsch.
Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
Angenommen, G ist nicht Hamiltonsch.
Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



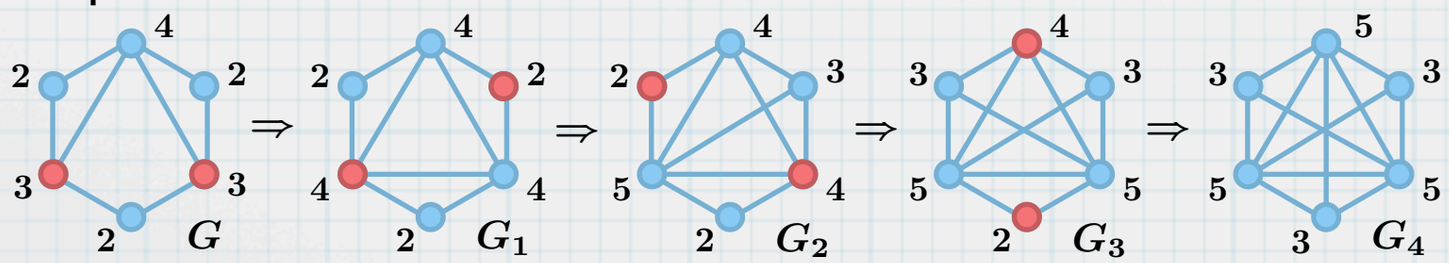
Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
($\Rightarrow$): Sei G Hamiltonsch.
Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
Angenommen, G ist nicht Hamiltonsch.
Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



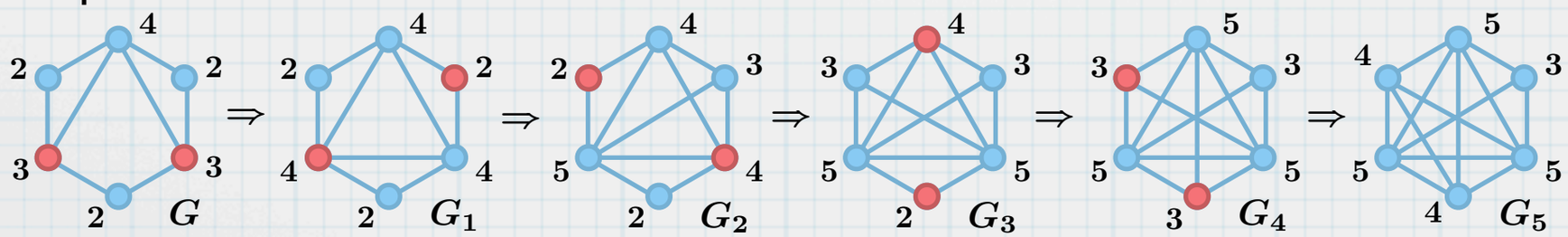
Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
($\Rightarrow$): Sei G Hamiltonsch.
Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
Angenommen, G ist nicht Hamiltonsch.
Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



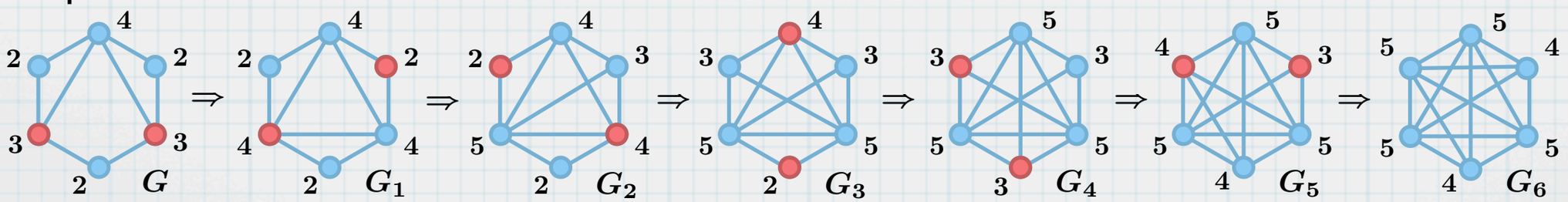
Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
($\Rightarrow$): Sei G Hamiltonsch.
Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
Angenommen, G ist nicht Hamiltonsch.
Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



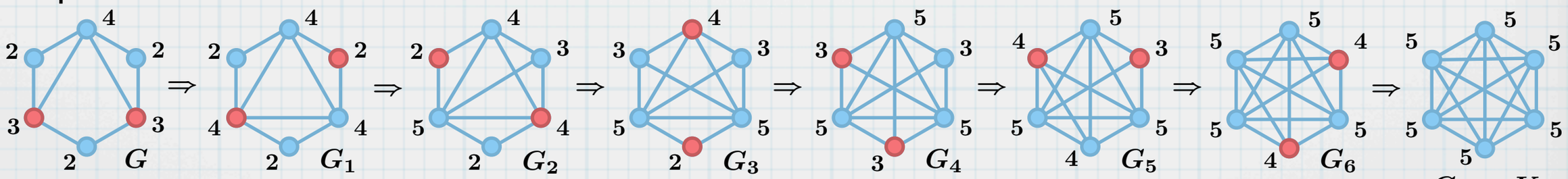
Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
($\Rightarrow$): Sei G Hamiltonsch.
Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
Angenommen, G ist nicht Hamiltonsch.
Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



Hülle eines Graphen

* **Korollar 15**:
Sei G ein schlichter Graph mit n Knoten. Seien u, v zwei nichtadjazente Knoten mit $\deg(u) + \deg(v) \geq n$. Sei $G + \{u, v\}$ der schlichte Obergraph von G, der aus G durch Hinzufügen der Kante zwischen u und v entsteht. Dann ist $G + \{u, v\}$ genau dann Hamiltonsch, wenn G Hamiltonsch ist.
* Beweis:
($\Rightarrow$): Sei G Hamiltonsch.
Da $G + \{u, v\}$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.
($\Leftarrow$): Sei $G + \{u, v\}$ Hamiltonsch.
Angenommen, G ist nicht Hamiltonsch.
Dann gilt (siehe Beweis von Satz 14) $\deg(u) + \deg(v) < n$. Widerspruch.
* **Definition 16**:
Sei G ein schlichter Graph. Gibt es in G zwei nichtadjazente Knoten u_1, v_1 mit $\deg(u_1) + \deg(v_1) \geq n$, bilde den (schlichten) Obergraphen $G_1 := G + \{u_1, v_1\}$. Wenn es in G_1 zwei nichtadjazente Knoten u_2, v_2 mit $\deg(u_2) + \deg(v_2) \geq n$ gibt, bilde man den (schlichten) Obergraphen $G_2 := G_1 + \{u_2, v_2\}$. Auf diese Weise bilde man nun eine Sequenz $G_1, G_2, G_3, \ldots$ von Graphen, wobei G_{i+1} Obergraph von G_i ist, bis es kein geeignetes Knotenpaar mehr gibt. Der letzte Obergraph, den wir dadurch erhalten haben, wird **Hülle** von G genannt und mit $\mathcal{H}(G)$ bezeichnet.
* Beispiel:



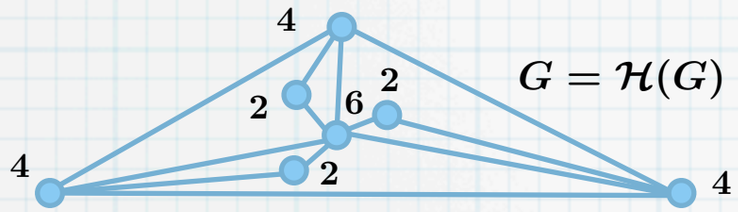
Satz von Bondy und Chvatal

Satz von Bondy und Chvatal

* Beispiel:

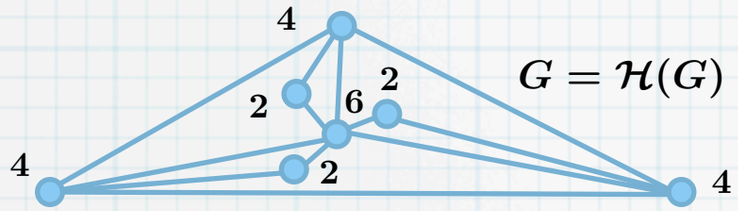
Satz von Bondy und Chvatal

* Beispiel:



Satz von Bondy und Chvatal

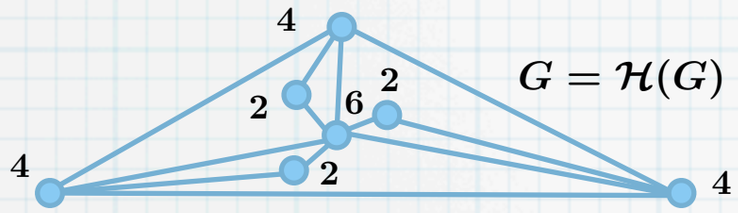
* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.

Satz von Bondy und Chvatal

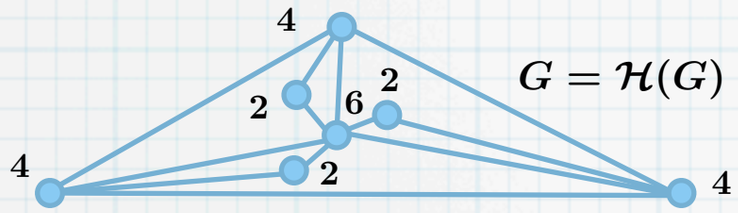
* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

Satz von Bondy und Chvatal

* Beispiel:

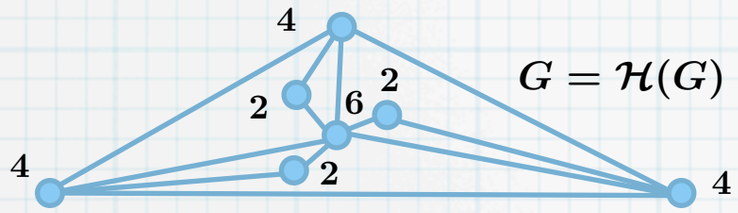


* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

Satz von Bondy und Chvatal

* Beispiel:



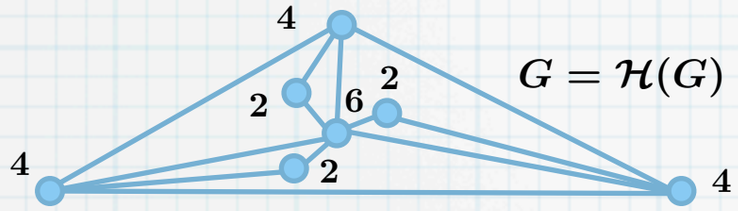
* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

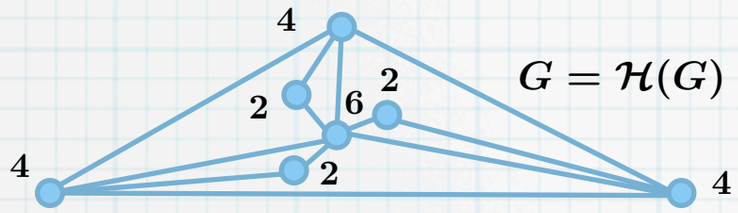
  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

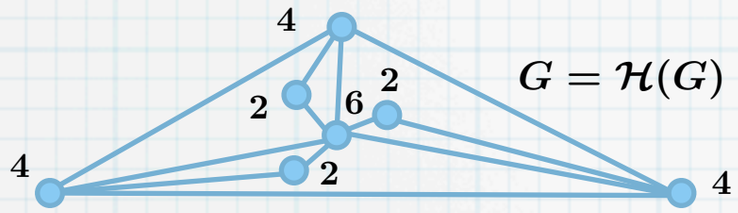
  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

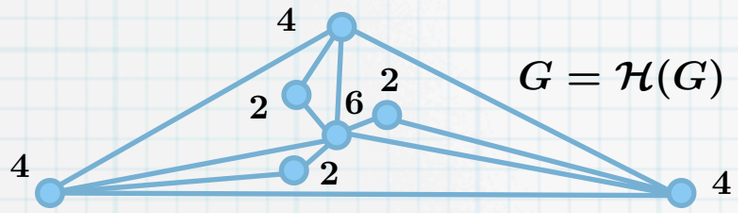
  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

  $G_k = \mathcal{H}(G)$ entstand durch Hinzufügen einer Kante $\{u, v\}$ von nichtadjazenten Knoten u, v aus G_{k-1}, also: $G_k := G_{k-1} + \{u, v\}$, wobei $\deg(u) + \deg(v) \geq n$.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

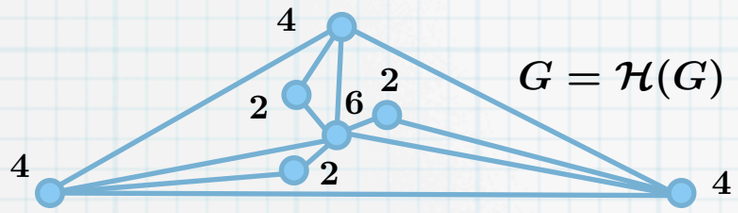
  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

  $G_k = \mathcal{H}(G)$ entstand durch Hinzufügen einer Kante $\{u, v\}$ von nichtadjazenten Knoten u, v aus G_{k-1}, also: $G_k := G_{k-1} + \{u, v\}$, wobei $\deg(u) + \deg(v) \geq n$.

  Nach Korollar 15 ist G_{k-1} daher Hamiltonsch.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

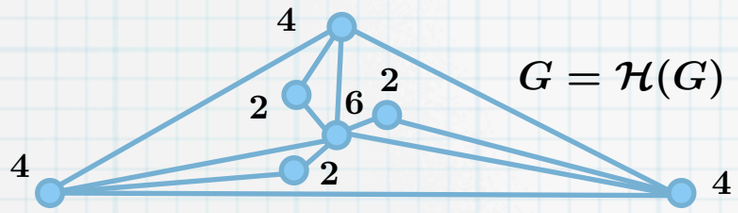
  $G_k = \mathcal{H}(G)$ entstand durch Hinzufügen einer Kante $\{u, v\}$ von nichtadjazenten Knoten u, v aus G_{k-1}, also: $G_k := G_{k-1} + \{u, v\}$, wobei $\deg(u) + \deg(v) \geq n$.

  Nach Korollar 15 ist G_{k-1} daher Hamiltonsch.

  Analog argumentiert man nun für $G_{k-2}, G_{k-3}, \ldots, G_1, G$.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

  $G_k = \mathcal{H}(G)$ entstand durch Hinzufügen einer Kante $\{u, v\}$ von nichtadjazenten Knoten u, v aus G_{k-1}, also: $G_k := G_{k-1} + \{u, v\}$, wobei $\deg(u) + \deg(v) \geq n$.

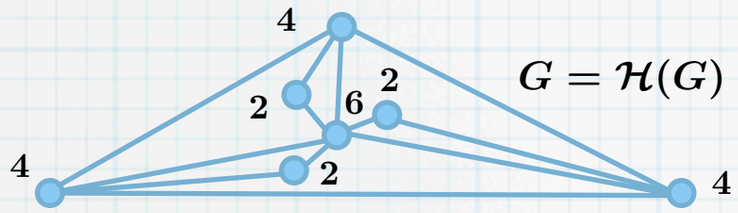
  Nach Korollar 15 ist G_{k-1} daher Hamiltonsch.

  Analog argumentiert man nun für $G_{k-2}, G_{k-3}, \ldots, G_1, G$.

  Somit muss letztendlich G auch Hamiltonsch sein.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

  $G_k = \mathcal{H}(G)$ entstand durch Hinzufügen einer Kante $\{u, v\}$ von nichtadjazenten Knoten u, v aus G_{k-1}, also: $G_k := G_{k-1} + \{u, v\}$, wobei $\deg(u) + \deg(v) \geq n$.

  Nach Korollar 15 ist G_{k-1} daher Hamiltonsch.

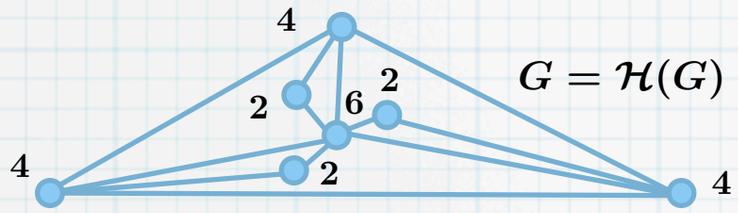
  Analog argumentiert man nun für $G_{k-2}, G_{k-3}, \ldots, G_1, G$.

  Somit muss letztendlich G auch Hamiltonsch sein.

* **Korollar 18**:
Sei G ein schlichter Graph mit $n \geq 3$ Knoten. Ist $\mathcal{H}(G)$ vollständig, $\mathcal{H}(G) = K_n$, dann ist G Hamiltonsch.

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
  Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

  $G_k = \mathcal{H}(G)$ entstand durch Hinzufügen einer Kante $\{u, v\}$ von nichtadjazenten Knoten u, v aus G_{k-1}, also: $G_k := G_{k-1} + \{u, v\}$, wobei $\deg(u) + \deg(v) \geq n$.

  Nach Korollar 15 ist G_{k-1} daher Hamiltonsch.

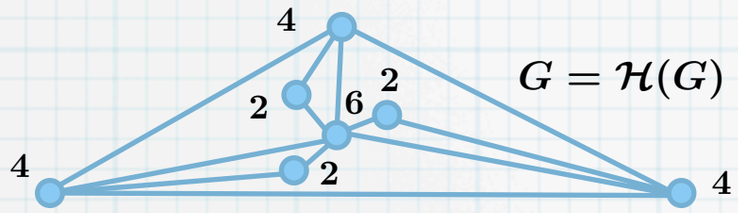
  Analog argumentiert man nun für $G_{k-2}, G_{k-3}, \ldots, G_1, G$.

  Somit muss letztendlich G auch Hamiltonsch sein.

* **Korollar 18**:
  Sei G ein schlichter Graph mit $n \geq 3$ Knoten. Ist $\mathcal{H}(G)$ vollständig, $\mathcal{H}(G) = K_n$, dann ist G Hamiltonsch.
* Beweis:

Satz von Bondy und Chvatal

* Beispiel:



* **Satz 17** (Bondy, Chvatal, 1976):
Ein schlichter Graph G ist genau dann Hamiltonsch, wenn seine Hülle $\mathcal{H}(G)$ Hamiltonsch ist.
* Beweis:

  ($\Rightarrow$): Sei G Hamiltonsch.

  Da $\mathcal{H}(G)$ ein Obergraph von G ist, ist dieser auch Hamiltonsch.

  ($\Leftarrow$): Sei $\mathcal{H}(G)$ Hamiltonsch.

  Sei $G, G_1, G_2, \ldots, G_{k-1}, G_k = \mathcal{H}(G)$ eine Folge der konstruierten Graphen.

  $G_k = \mathcal{H}(G)$ entstand durch Hinzufügen einer Kante $\{u, v\}$ von nichtadjazenten Knoten u, v aus G_{k-1}, also: $G_k := G_{k-1} + \{u, v\}$, wobei $\deg(u) + \deg(v) \geq n$.

  Nach Korollar 15 ist G_{k-1} daher Hamiltonsch.

  Analog argumentiert man nun für $G_{k-2}, G_{k-3}, \ldots, G_1, G$.

  Somit muss letztendlich G auch Hamiltonsch sein.

* **Korollar 18**:
Sei G ein schlichter Graph mit $n \geq 3$ Knoten. Ist $\mathcal{H}(G)$ vollständig, $\mathcal{H}(G) = K_n$, dann ist G Hamiltonsch.
* Beweis:

  Siehe Satz 17 und bemerke, dass jeder vollständige Graph Hamiltonsch ist.

Nicht-Hamiltonsche ebene Graphen

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.

Nicht-Hamiltonsche ebene Graphen

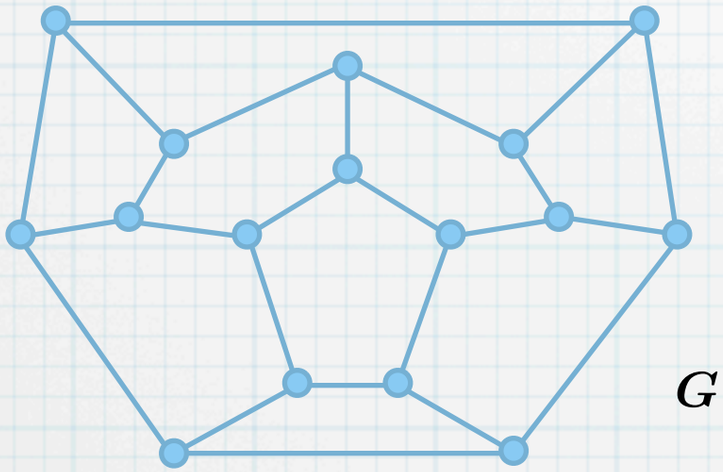
* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.

  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.

  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:

Nicht-Hamiltonsche ebene Graphen

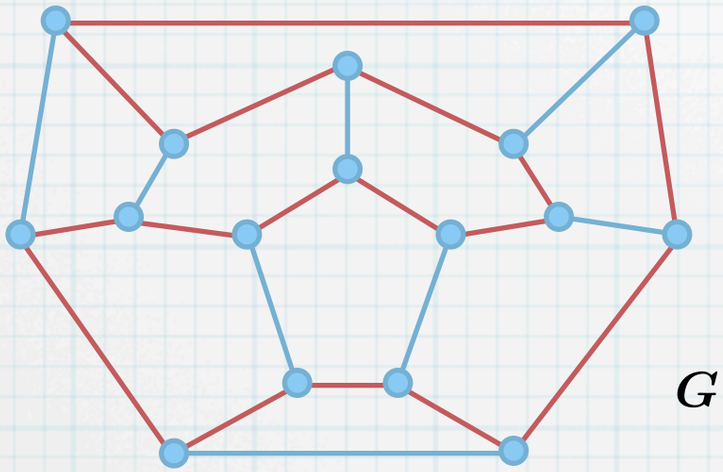
* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:



G

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:



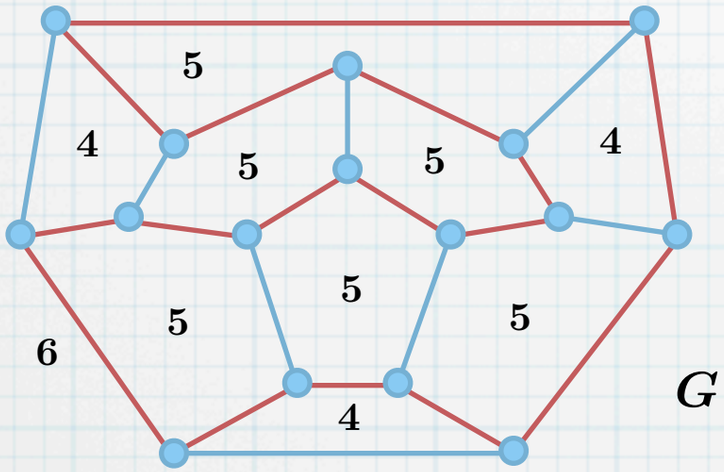
G

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.

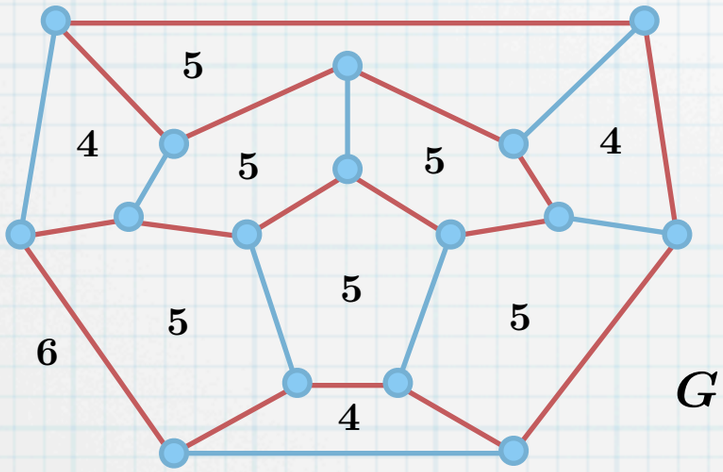
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.

  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:



Nicht-Hamiltonsche ebene Graphen

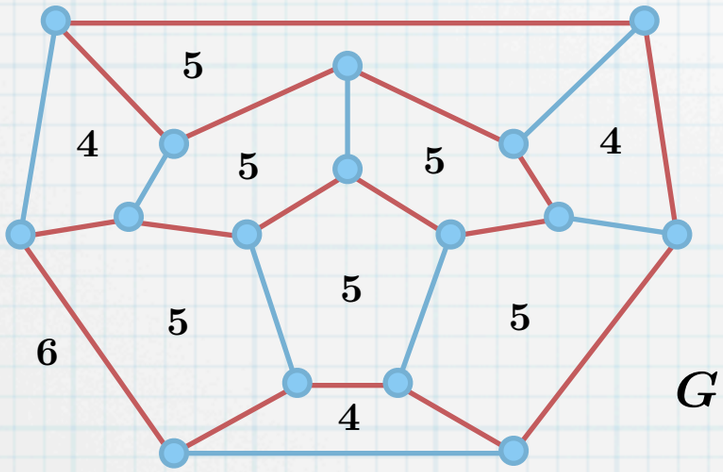
* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:



$$\alpha_4 = 1,\ \beta_4 = 2$$

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:

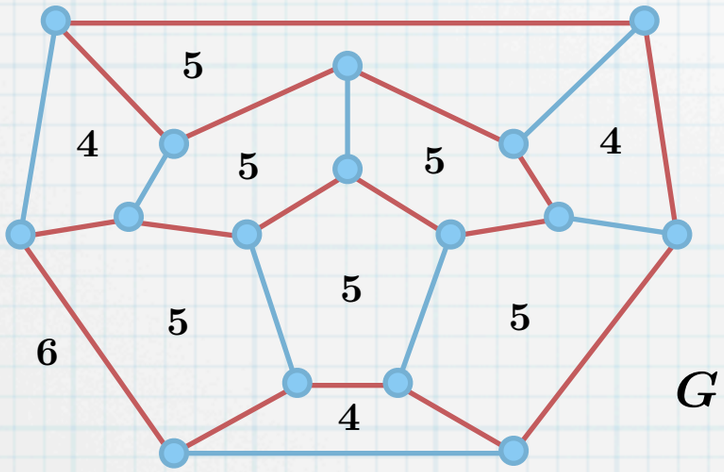


$$\alpha_4 = 1,\ \beta_4 = 2$$

$$\alpha_5 = 4,\ \beta_5 = 2$$

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:



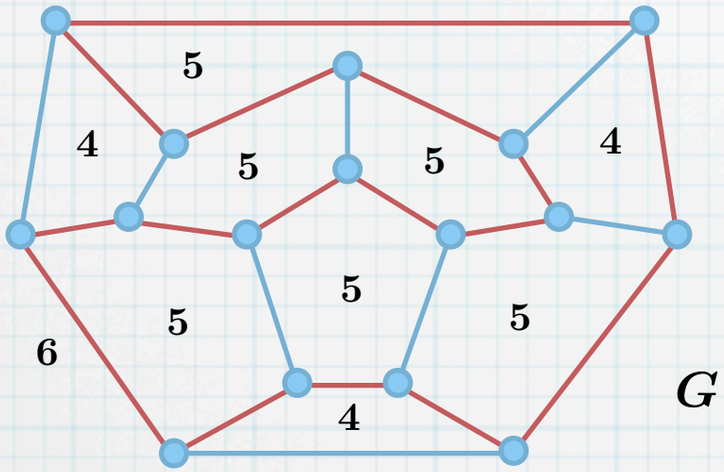
$$\alpha_4 = 1,\ \beta_4 = 2$$

$$\alpha_5 = 4,\ \beta_5 = 2$$

$$\alpha_6 = 0,\ \beta_6 = 1$$

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:

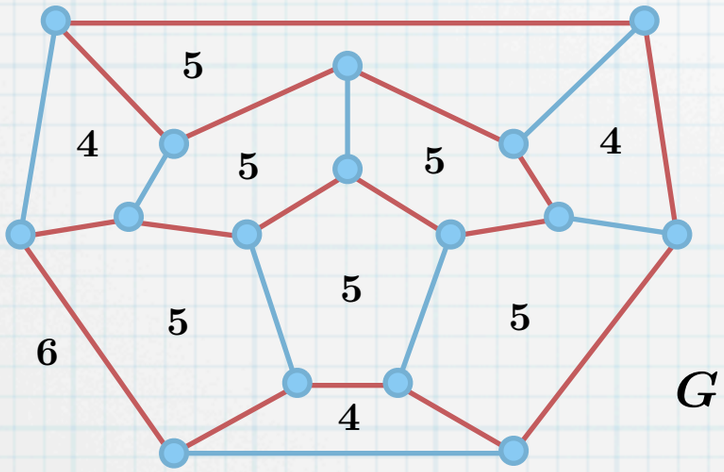


$$\alpha_4 = 1,\ \beta_4 = 2$$
$$\alpha_5 = 4,\ \beta_5 = 2$$
$$\alpha_6 = 0,\ \beta_6 = 1$$

* **Definition 20**:
  Sei G ein ebener Hamiltonscher Graph ohne Schlingen, und sei C ein Hamiltonscher Zyklus in G. Eine **Sehne** bezüglich C ist eine Kante in G, die dem Zyklus C nicht angehört.

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:

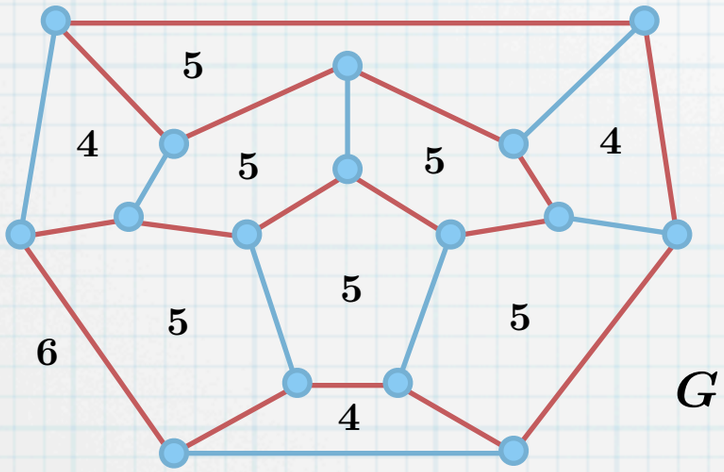


$$\alpha_4 = 1,\ \beta_4 = 2$$
$$\alpha_5 = 4,\ \beta_5 = 2$$
$$\alpha_6 = 0,\ \beta_6 = 1$$

* **Definition 20**:
  Sei G ein ebener Hamiltonscher Graph ohne Schlingen, und sei C ein Hamiltonscher Zyklus in G. Eine **Sehne** bezüglich C ist eine Kante in G, die dem Zyklus C nicht angehört.
  Eine **innere Sehne** ist eine Sehne, die im Inneren von C liegt.

Nicht-Hamiltonsche ebene Graphen

* **Definition 19**:
  Sei G ein ebener Graph, in dem es einen Hamiltonschen Zyklus C gibt. Der **Grad einer Fläche** ist die Anzahl der Knoten (oder Kanten), die diese Fläche begrenzen.
  Mit α_k bezeichnen wir die Anzahl Flächen vom Grad k, die innerhalb von C liegen.
  Mit β_k bezeichnen wir die Anzahl Flächen vom Grad k, die außerhalb von C liegen.
* Beispiel:



$$\alpha_4 = 1,\ \beta_4 = 2$$
$$\alpha_5 = 4,\ \beta_5 = 2$$
$$\alpha_6 = 0,\ \beta_6 = 1$$

* **Definition 20**:
  Sei G ein ebener Hamiltonscher Graph ohne Schlingen, und sei C ein Hamiltonscher Zyklus in G. Eine **Sehne** bezüglich C ist eine Kante in G, die dem Zyklus C nicht angehört.
  Eine **innere Sehne** ist eine Sehne, die im Inneren von C liegt.
  Eine **äußere Sehne** ist eine Sehne, die im Äußeren von C liegt.

Der Satz von Grinberg

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.
  Weil G eben ist, schneiden sich die Sehnen nicht.
  Jede innere Sehne gehört zu zwei inneren Flächen von C.
  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.
  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.
  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.
  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.
  Folglich ist $d = 2s + n$.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.
  Weil G eben ist, schneiden sich die Sehnen nicht.
  Jede innere Sehne gehört zu zwei inneren Flächen von C.
  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.
  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.
  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.
  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.
  Folglich ist $d = 2s + n$.
  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

  Zusammen genommen liefert dieses $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

  Zusammen genommen liefert dieses $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beispiel: Für den Graphen auf Seite 26 mit $\alpha_4 = 1, \beta_4 = 2, \alpha_5 = 4, \beta_5 = 2, \alpha_6 = 0, \beta_6 = 1$ gilt

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

  Zusammen genommen liefert dieses $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beispiel: Für den Graphen auf Seite 26 mit $\alpha_4 = 1, \beta_4 = 2, \alpha_5 = 4, \beta_5 = 2, \alpha_6 = 0, \beta_6 = 1$ gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k)$

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

  Zusammen genommen liefert dieses $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beispiel: Für den Graphen auf Seite 26 mit $\alpha_4 = 1, \beta_4 = 2, \alpha_5 = 4, \beta_5 = 2, \alpha_6 = 0, \beta_6 = 1$ gilt
  $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = (4-2)(\alpha_4 - \beta_4) + (5-2)(\alpha_5 - \beta_5) + (6-2)(\alpha_6 - \beta_6)$

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

  Zusammen genommen liefert dieses $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beispiel: Für den Graphen auf Seite 26 mit $\alpha_4 = 1, \beta_4 = 2, \alpha_5 = 4, \beta_5 = 2, \alpha_6 = 0, \beta_6 = 1$ gilt

$$\begin{aligned}\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) &= (4-2)(\alpha_4 - \beta_4) + (5-2)(\alpha_5 - \beta_5) + (6-2)(\alpha_6 - \beta_6) \\ &= 2(1-2) + 3(4-2) + 4(0-1)\end{aligned}$$

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

  Zusammen genommen liefert dieses $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beispiel: Für den Graphen auf Seite 26 mit $\alpha_4 = 1, \beta_4 = 2, \alpha_5 = 4, \beta_5 = 2, \alpha_6 = 0, \beta_6 = 1$ gilt

$$\begin{aligned}\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) &= (4-2)(\alpha_4 - \beta_4) + (5-2)(\alpha_5 - \beta_5) + (6-2)(\alpha_6 - \beta_6) \\ &= 2(1-2) + 3(4-2) + 4(0-1) = -2 + 6 - 4\end{aligned}$$

Der Satz von Grinberg

* **Satz 21** (Grinberg, 1968):
  Sei $G = (V, E)$ ein ebener Graph ohne Schlingen und $n := |V|$. Wenn G einen Hamiltonschen Zyklus C enthält, dann gilt $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beweis:

  Angenommen, es gibt s innere Sehnen bzgl. Zyklus C.

  Weil G eben ist, schneiden sich die Sehnen nicht.

  Jede innere Sehne gehört zu zwei inneren Flächen von C.

  Alle inneren Sehnen zusammen unterteilen das Innere von C in $s+1$ Flächen.

  Also gilt $\sum_{k=2}^{n} \alpha_k = s + 1$.

  Sei $d := \sum_{k=2}^{n} k \cdot \alpha_k$ die gewichtete Summe der Grade der inneren Flächen von C.

  In d wurde jede innere Sehne doppelt gezählt, und jede Kante von C einmal.

  Folglich ist $d = 2s + n$.

  Daraus wiederum folgt $\sum_{k=2}^{n} k \cdot \alpha_k = d = 2\left(\sum_{k=2}^{n} \alpha_k - 1\right) + n$.

  Somit ist $\sum_{k=2}^{n} (k-2) \cdot \alpha_k = n - 2$.

  Analog dazu zeigt man, dass für die äußere Sehnen gilt: $\sum_{k=2}^{n} (k-2) \cdot \beta_k = n - 2$.

  Zusammen genommen liefert dieses $\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) = 0$.
* Beispiel: Für den Graphen auf Seite 26 mit $\alpha_4 = 1, \beta_4 = 2, \alpha_5 = 4, \beta_5 = 2, \alpha_6 = 0, \beta_6 = 1$ gilt

$$\begin{aligned}\sum_{k=2}^{n} (k-2) \cdot (\alpha_k - \beta_k) &= (4-2)(\alpha_4 - \beta_4) + (5-2)(\alpha_5 - \beta_5) + (6-2)(\alpha_6 - \beta_6) \\ &= 2(1-2) + 3(4-2) + 4(0-1) = -2 + 6 - 4 = 0.\end{aligned}$$

Der Grinbergsche Graph

Der Grinbergsche Graph

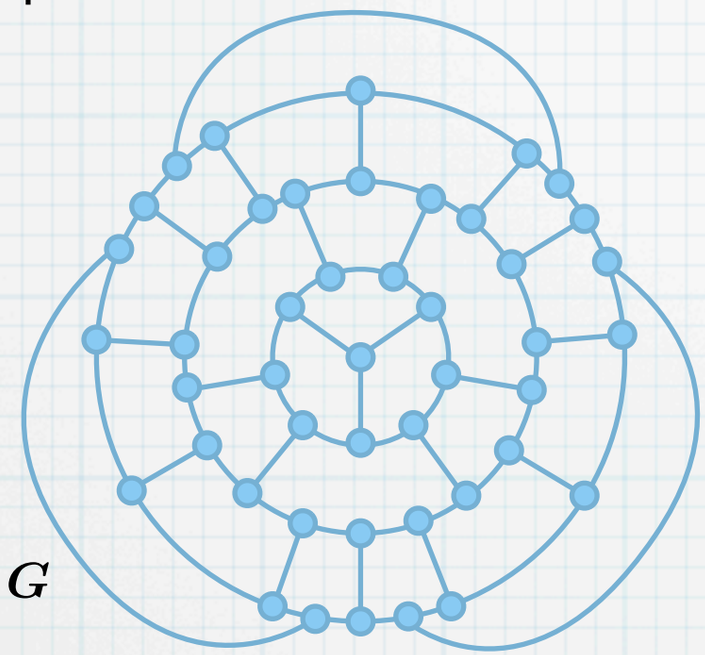
* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:

Der Grinbergsche Graph

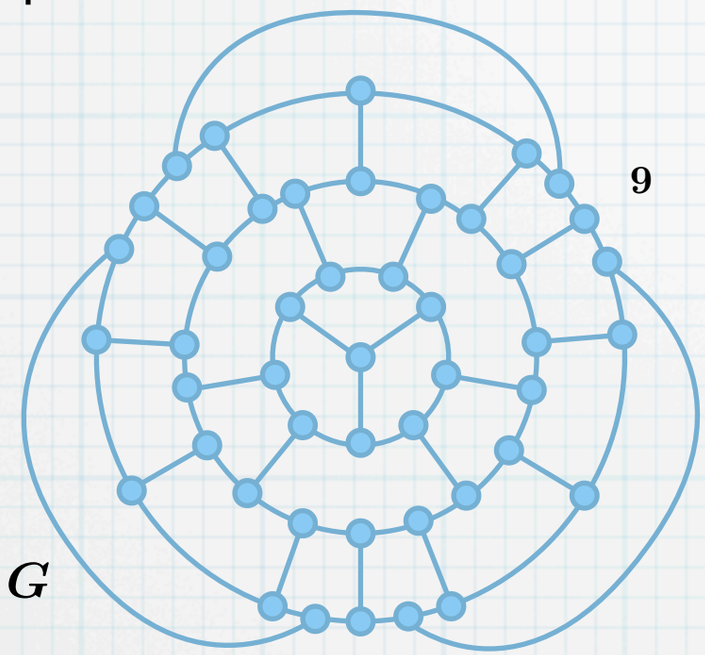
* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:



G

Der Grinbergsche Graph

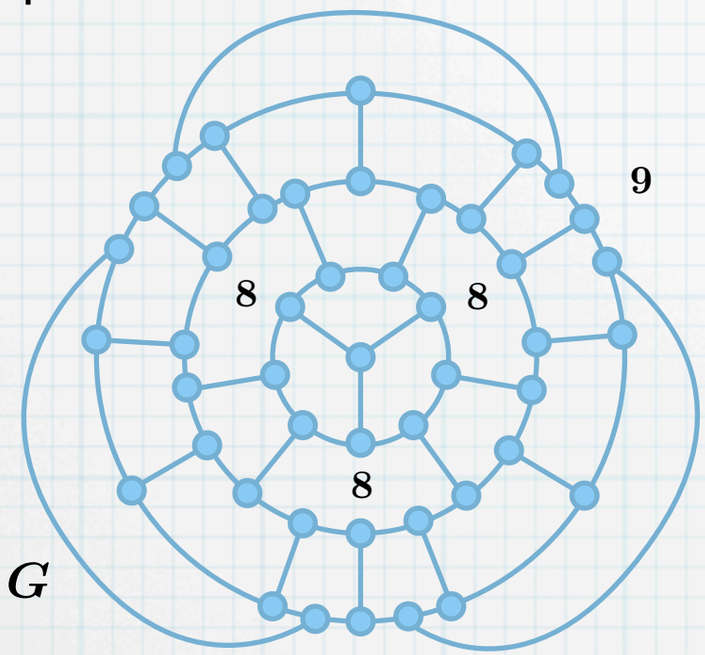
* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:



G

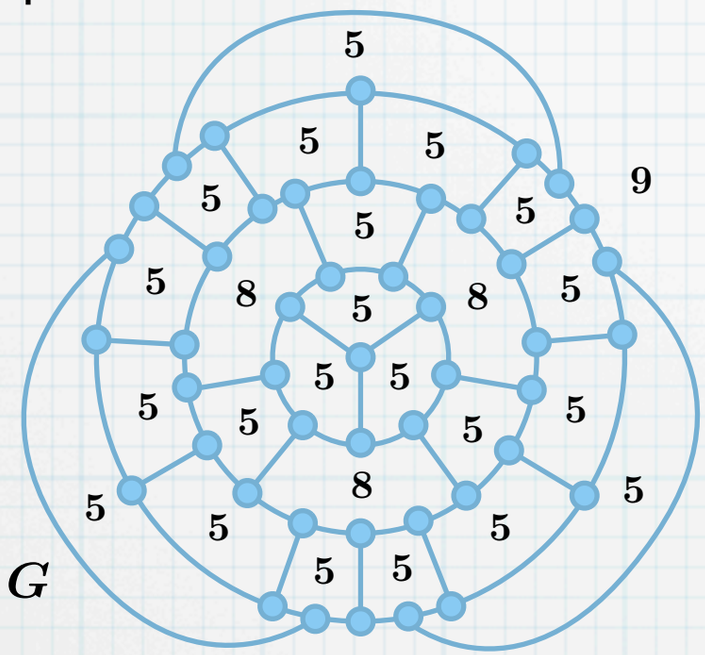
Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:



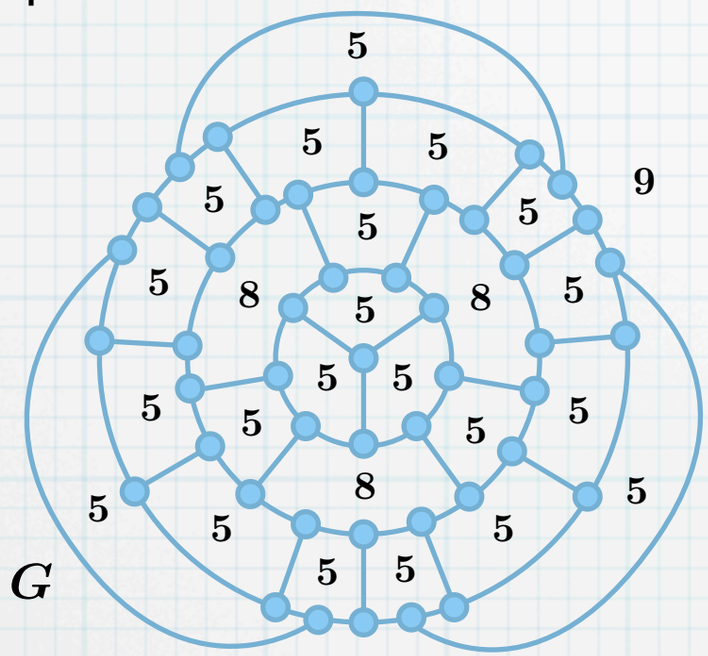
Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:



Der Grinbergsche Graph

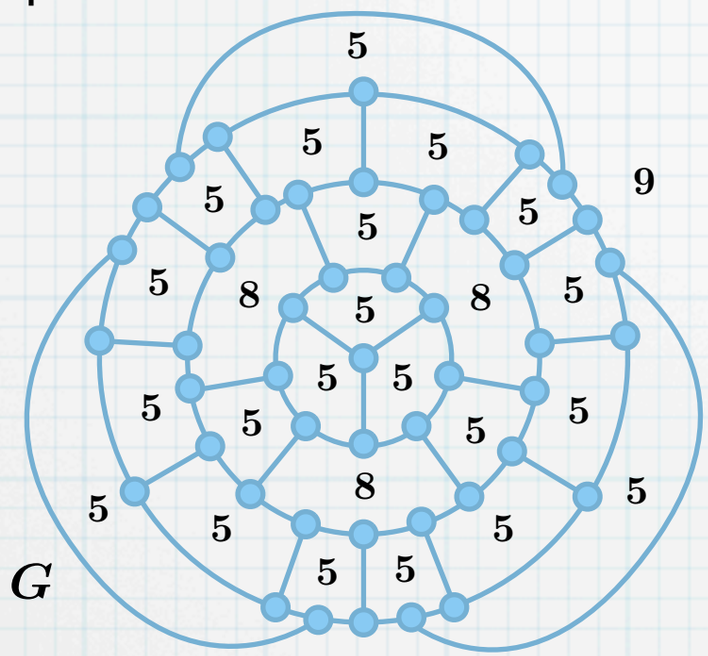
* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:



* **Satz 22**:
Der Grinbergsche Graph ist nicht-Hamiltonsch.

Der Grinbergsche Graph

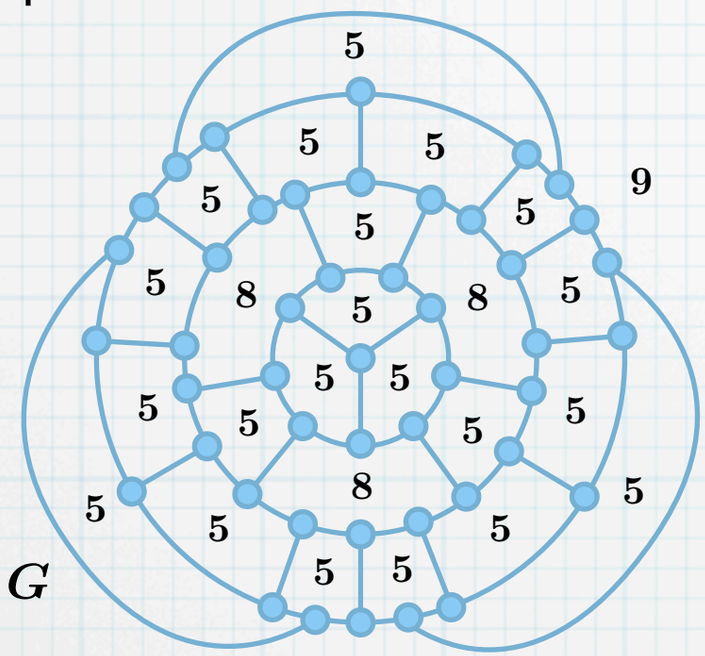
* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:



* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:

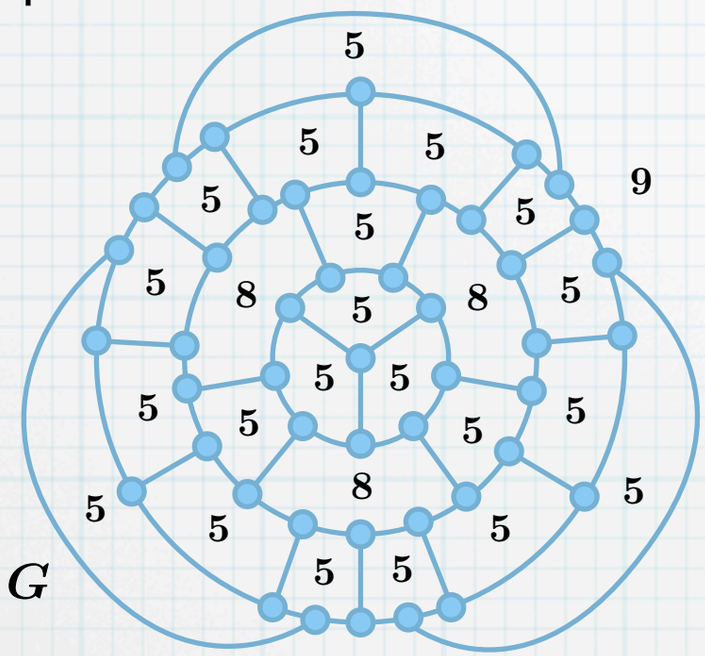


* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

  Angenommen, der Graph hat einen Hamiltonschen Zyklus C.

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:

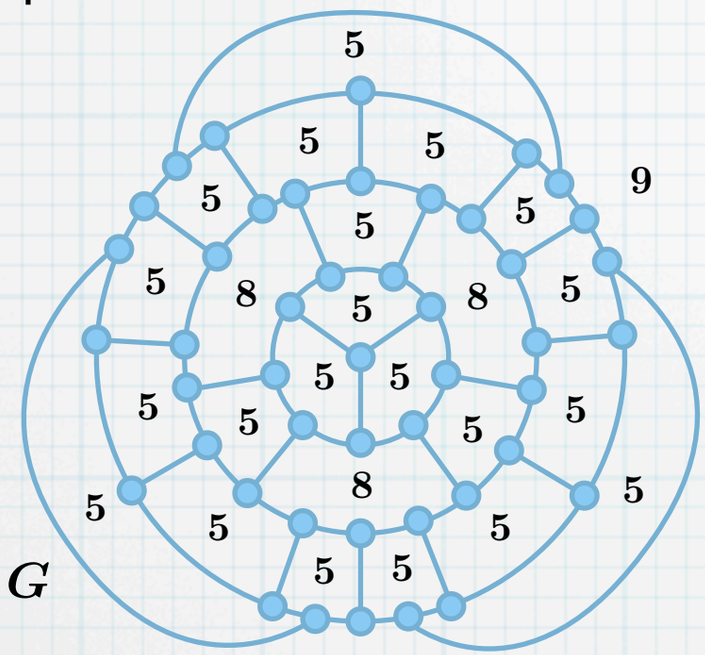


* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

  Angenommen, der Graph hat einen Hamiltonschen Zyklus C.
  Nach dem Satz von Grinberg ist dann $\sum_{k=2}^{n}(k-2)\cdot(\alpha_k-\beta_k)=0$.

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:

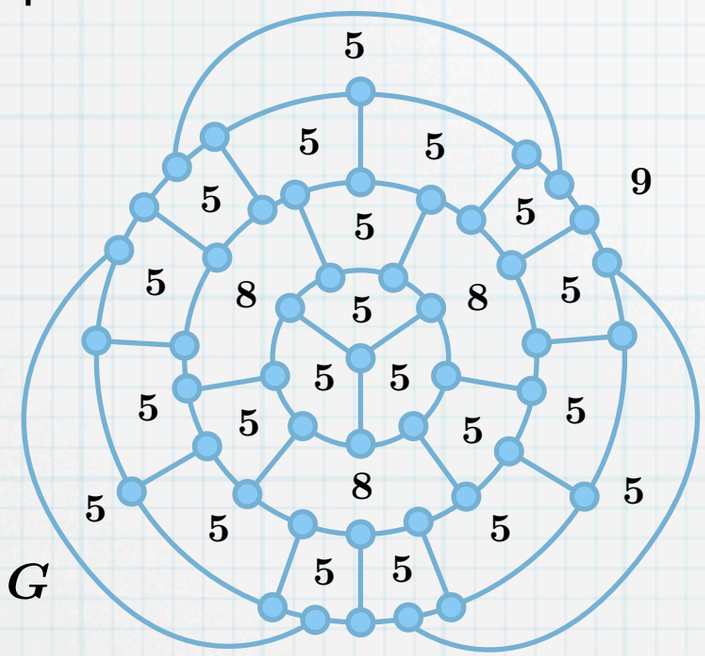


* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

  Angenommen, der Graph hat einen Hamiltonschen Zyklus C.
  Nach dem Satz von Grinberg ist dann $\sum_{k=2}^{n}(k-2)\cdot(\alpha_k-\beta_k)=0$.
  Also hier: $3(\alpha_5-\beta_5)+6(\alpha_8-\beta_8)+7(\alpha_9-\beta_9)=0$,

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:

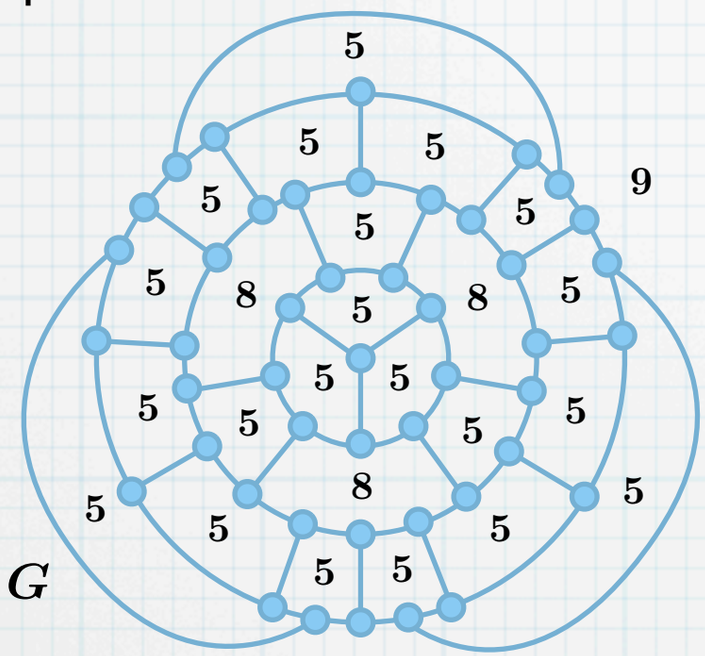


* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

  Angenommen, der Graph hat einen Hamiltonschen Zyklus C.
  Nach dem Satz von Grinberg ist dann $\sum_{k=2}^{n}(k-2)\cdot(\alpha_k-\beta_k)=0$.
  Also hier: $3(\alpha_5-\beta_5)+6(\alpha_8-\beta_8)+7(\alpha_9-\beta_9)=0$,
  was äquivalent ist zu $3((\alpha_5-\beta_5)+2(\alpha_8-\beta_8))=7(\beta_9-\alpha_9)$.

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:

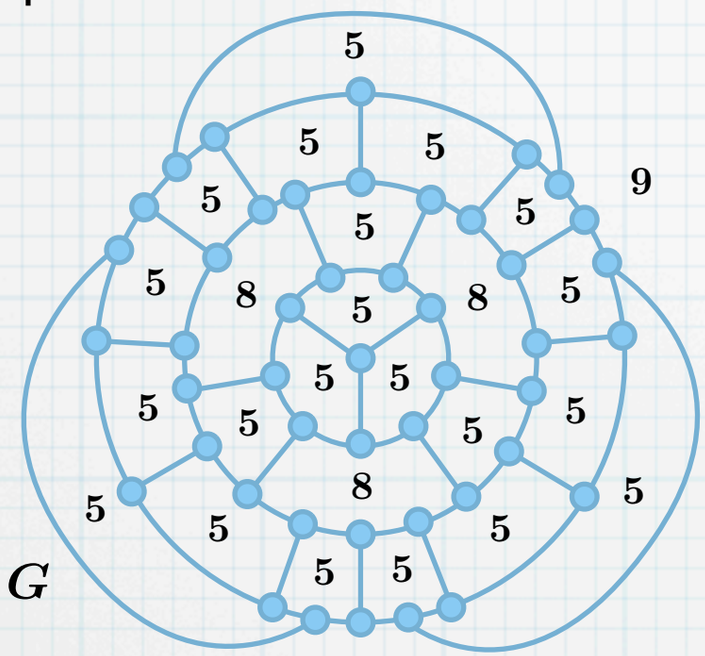


* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

  Angenommen, der Graph hat einen Hamiltonschen Zyklus C.
  Nach dem Satz von Grinberg ist dann $\sum_{k=2}^{n}(k-2)\cdot(\alpha_k-\beta_k)=0$.
  Also hier: $3(\alpha_5-\beta_5)+6(\alpha_8-\beta_8)+7(\alpha_9-\beta_9)=0$,
  was äquivalent ist zu $3((\alpha_5-\beta_5)+2(\alpha_8-\beta_8))=7(\beta_9-\alpha_9)$.
  Dann ist 3 ein Teiler von $7(\beta_9-\alpha_9)$.

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:

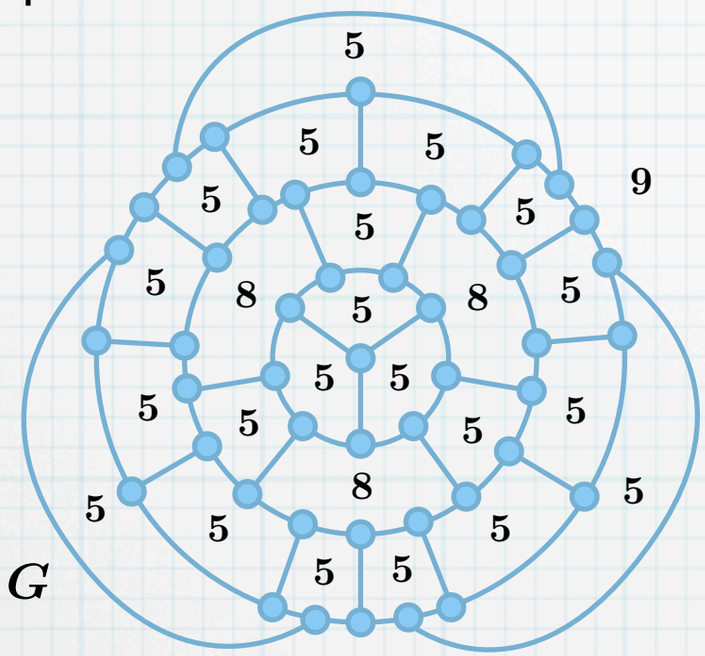


* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

  Angenommen, der Graph hat einen Hamiltonschen Zyklus C.
  Nach dem Satz von Grinberg ist dann $\sum_{k=2}^{n}(k-2)\cdot(\alpha_k-\beta_k)=0$.
  Also hier: $3(\alpha_5-\beta_5)+6(\alpha_8-\beta_8)+7(\alpha_9-\beta_9)=0$,
  was äquivalent ist zu $3((\alpha_5-\beta_5)+2(\alpha_8-\beta_8))=7(\beta_9-\alpha_9)$.
  Dann ist 3 ein Teiler von $7(\beta_9-\alpha_9)$.
  Die einzige Fläche vom Grad 9 ist die äußere Fläche: $\beta_9=1, \alpha_9=0$.

Der Grinbergsche Graph

* Der Satz von Grinberg wird häuptsächlich benutzt um zu zeigen, dass ein ebener Graph nicht-Hamiltonsch ist.
* Beispiel:



* **Satz 22**:
  Der Grinbergsche Graph ist nicht-Hamiltonsch.
* Beweis:

  Angenommen, der Graph hat einen Hamiltonschen Zyklus C.
  Nach dem Satz von Grinberg ist dann $\sum_{k=2}^{n}(k-2)\cdot(\alpha_k-\beta_k)=0$.
  Also hier: $3(\alpha_5-\beta_5)+6(\alpha_8-\beta_8)+7(\alpha_9-\beta_9)=0$,
  was äquivalent ist zu $3((\alpha_5-\beta_5)+2(\alpha_8-\beta_8))=7(\beta_9-\alpha_9)$.
  Dann ist 3 ein Teiler von $7(\beta_9-\alpha_9)$.
  Die einzige Fläche vom Grad 9 ist die äußere Fläche: $\beta_9=1, \alpha_9=0$.
  Aber 3 ist kein Teiler von $7(\beta_9-\alpha_9)=7$. Widerspruch.

Das Problem des Handlungsreisenden (TSP)

Das Problem des Handlungsreisenden (TSP)

Das Problem des Handlungsreisenden (TSP)

* Angenommen, das Gebiet eines Handlungsreisenden umfasst zahlreiche Städte, die durch ein Straßennetz miteinander verbunden sind.

Das Problem des Handlungsreisenden (TSP)

* Angenommen, das Gebiet eines Handlungsreisenden umfasst zahlreiche Städte, die durch ein Straßennetz miteinander verbunden sind.
* Seine Arbeit erfordert es, sämtliche Städte nacheinander zu besuchen. Am Ende der Tour möchte er wieder am Wohnort ankommen.

Das Problem des Handlungsreisenden (TSP)

* Angenommen, das Gebiet eines Handlungsreisenden umfasst zahlreiche Städte, die durch ein Straßennetz miteinander verbunden sind.
* Seine Arbeit erfordert es, sämtliche Städte nacheinander zu besuchen. Am Ende der Tour möchte er wieder am Wohnort ankommen.
* Ist es möglich, sämtliche Städte zu einer Rundtour zu verbinden, so dass jede Stadt genau einmal besucht wird (genauer gesagt: einmal betreten und einmal verlassen wird)?

Das Problem des Handlungsreisenden (TSP)

* Angenommen, das Gebiet eines Handlungsreisenden umfasst zahlreiche Städte, die durch ein Straßennetz miteinander verbunden sind.
* Seine Arbeit erfordert es, sämtliche Städte nacheinander zu besuchen. Am Ende der Tour möchte er wieder am Wohnort ankommen.
* Ist es möglich, sämtliche Städte zu einer Rundtour zu verbinden, so dass jede Stadt genau einmal besucht wird (genauer gesagt: einmal betreten und einmal verlassen wird)?
* Falls dieses möglich ist, wie kann eine Tour mit minimaler Gesamtlänge gefunden werden?

Das Problem des Handlungsreisenden (TSP)

* Angenommen, das Gebiet eines Handlungsreisenden umfasst zahlreiche Städte, die durch ein Straßennetz miteinander verbunden sind.
* Seine Arbeit erfordert es, sämtliche Städte nacheinander zu besuchen. Am Ende der Tour möchte er wieder am Wohnort ankommen.
* Ist es möglich, sämtliche Städte zu einer Rundtour zu verbinden, so dass jede Stadt genau einmal besucht wird (genauer gesagt: einmal betreten und einmal verlassen wird)?
* Falls dieses möglich ist, wie kann eine Tour mit minimaler Gesamtlänge gefunden werden?
* **Definition 23**:
  Das Problem, auf dem vollständigen bewerteten Graphen (K_n, c) mit $c_{i,j} \geq 0$ einen Hamiltonschen Kreis C zu finden, so dass $c(C)$ minimal ist, wird als **Handlungsreisendenproblem** (**traveling salesman problem**, **TSP**) bezeichnet.

Das Problem des Handlungsreisenden (TSP)

* Angenommen, das Gebiet eines Handlungsreisenden umfasst zahlreiche Städte, die durch ein Straßennetz miteinander verbunden sind.
* Seine Arbeit erfordert es, sämtliche Städte nacheinander zu besuchen. Am Ende der Tour möchte er wieder am Wohnort ankommen.
* Ist es möglich, sämtliche Städte zu einer Rundtour zu verbinden, so dass jede Stadt genau einmal besucht wird (genauer gesagt: einmal betreten und einmal verlassen wird)?
* Falls dieses möglich ist, wie kann eine Tour mit minimaler Gesamtlänge gefunden werden?
* **Definition 23**:
  Das Problem, auf dem vollständigen bewerteten Graphen (K_n, c) mit $c_{i,j} \geq 0$ einen Hamiltonschen Kreis C zu finden, so dass $c(C)$ minimal ist, wird als **Handlungsreisendenproblem** (**traveling salesman problem**, **TSP**) bezeichnet.

  Erfüllt c für alle Knotentripel i, j, k die Dreiecksungleichung $c_{i,j} \leq c_{i,k} + c_{k,j}$, dann spricht man vom **metrischen Handlungsreisendenproblem** (**mTSP**).

Das Problem des Handlungsreisenden (TSP)

* Angenommen, das Gebiet eines Handlungsreisenden umfasst zahlreiche Städte, die durch ein Straßennetz miteinander verbunden sind.
* Seine Arbeit erfordert es, sämtliche Städte nacheinander zu besuchen. Am Ende der Tour möchte er wieder am Wohnort ankommen.
* Ist es möglich, sämtliche Städte zu einer Rundtour zu verbinden, so dass jede Stadt genau einmal besucht wird (genauer gesagt: einmal betreten und einmal verlassen wird)?
* Falls dieses möglich ist, wie kann eine Tour mit minimaler Gesamtlänge gefunden werden?
* **Definition 23**:
  Das Problem, auf dem vollständigen bewerteten Graphen (K_n, c) mit $c_{i,j} \geq 0$ einen Hamiltonschen Kreis C zu finden, so dass $c(C)$ minimal ist, wird als **Handlungsreisendenproblem** (**traveling salesman problem**, **TSP**) bezeichnet.

  Erfüllt c für alle Knotentripel i, j, k die Dreiecksungleichung $c_{i,j} \leq c_{i,k} + c_{k,j}$, dann spricht man vom **metrischen Handlungsreisendenproblem** (**mTSP**).

  Das **asymmetrische Handlungsreisendenproblem** (**aTSP**) ist auf dem vollständigen gerichteten Graphen $\overleftrightarrow{K}_n$ definiert, wobei insbesondere $c_{i,j} \neq c_{j,i}$ zugelassen ist.

Ein Problem mit Geschichte

Ein Problem mit Geschichte

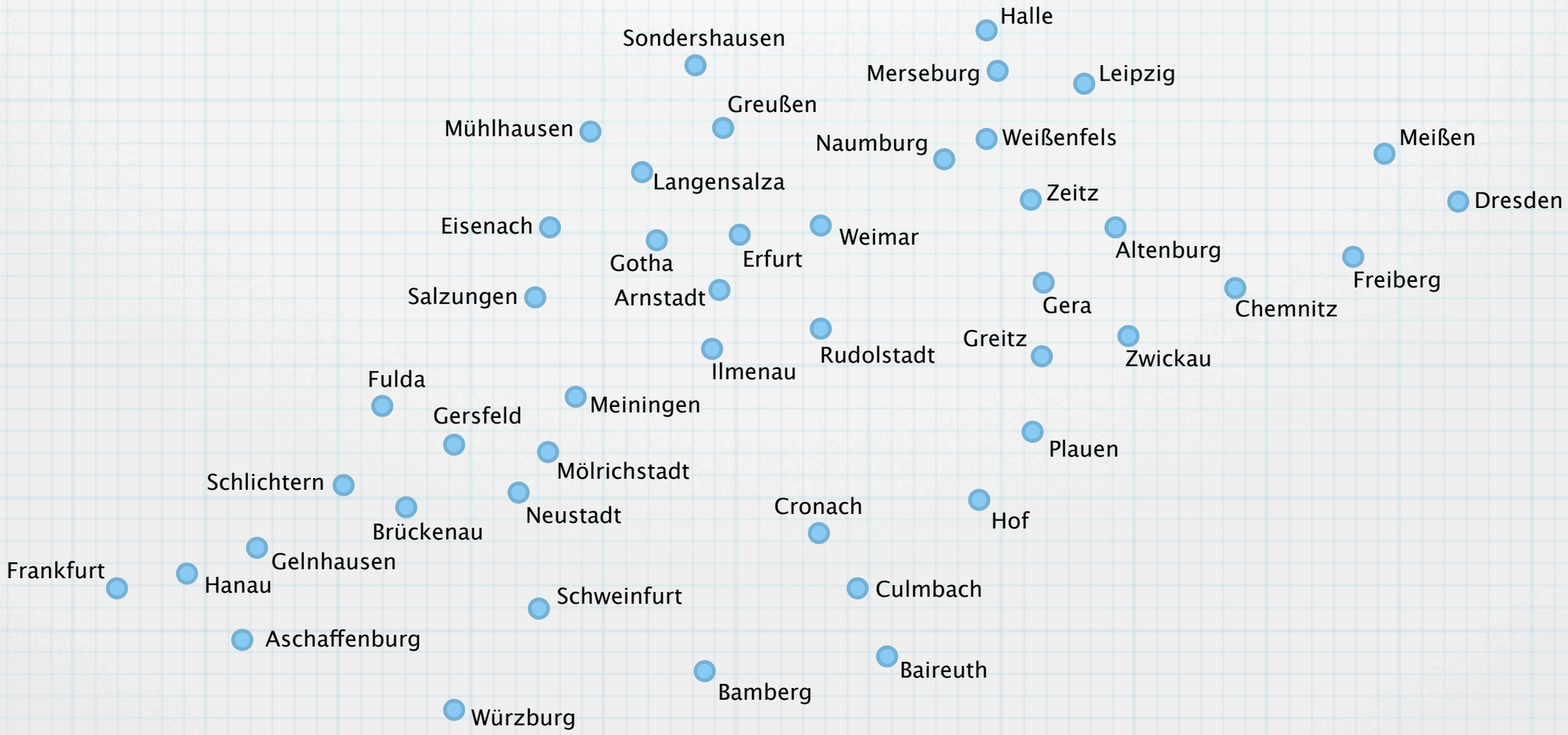
* Älteste Referenz: Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur (1832)

Ein Problem mit Geschichte

* Älteste Referenz: Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur (1832)
* „Die Geschäfte führen die Handlungsreisenden bald hier, bald dort hin, und es lassen sich nicht füglich Reisetouren angeben, die für alle vorkommende Fälle passend sind; aber es kann durch eine zweckmäßige Wahl und Eintheilung der Tour, manchmal so viel Zeit gewonnen werden, daß wir es nicht glauben umgehen zu dürfen, auch hierüber einige Vorschriften zu geben. Ein Jeder möge so viel davon benutzen, als er es seinem Zwecke für dienlich hält; so viel glauben wir aber davon versichern zu dürfen, daß es nicht wohl thunlich sein wird, die Touren durch Deutschland in Absicht der Entfernungen und, worauf der Reisende hauptsächlich zu sehen hat, des Hin- und Herreisens, mit mehr Oekonomie einzurichten. Die Hauptsache besteht immer darin: so viele Orte wie möglich mitzunehmen, ohne den nämlichen Ort zweimal berühren zu müssen.“

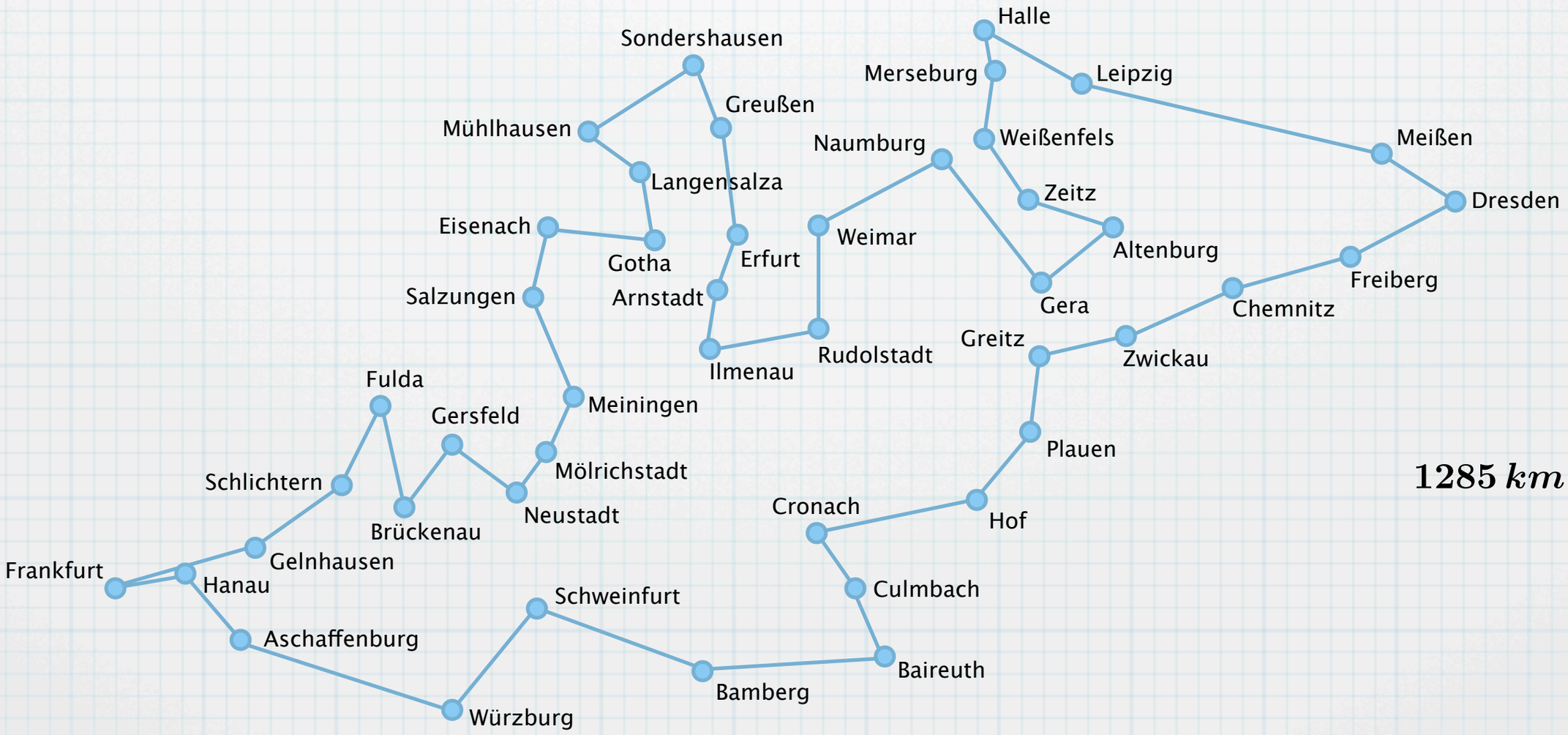
Ein Problem mit Geschichte

* Älteste Referenz: Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur (1832)
* „Die Geschäfte führen die Handlungsreisenden bald hier, bald dort hin, und es lassen sich nicht füglich Reisetouren angeben, die für alle vorkommende Fälle passend sind; aber es kann durch eine zweckmäßige Wahl und Eintheilung der Tour, manchmal so viel Zeit gewonnen werden, daß wir es nicht glauben umgehen zu dürfen, auch hierüber einige Vorschriften zu geben. Ein Jeder möge so viel davon benutzen, als er es seinem Zwecke für dienlich hält; so viel glauben wir aber davon versichern zu dürfen, daß es nicht wohl thunlich sein wird, die Touren durch Deutschland in Absicht der Entfernungen und, worauf der Reisende hauptsächlich zu sehen hat, des Hin- und Herreisens, mit mehr Oekonomie einzurichten. Die Hauptsache besteht immer darin: so viele Orte wie möglich mitzunehmen, ohne den nämlichen Ort zweimal berühren zu müssen.“



Ein Problem mit Geschichte

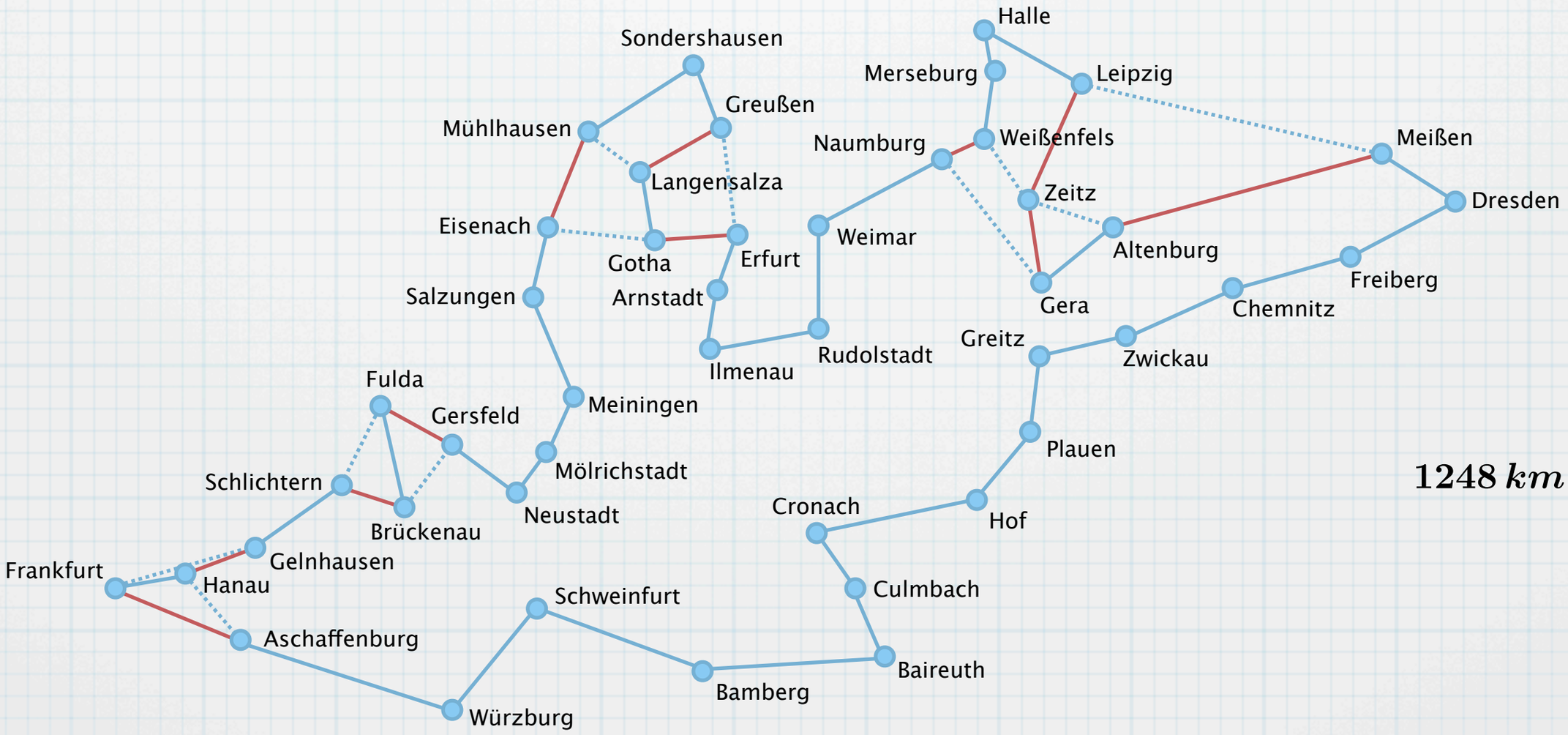
* Älteste Referenz: Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur (1832)
* „Die Geschäfte führen die Handlungsreisenden bald hier, bald dort hin, und es lassen sich nicht füglich Reisetouren angeben, die für alle vorkommende Fälle passend sind; aber es kann durch eine zweckmäßige Wahl und Eintheilung der Tour, manchmal so viel Zeit gewonnen werden, daß wir es nicht glauben umgehen zu dürfen, auch hierüber einige Vorschriften zu geben. Ein Jeder möge so viel davon benutzen, als er es seinem Zwecke für dienlich hält; so viel glauben wir aber davon versichern zu dürfen, daß es nicht wohl thunlich sein wird, die Touren durch Deutschland in Absicht der Entfernungen und, worauf der Reisende hauptsächlich zu sehen hat, des Hin- und Herreisens, mit mehr Oekonomie einzurichten. Die Hauptsache besteht immer darin: so viele Orte wie möglich mitzunehmen, ohne den nämlichen Ort zweimal berühren zu müssen.“



1285 *km*

Ein Problem mit Geschichte

* Älteste Referenz: Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur (1832)
* „Die Geschäfte führen die Handlungsreisenden bald hier, bald dort hin, und es lassen sich nicht füglich Reisetouren angeben, die für alle vorkommende Fälle passend sind; aber es kann durch eine zweckmäßige Wahl und Eintheilung der Tour, manchmal so viel Zeit gewonnen werden, daß wir es nicht glauben umgehen zu dürfen, auch hierüber einige Vorschriften zu geben. Ein Jeder möge so viel davon benutzen, als er es seinem Zwecke für dienlich hält; so viel glauben wir aber davon versichern zu dürfen, daß es nicht wohl thunlich sein wird, die Touren durch Deutschland in Absicht der Entfernungen und, worauf der Reisende hauptsächlich zu sehen hat, des Hin- und Herreisens, mit mehr Oekonomie einzurichten. Die Hauptsache besteht immer darin: so viele Orte wie möglich mitzunehmen, ohne den nämlichen Ort zweimal berühren zu müssen.“



1248 *km*

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

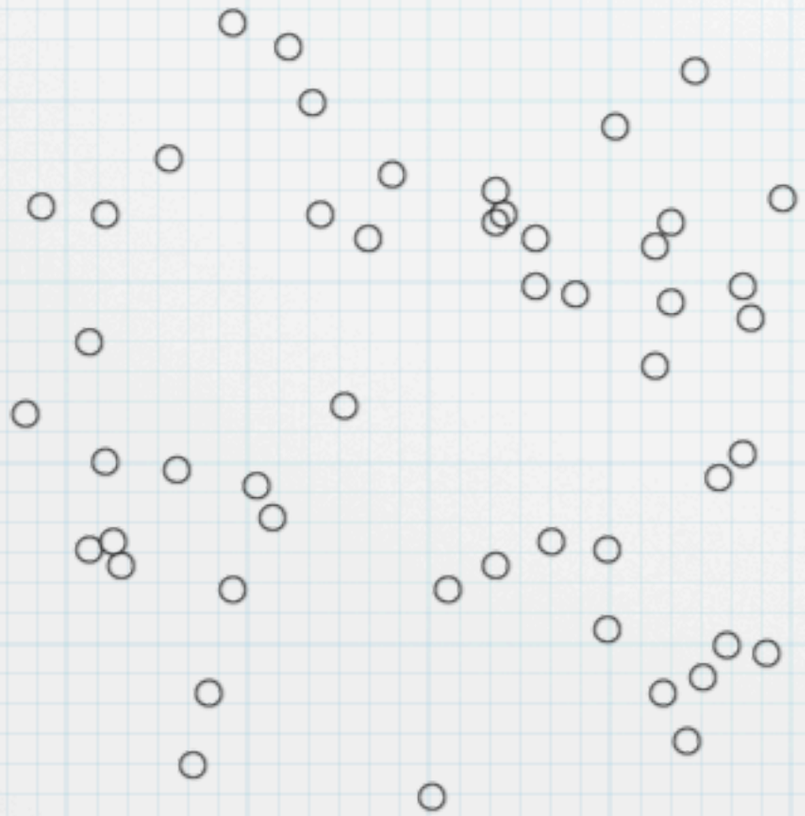
* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.

Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:

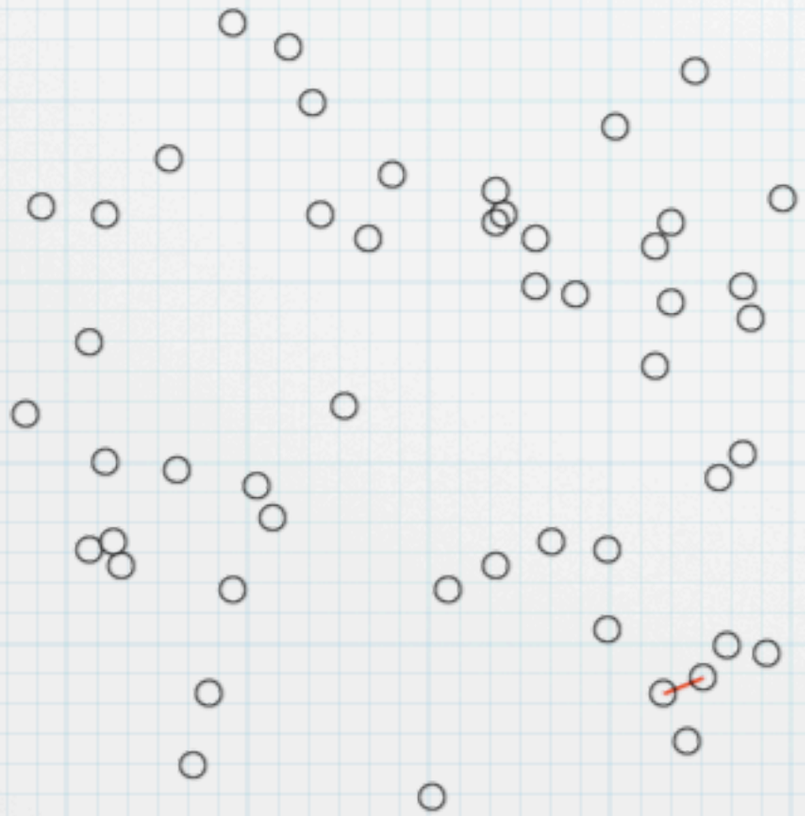
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



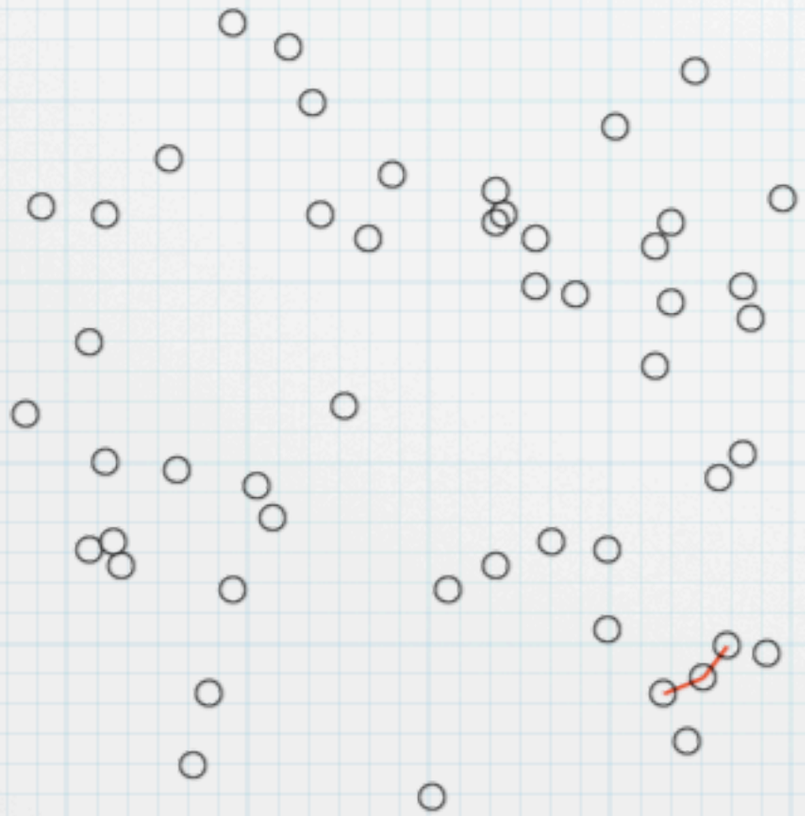
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



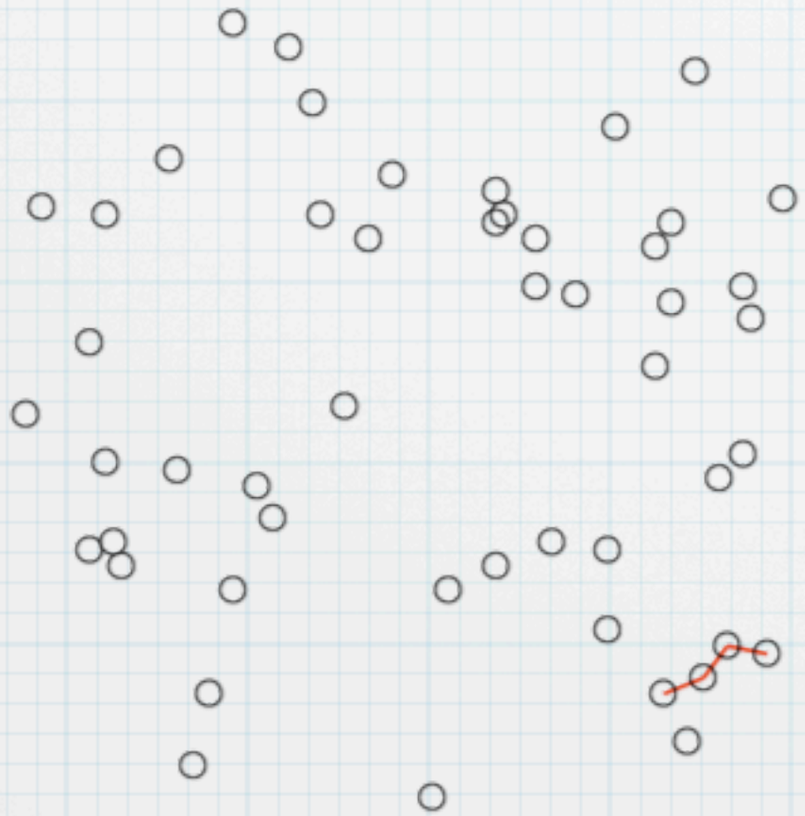
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



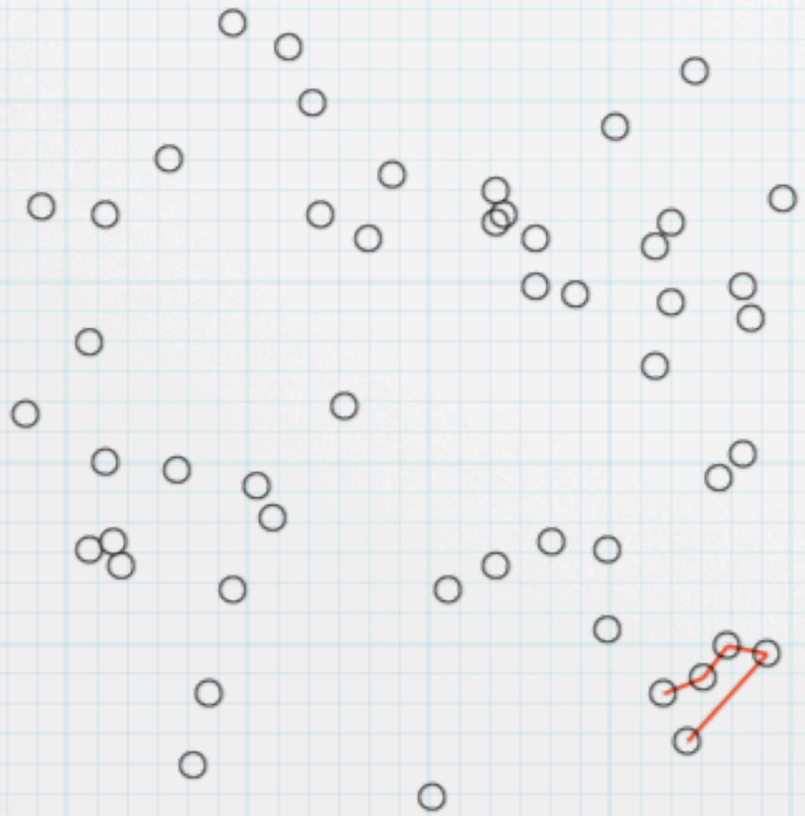
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



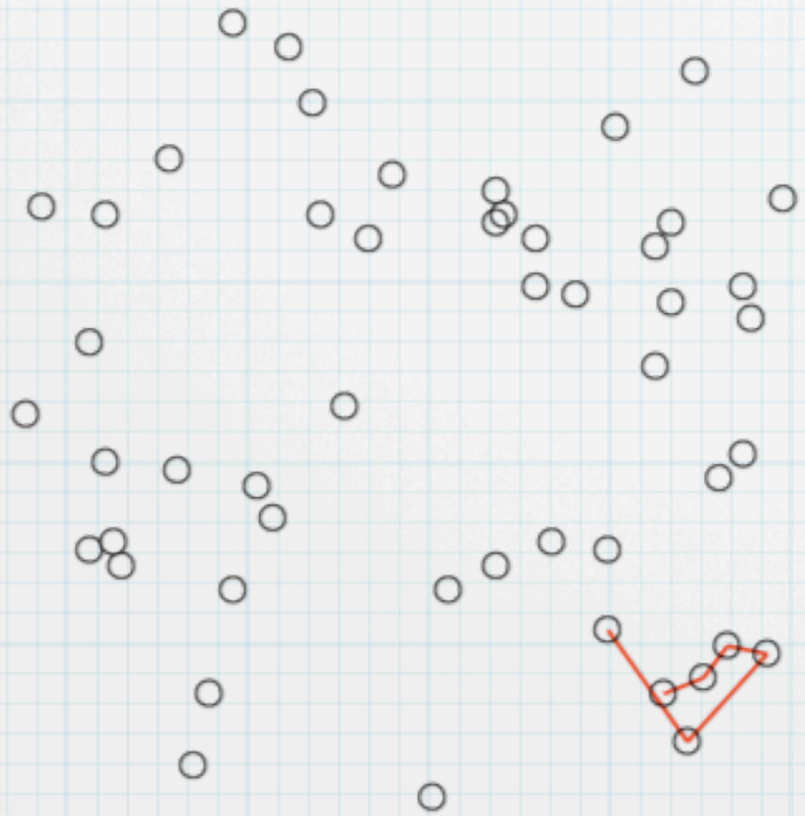
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



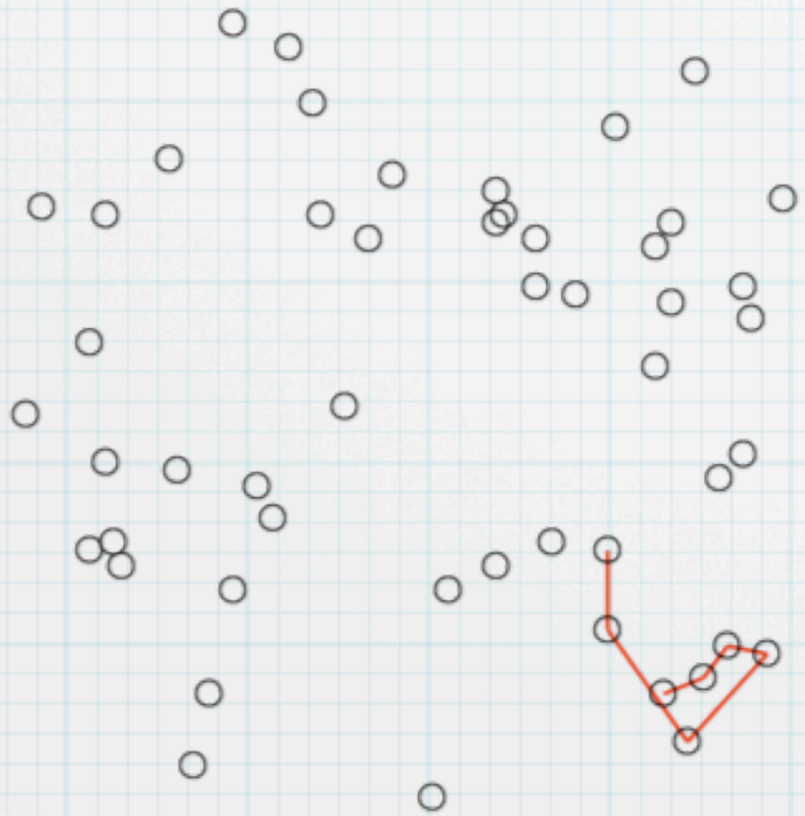
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



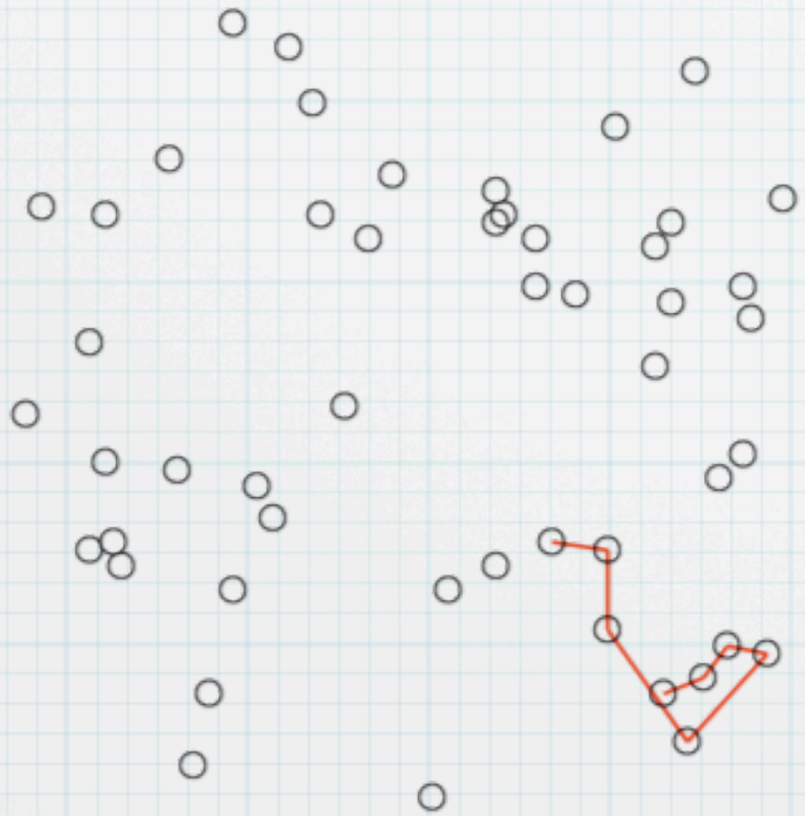
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



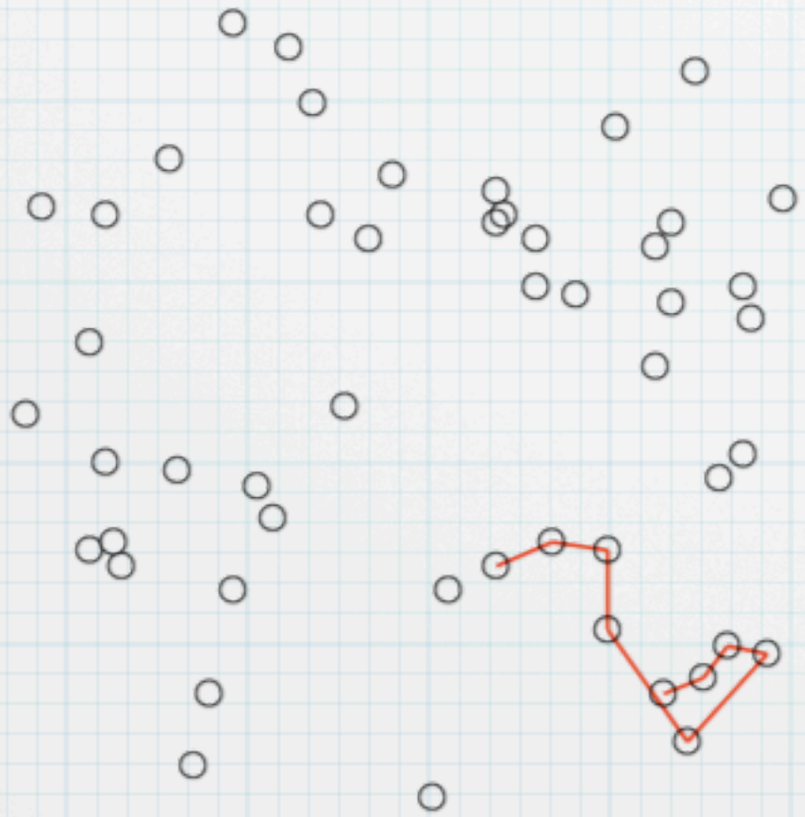
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



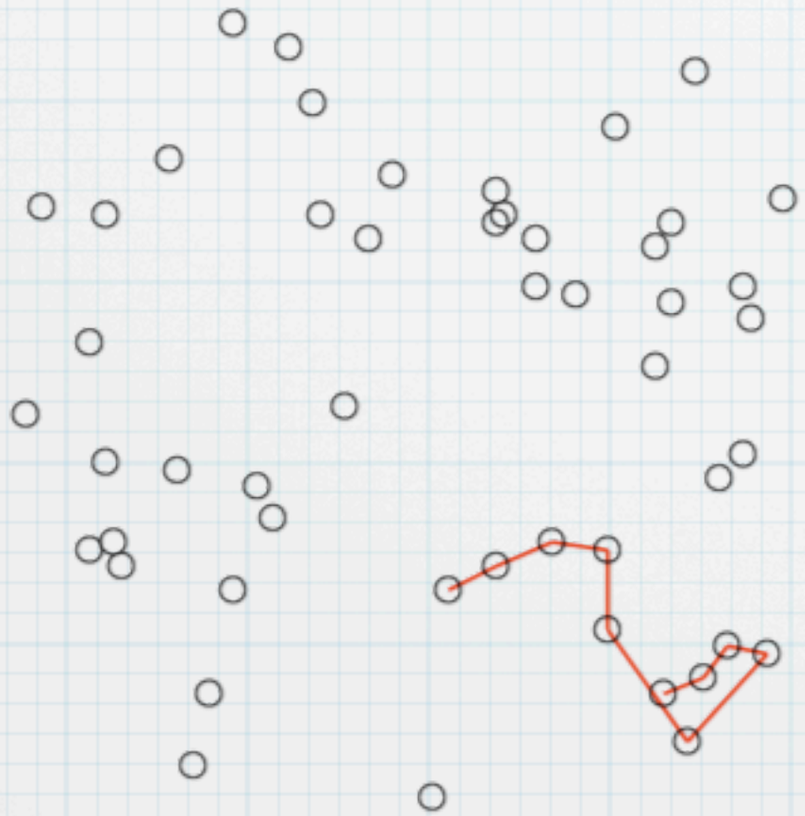
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



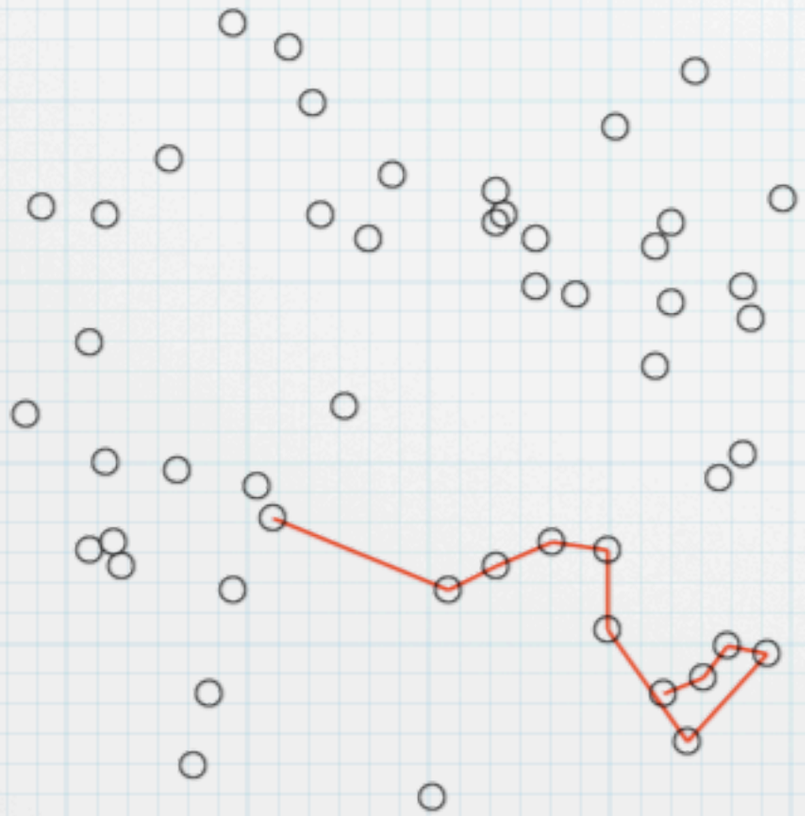
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



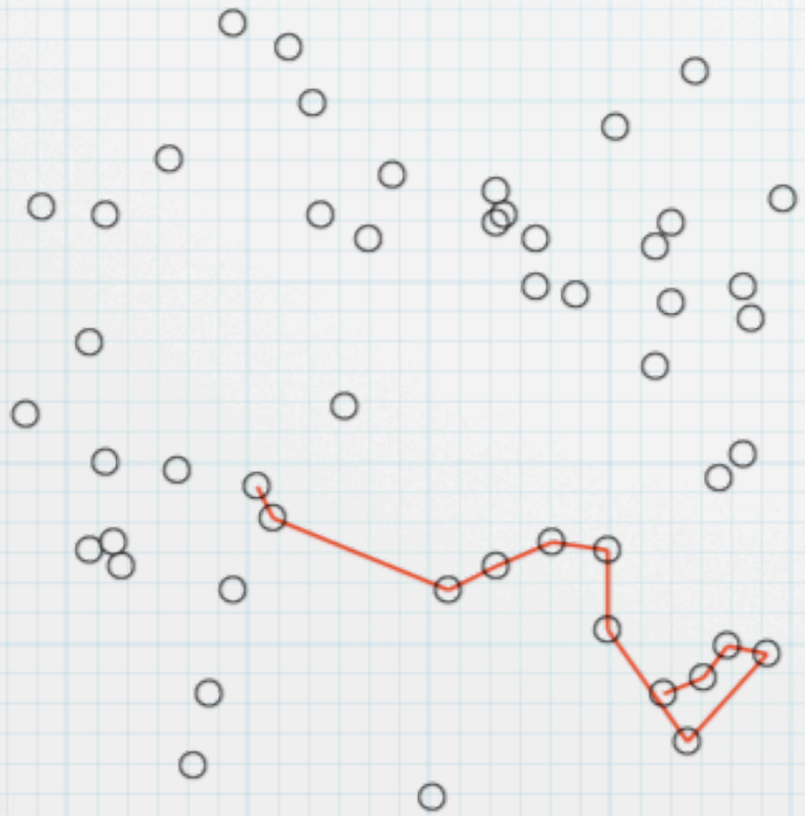
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



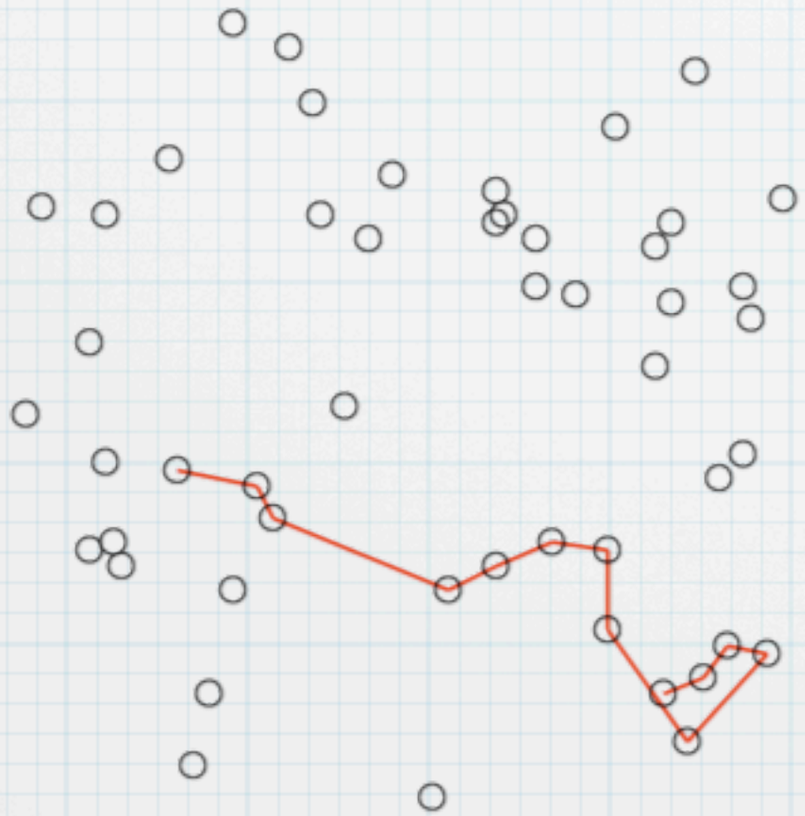
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



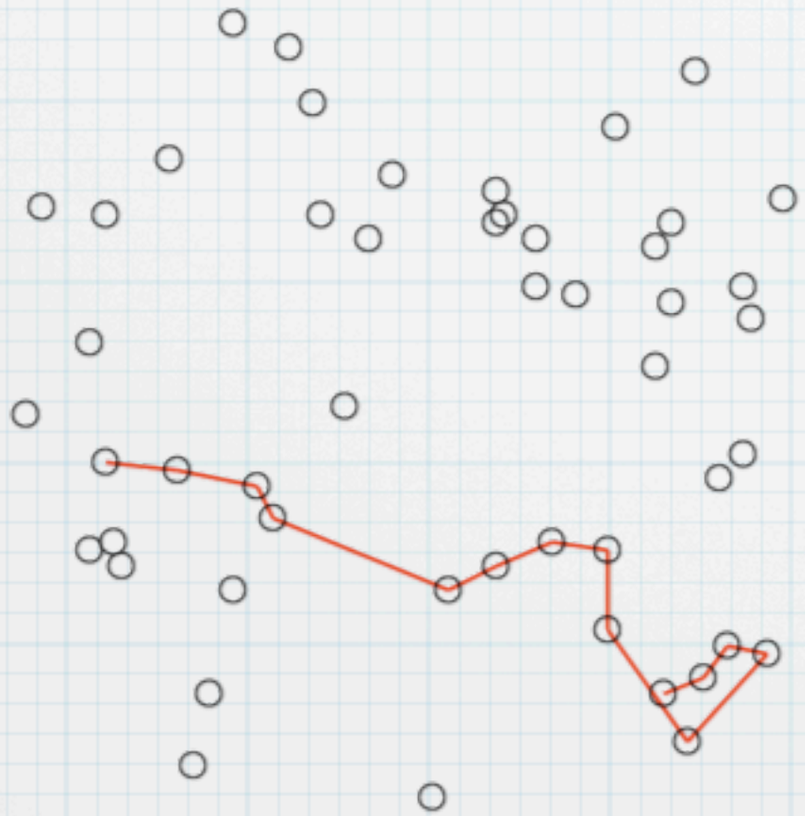
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



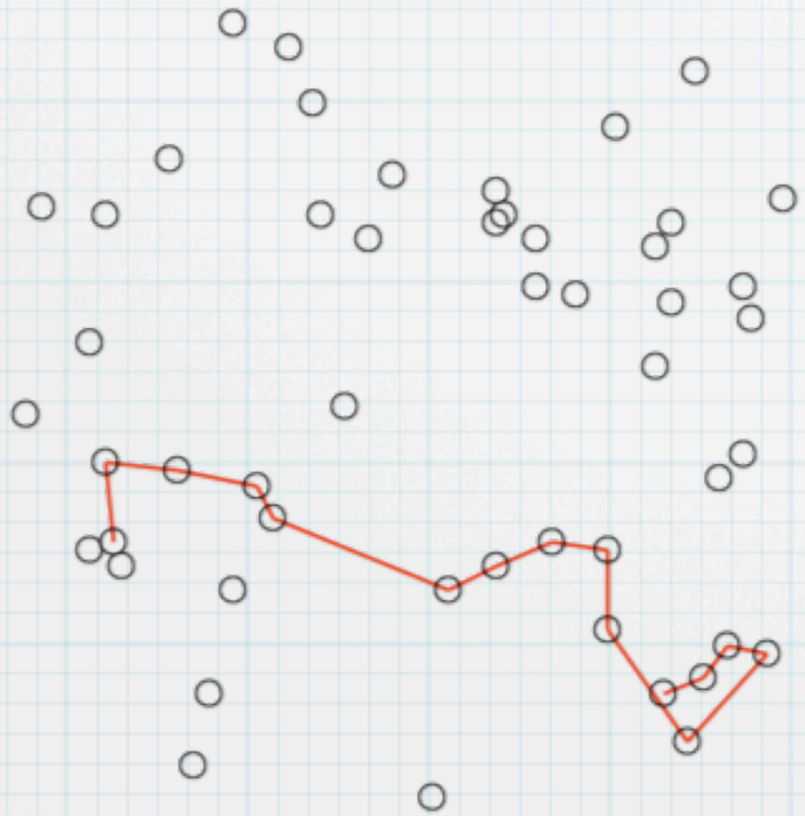
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



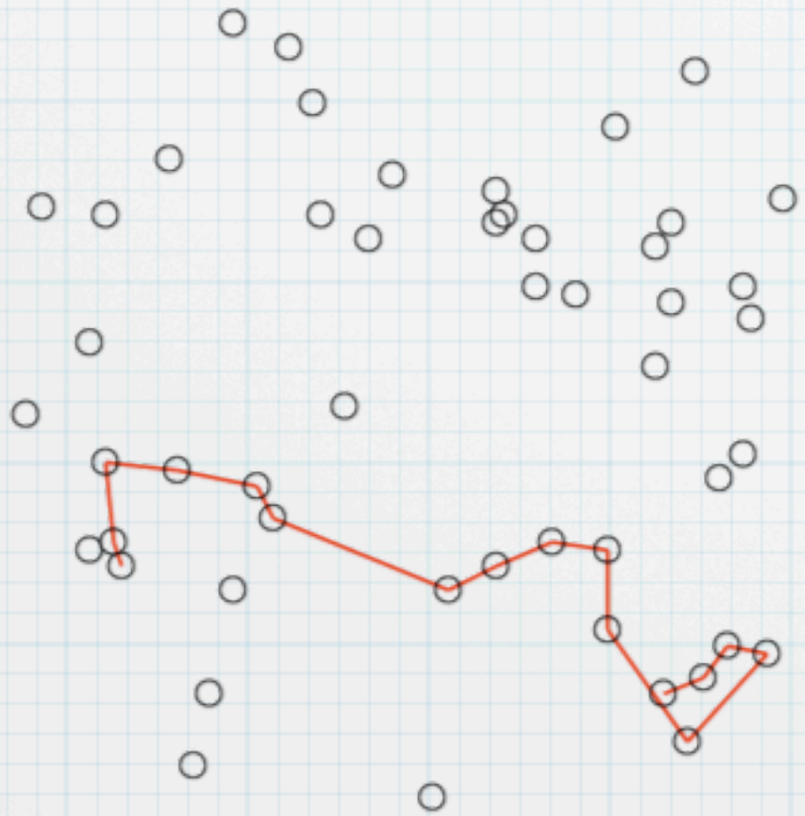
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



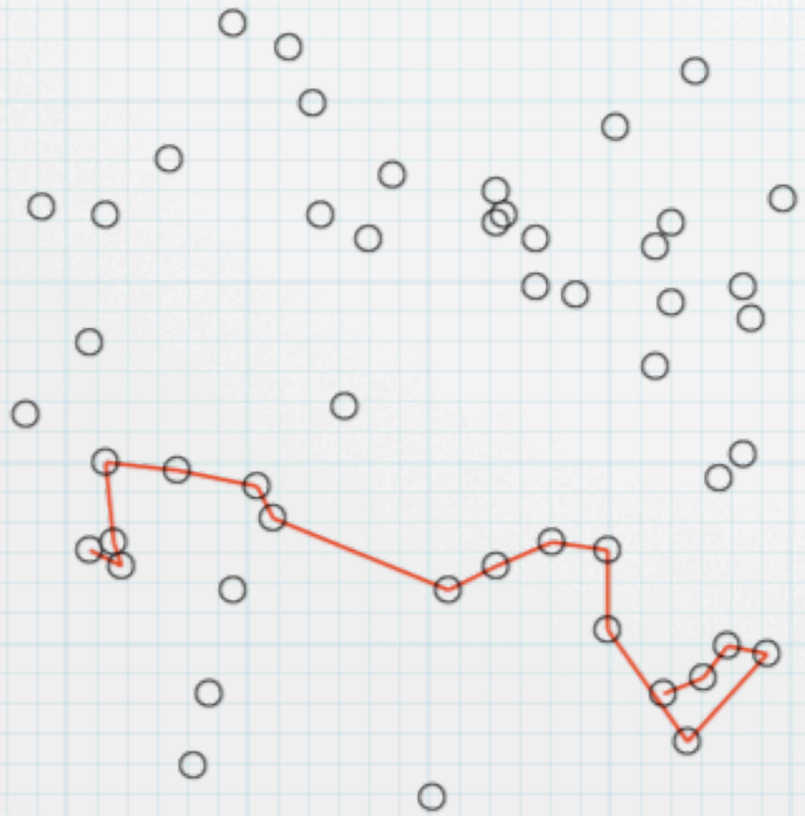
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



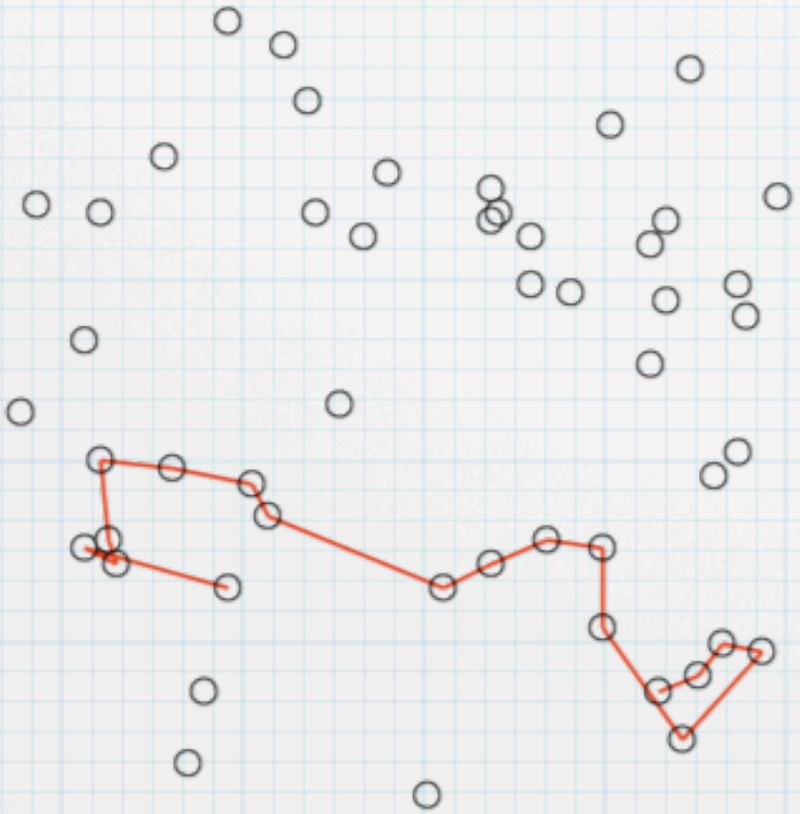
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



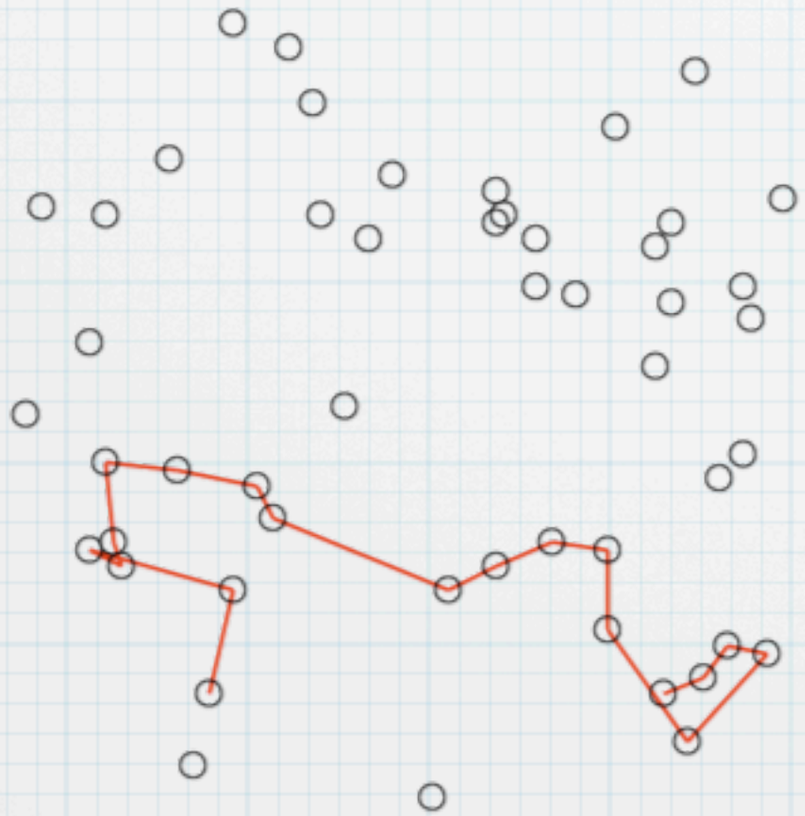
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



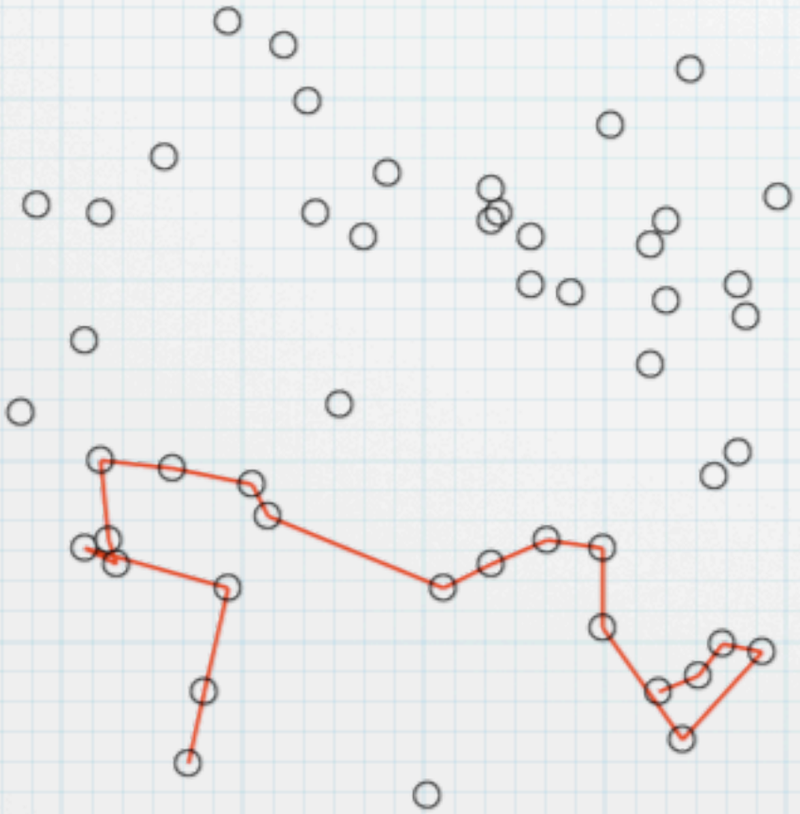
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



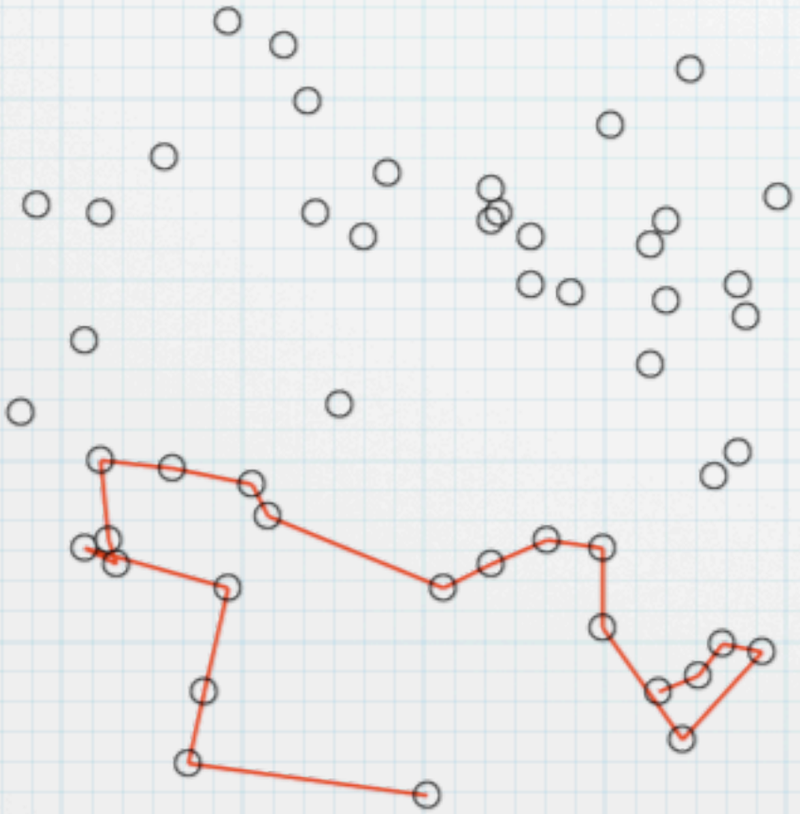
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



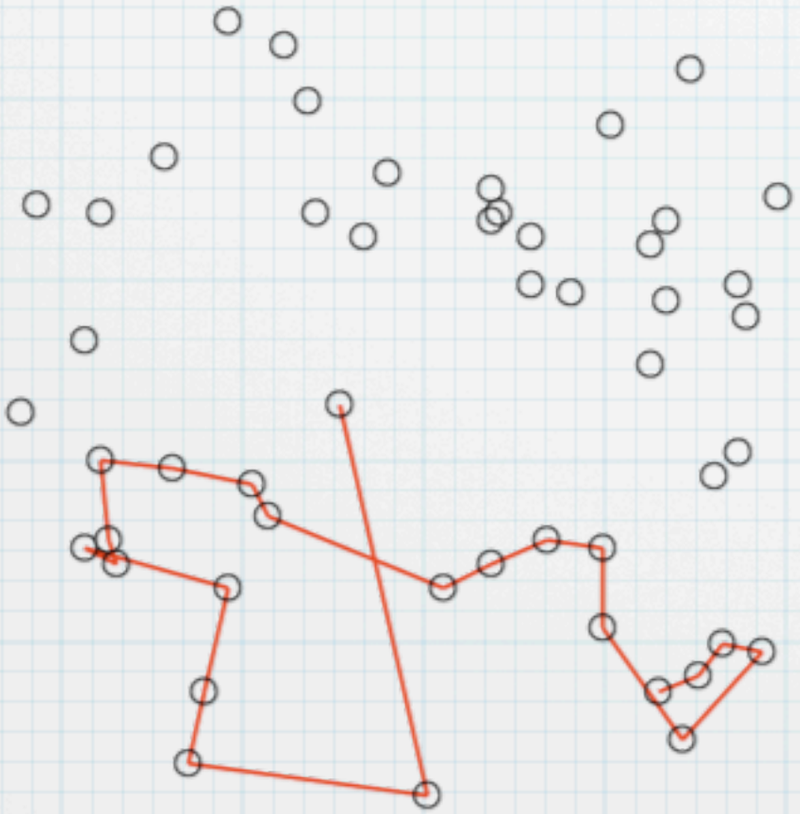
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



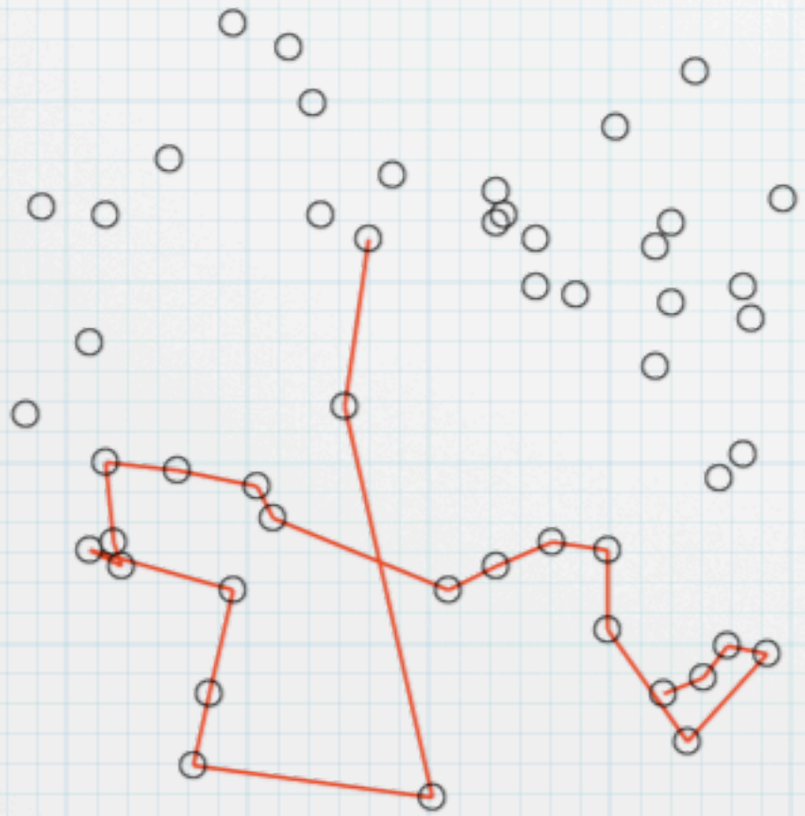
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



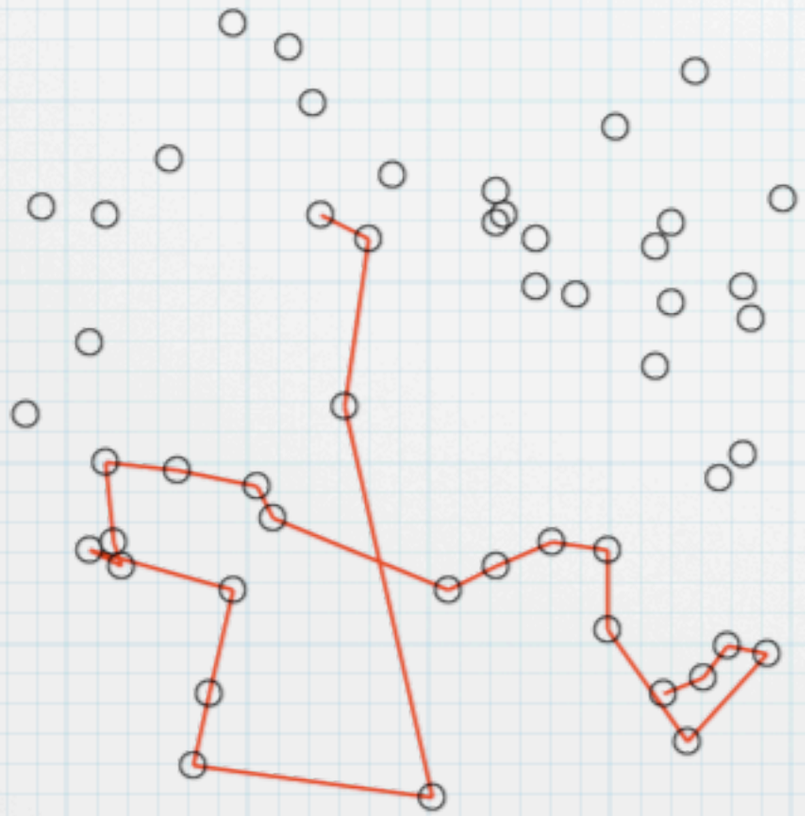
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



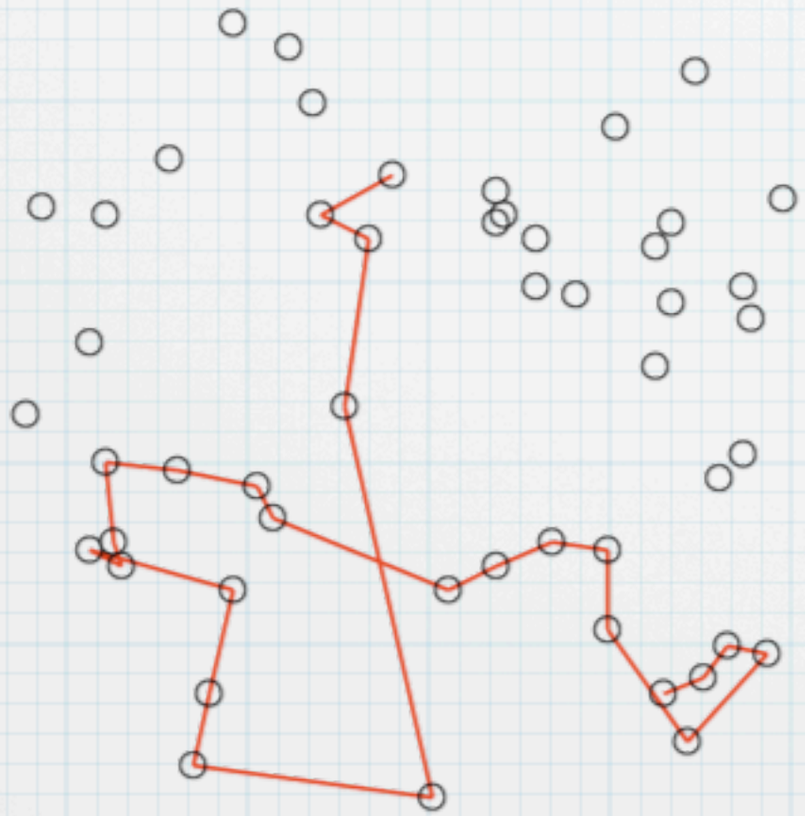
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



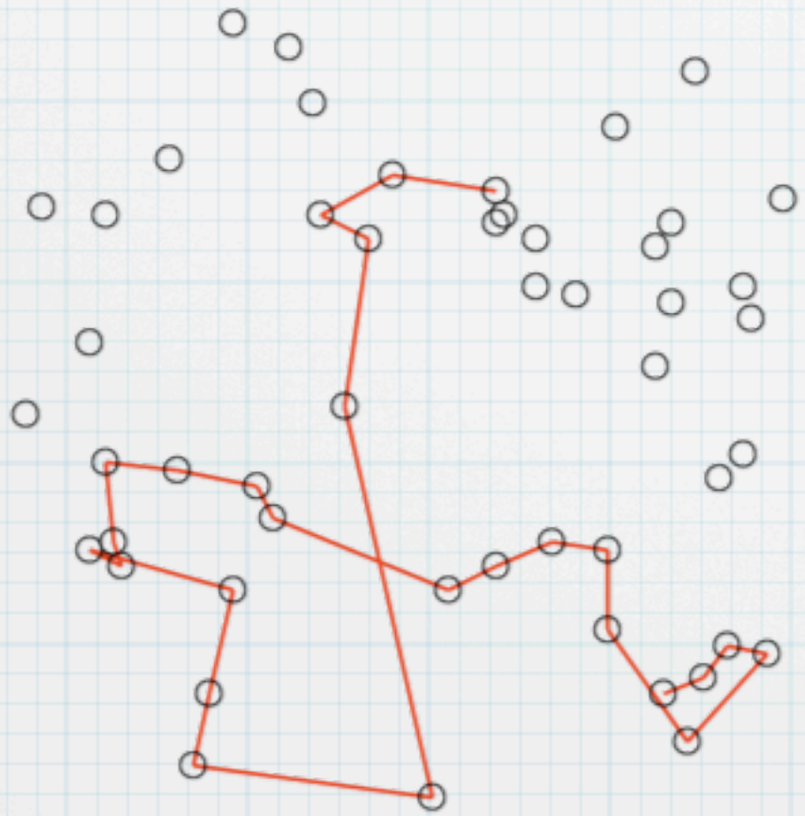
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



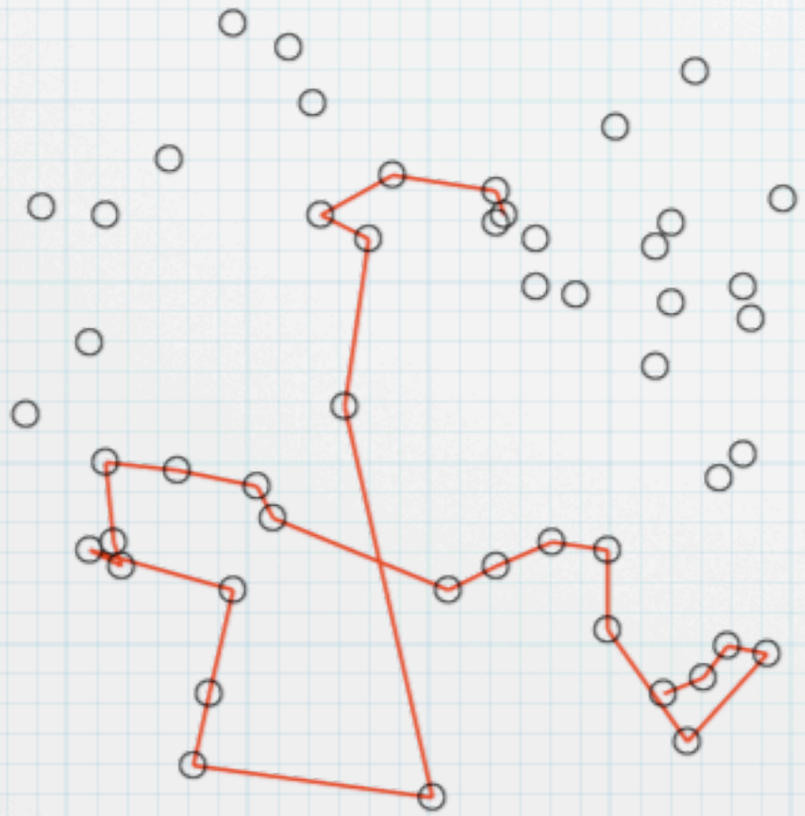
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



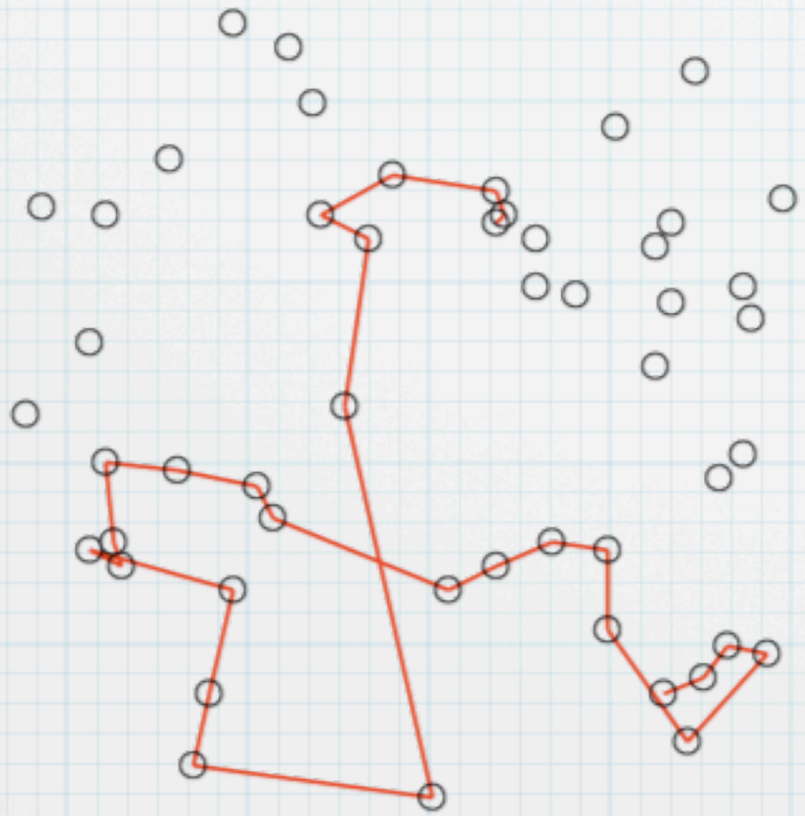
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



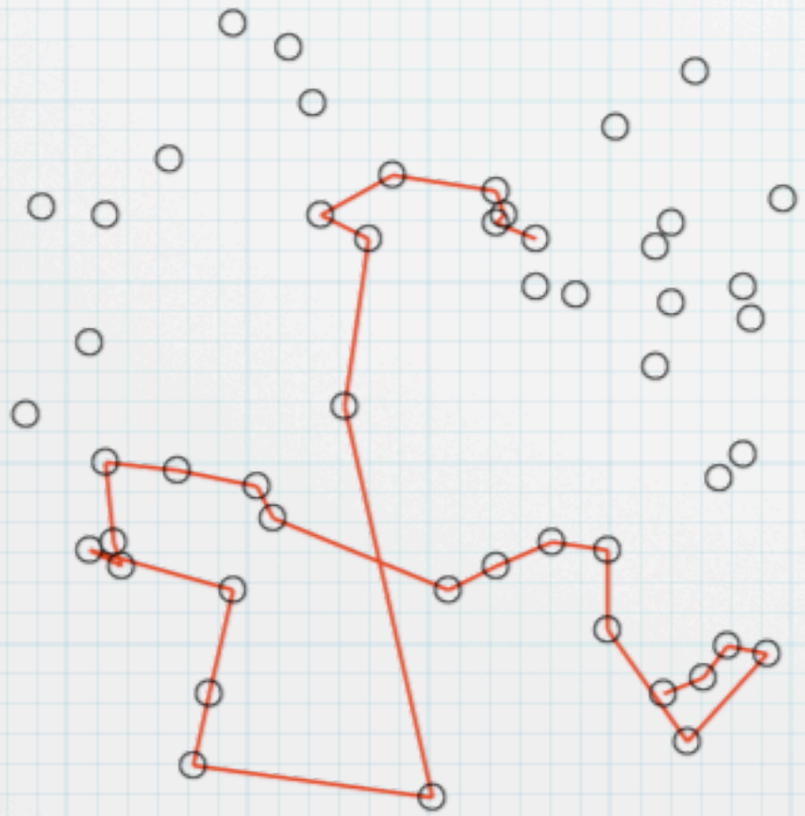
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



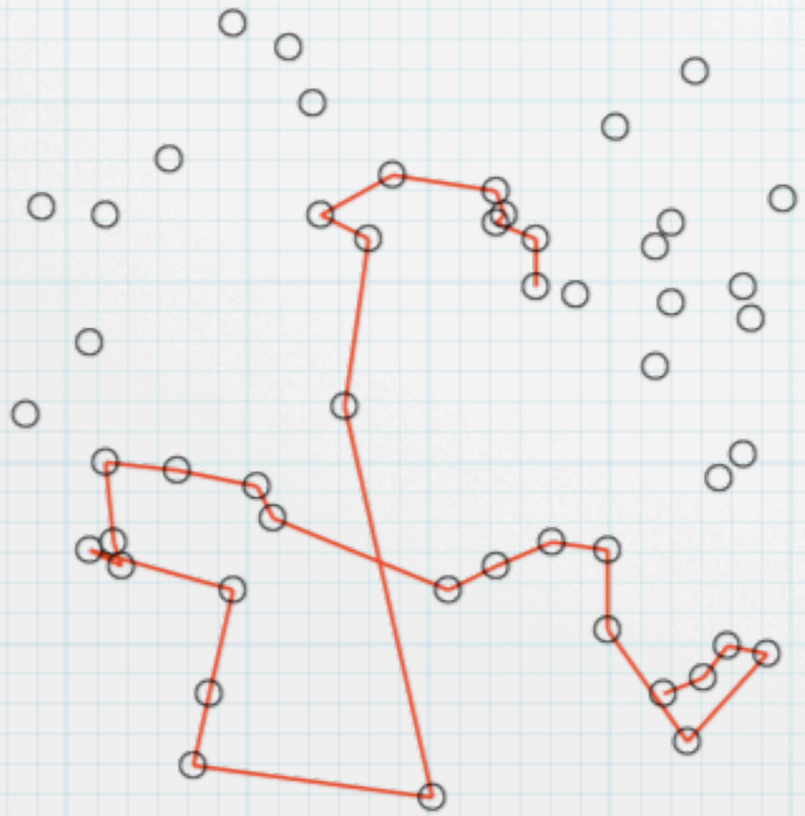
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



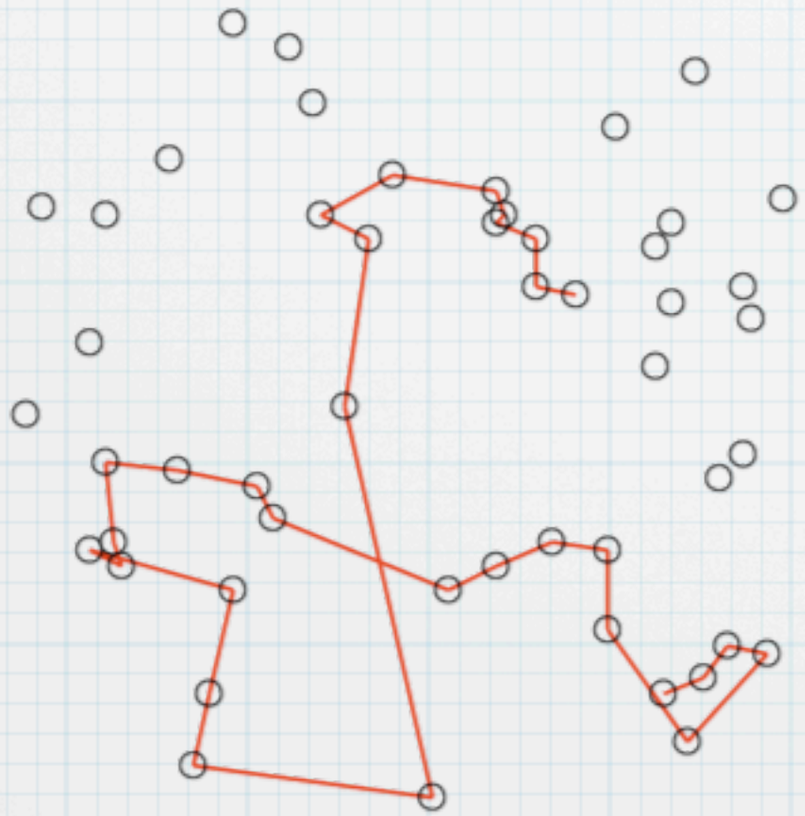
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



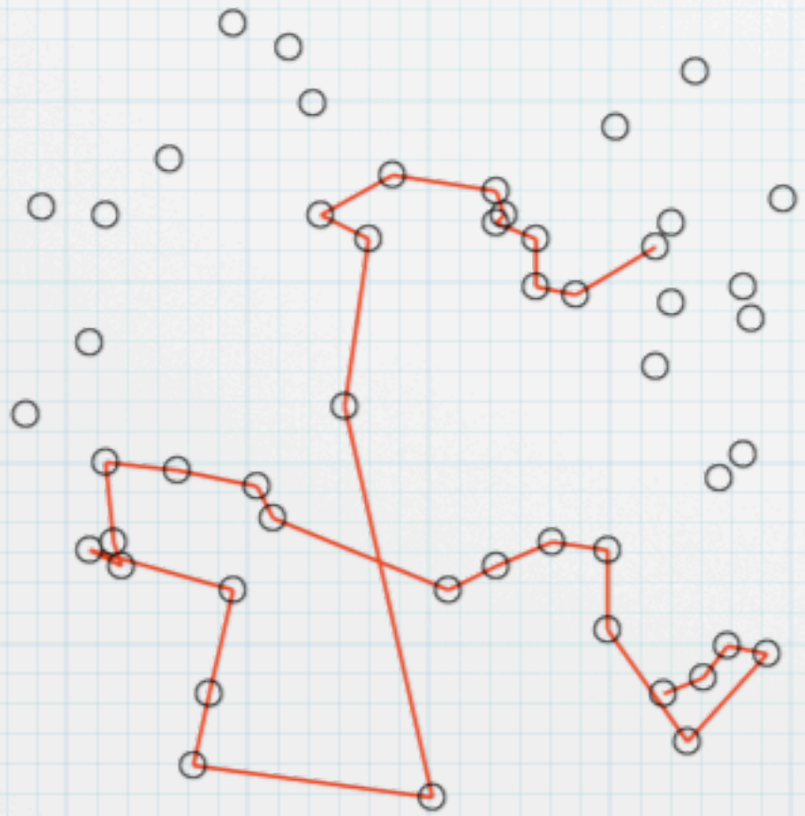
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



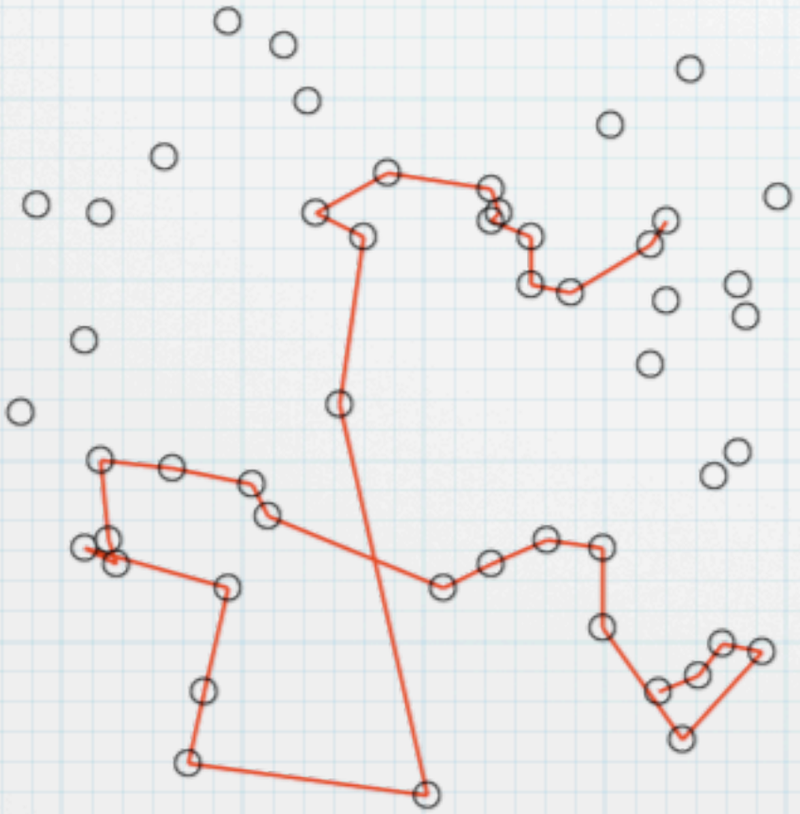
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



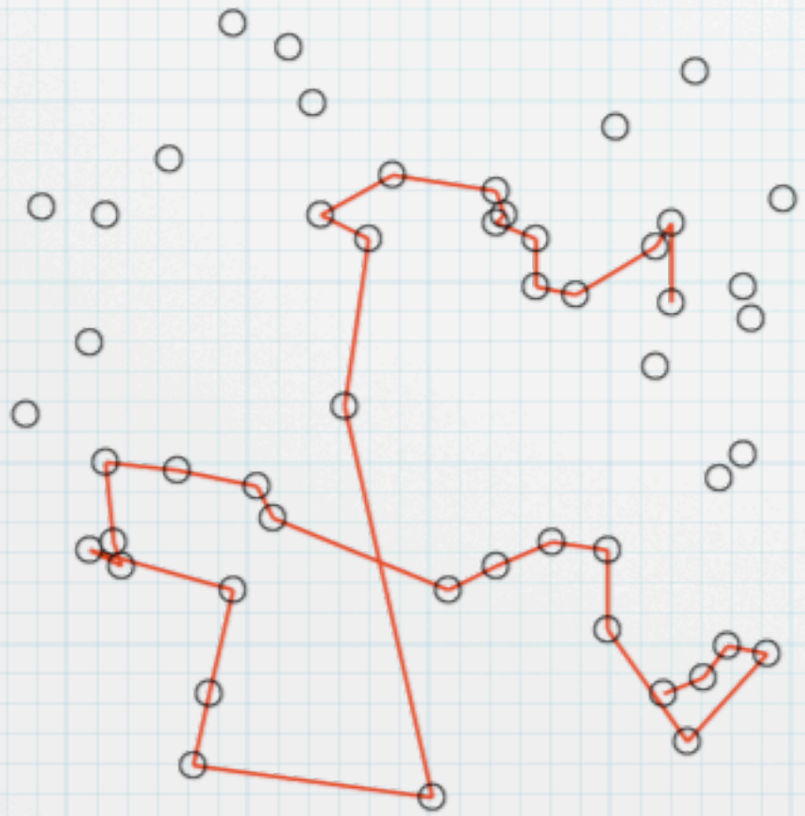
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



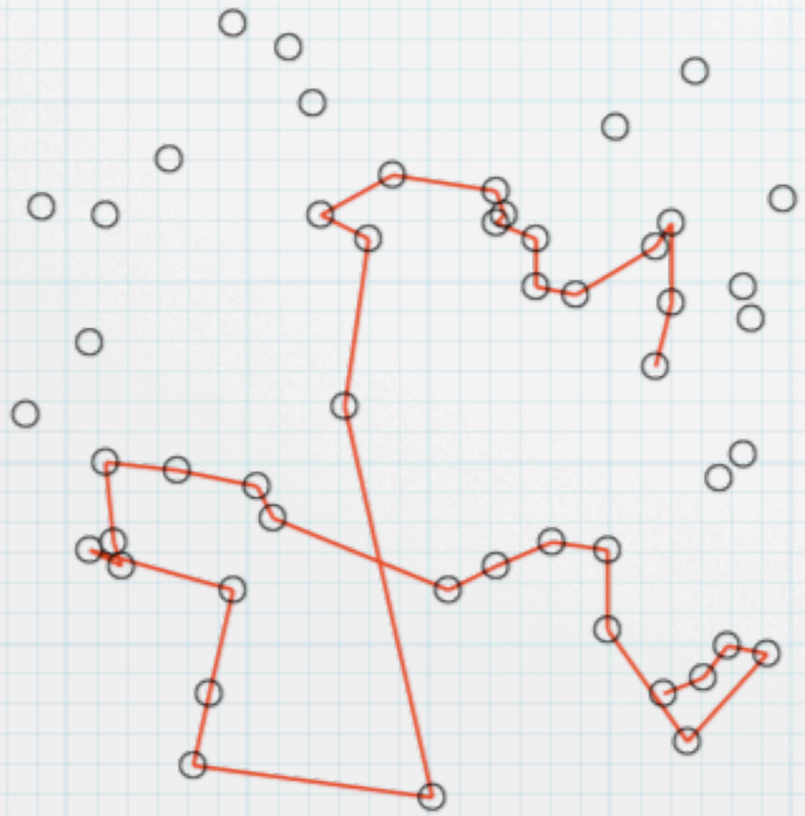
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



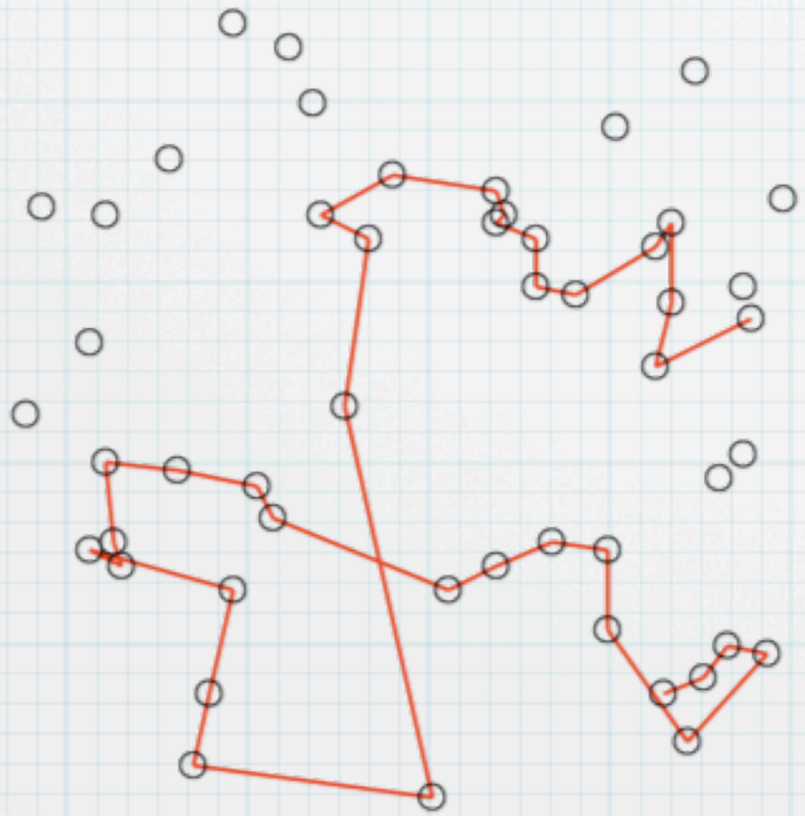
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



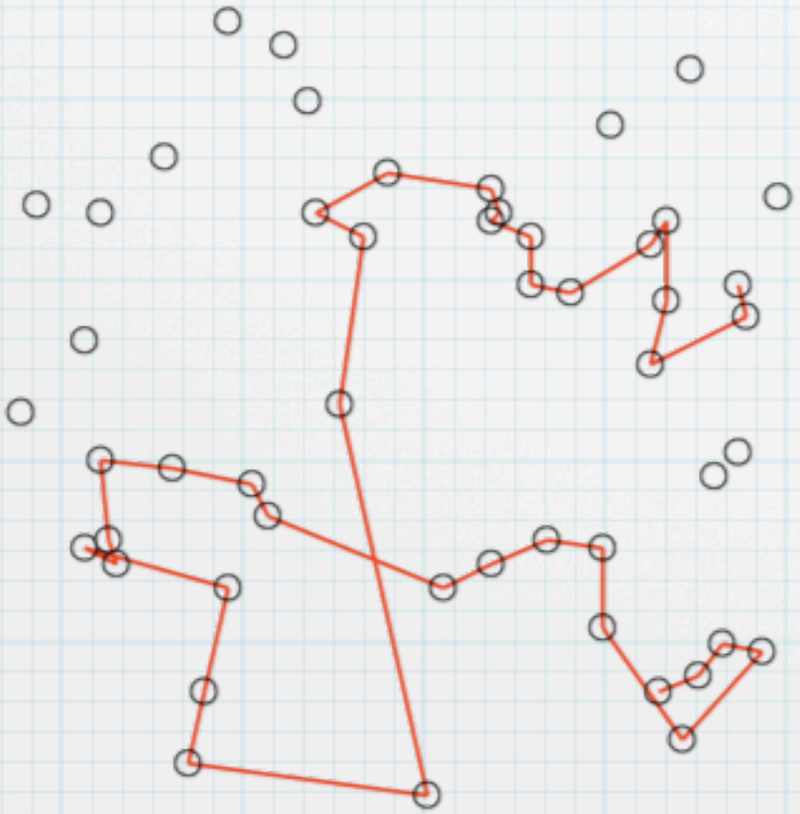
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



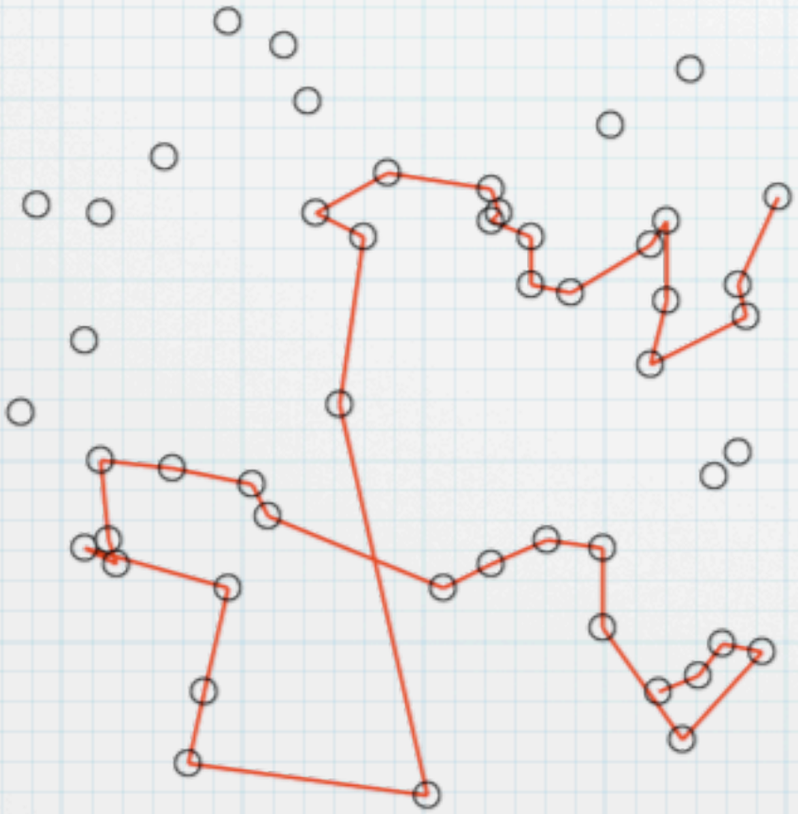
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



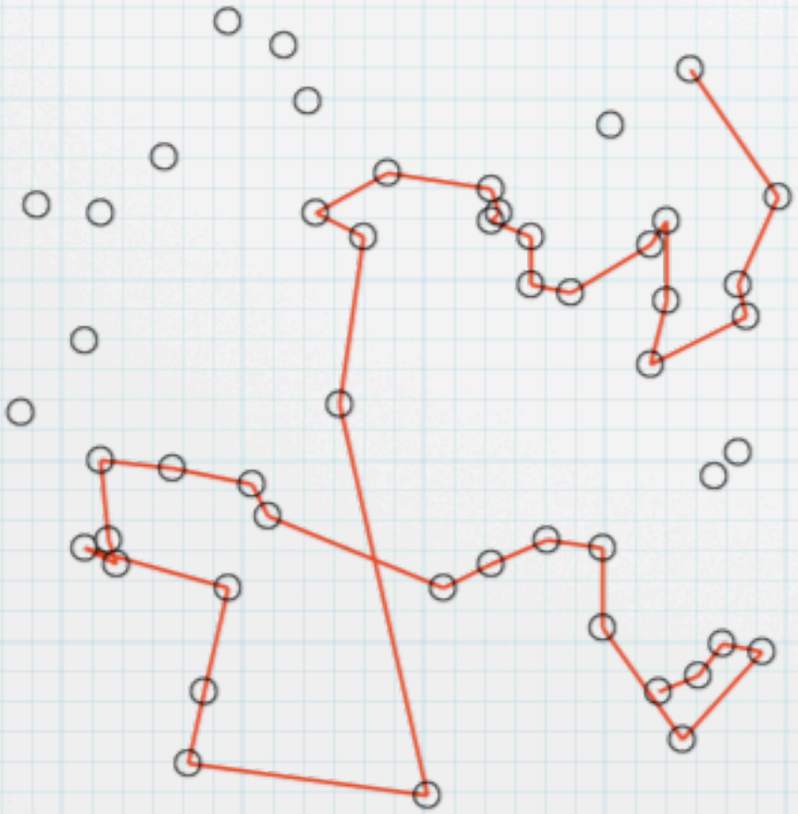
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



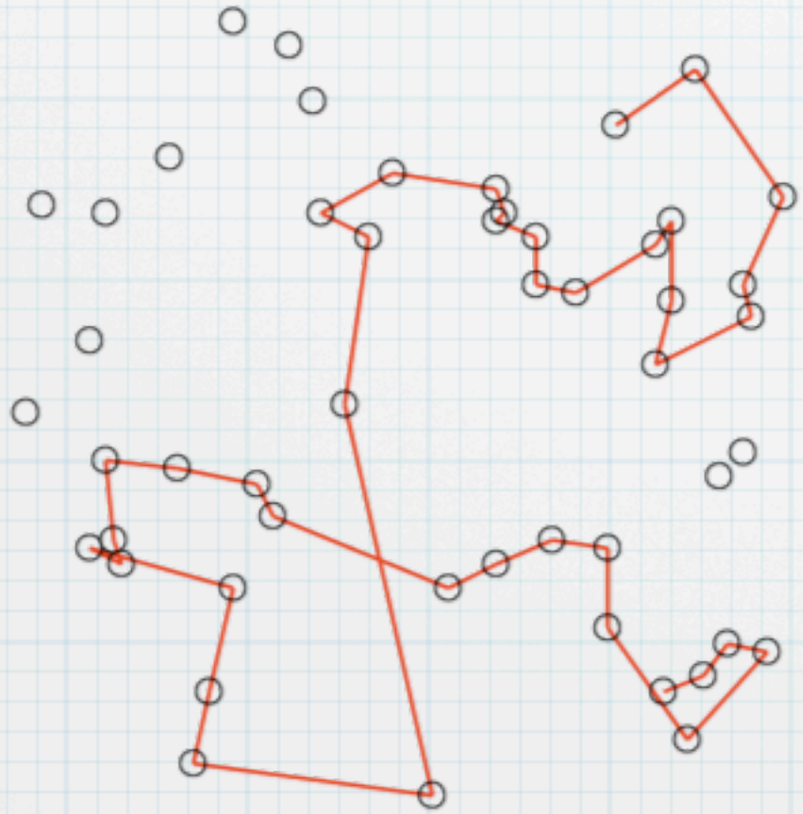
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



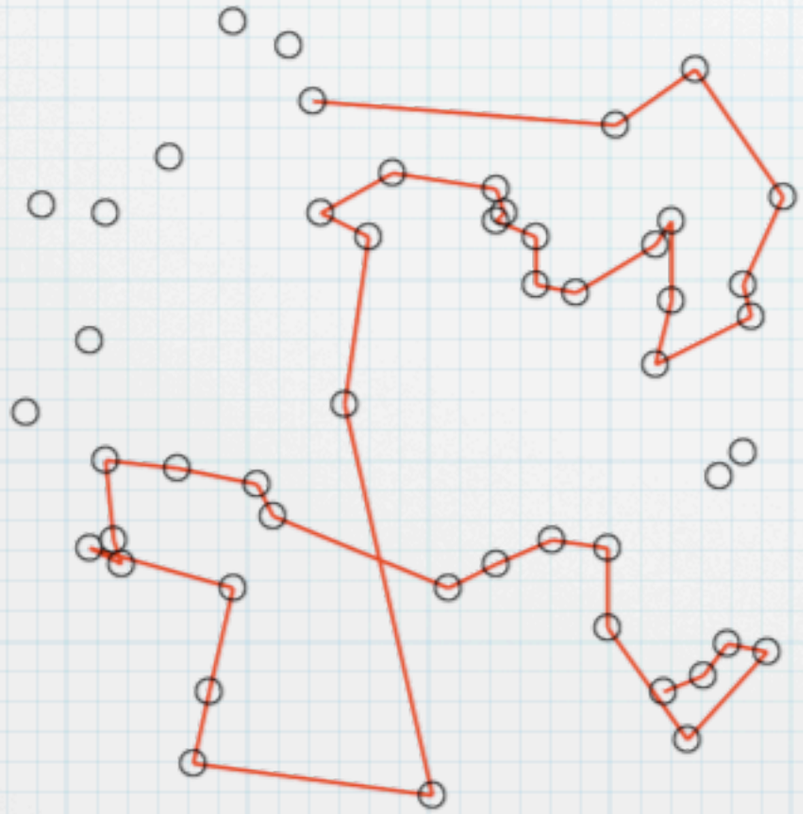
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



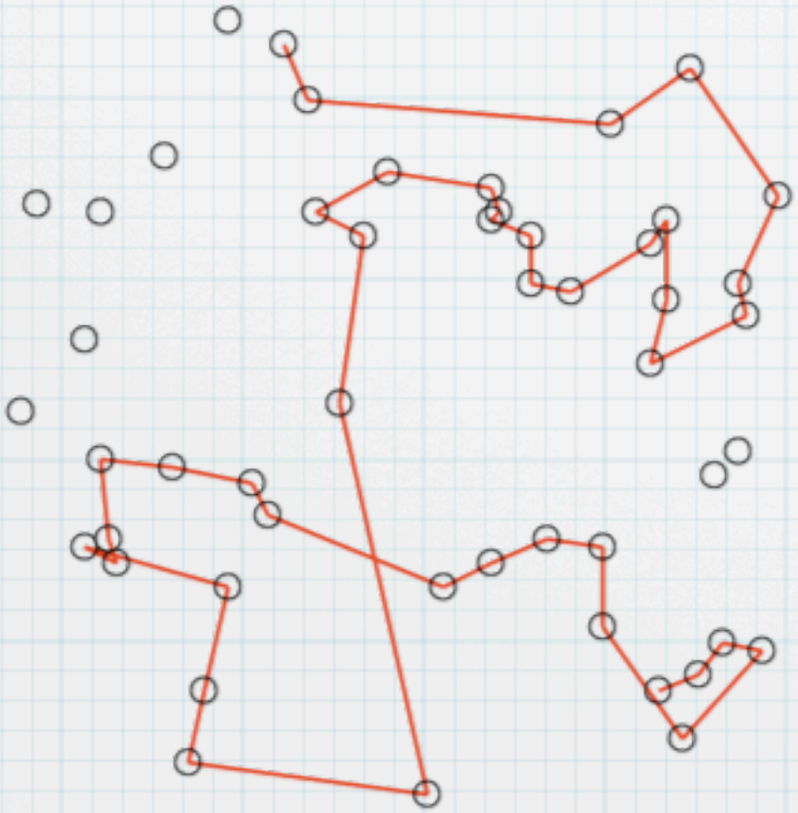
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



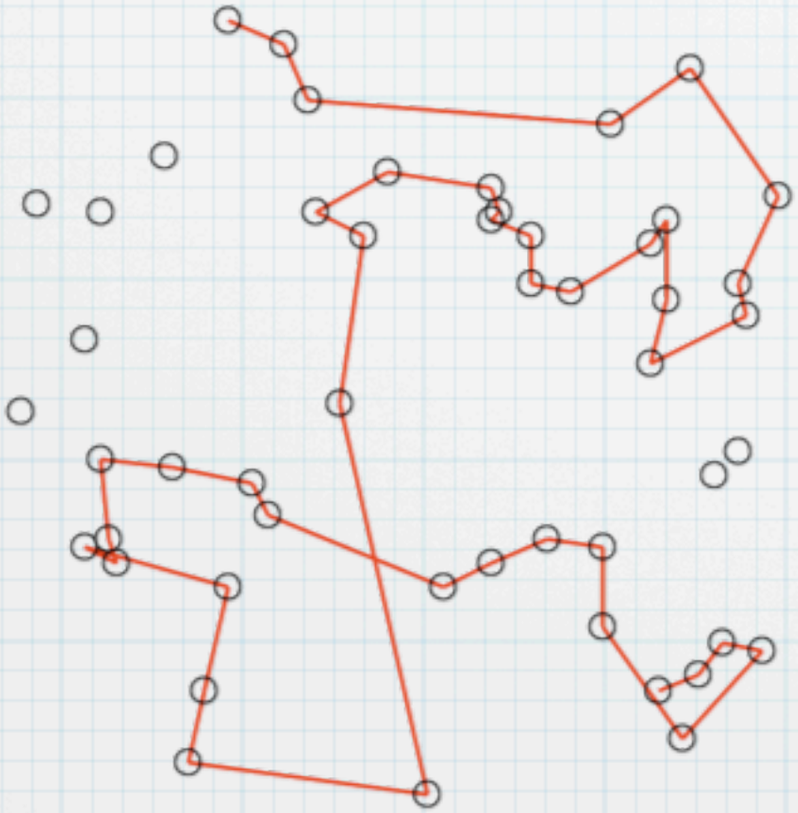
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



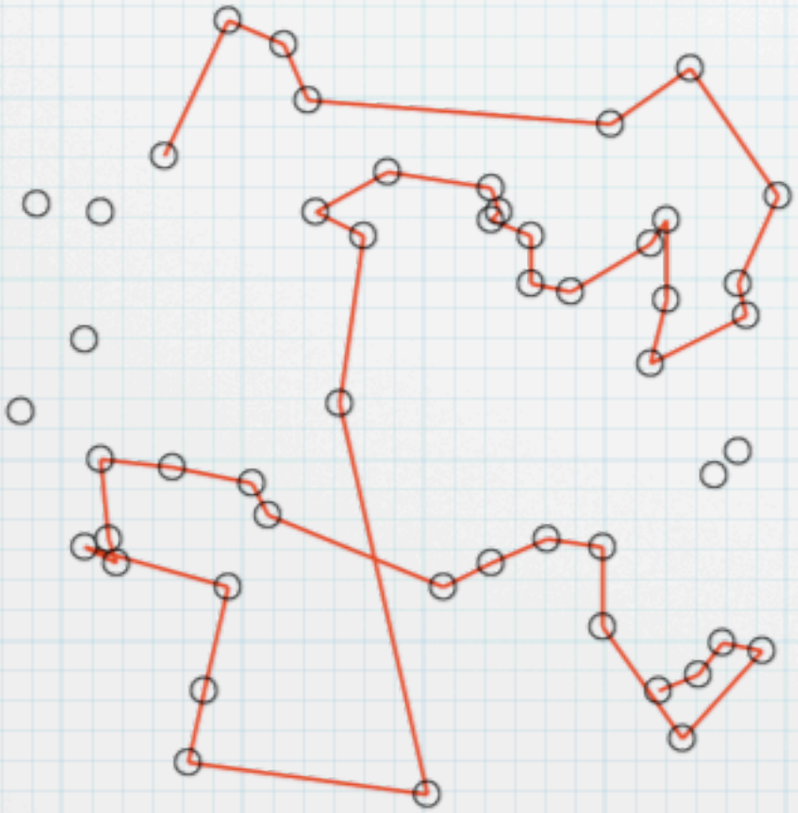
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



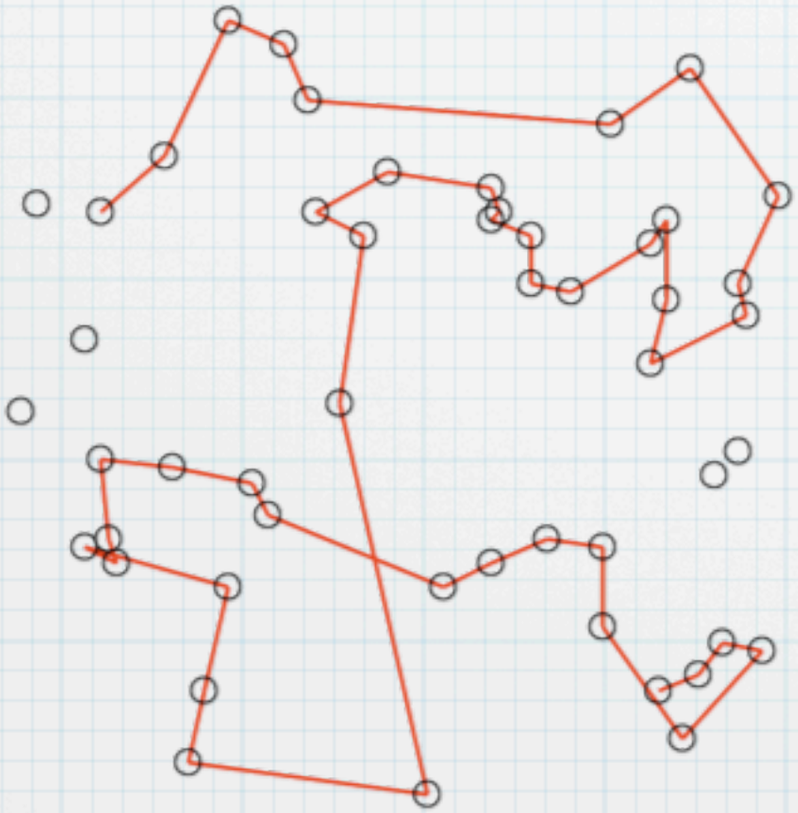
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



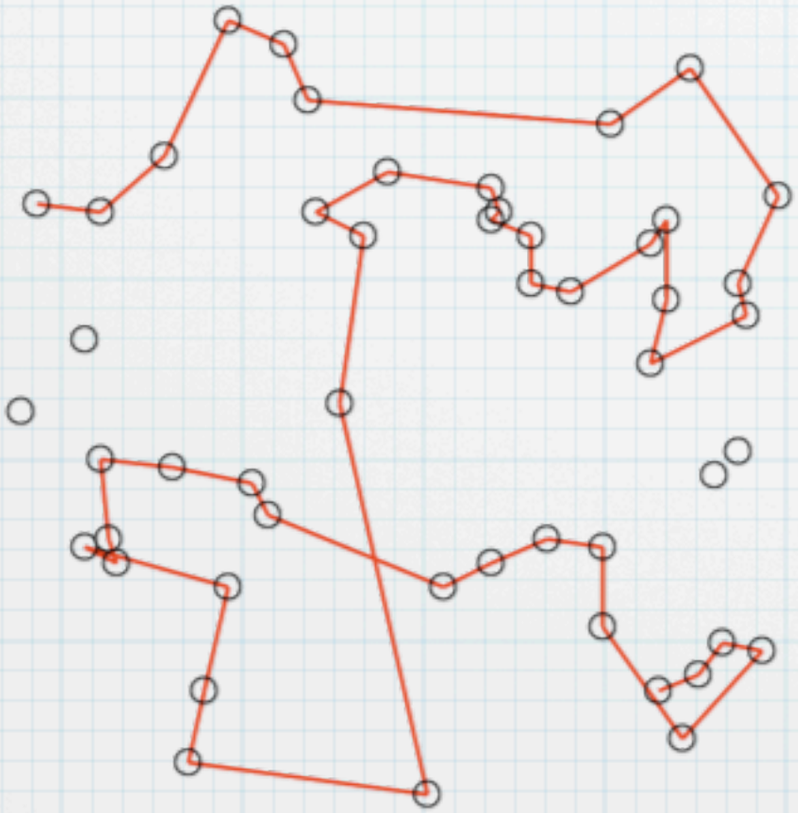
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



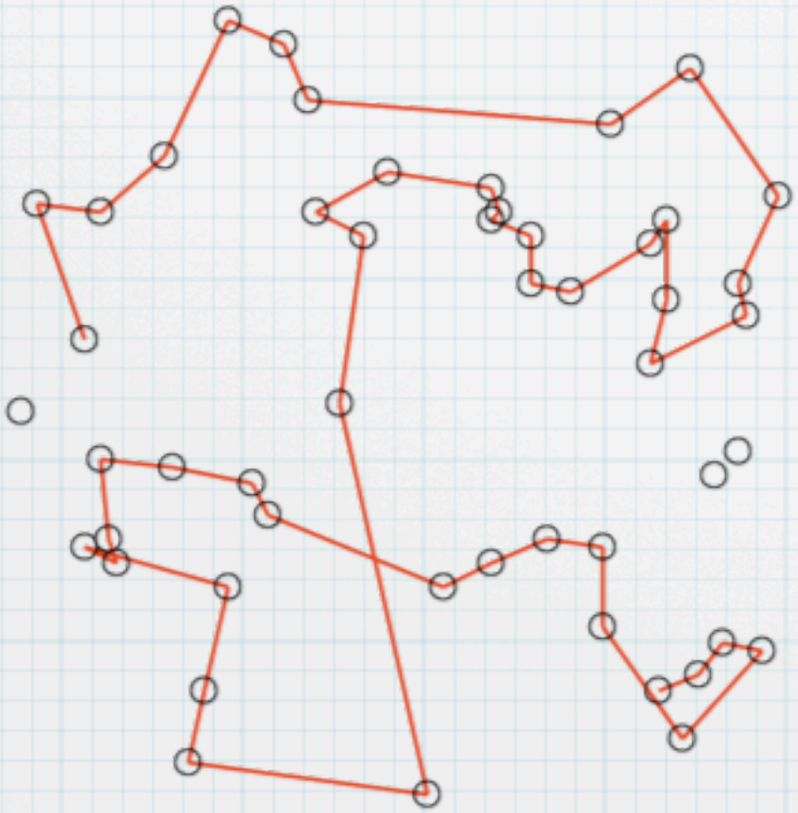
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



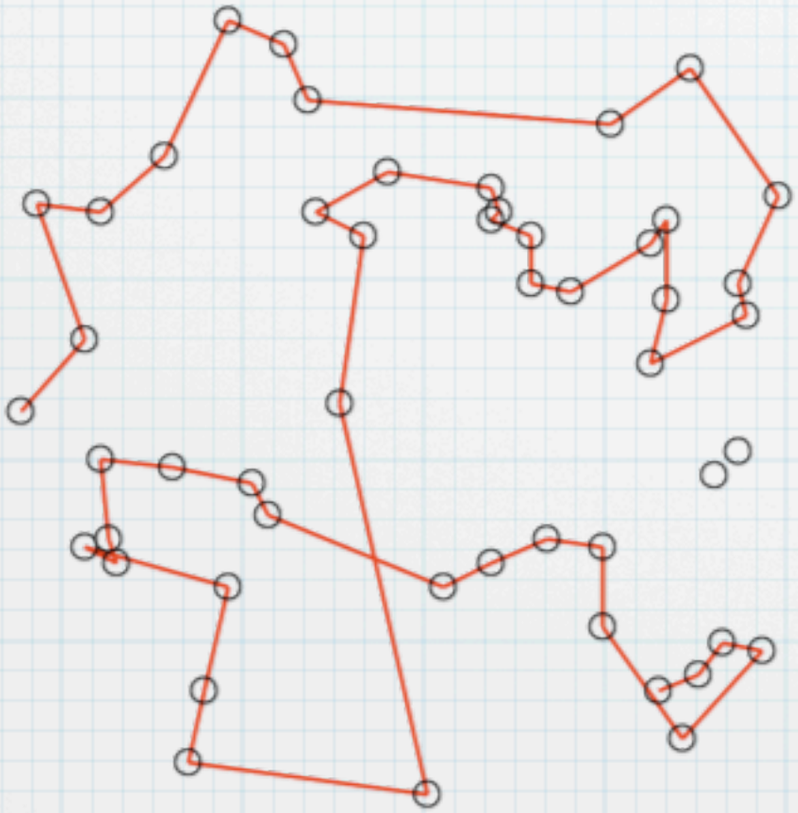
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



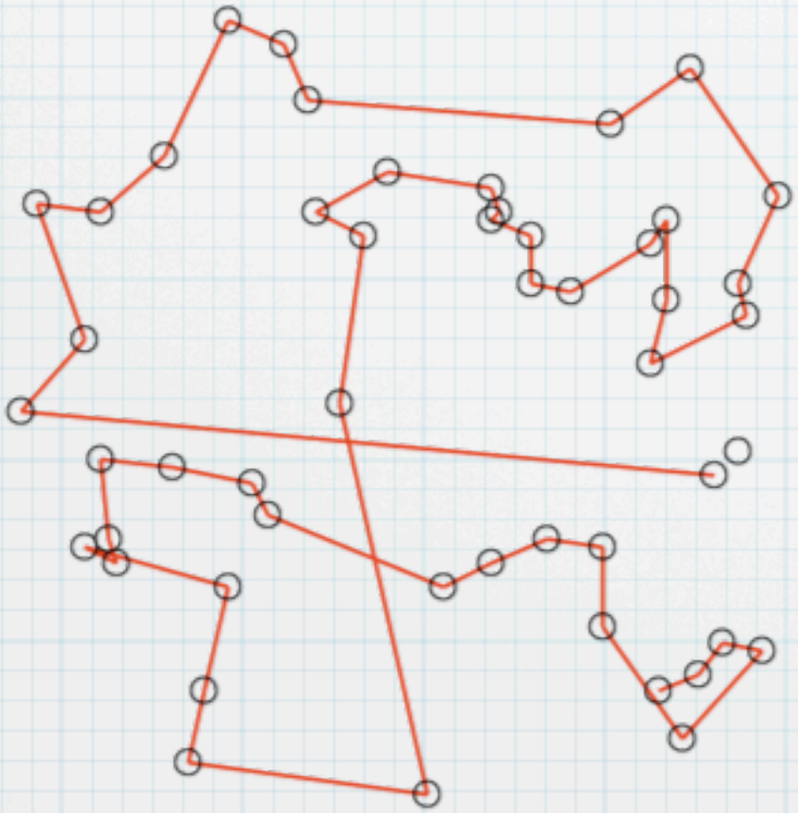
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

* Starte an einem beliebigen Knoten.
* Füge einen Knoten k an den Endknoten i des Pfades P an, der
  * noch nicht besucht wurde,
  * und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
* Wiederhole, bis alle Knoten eingefügt sind.
* Verbinde den letzten Knoten mit dem Startknoten.
* Beispiel:



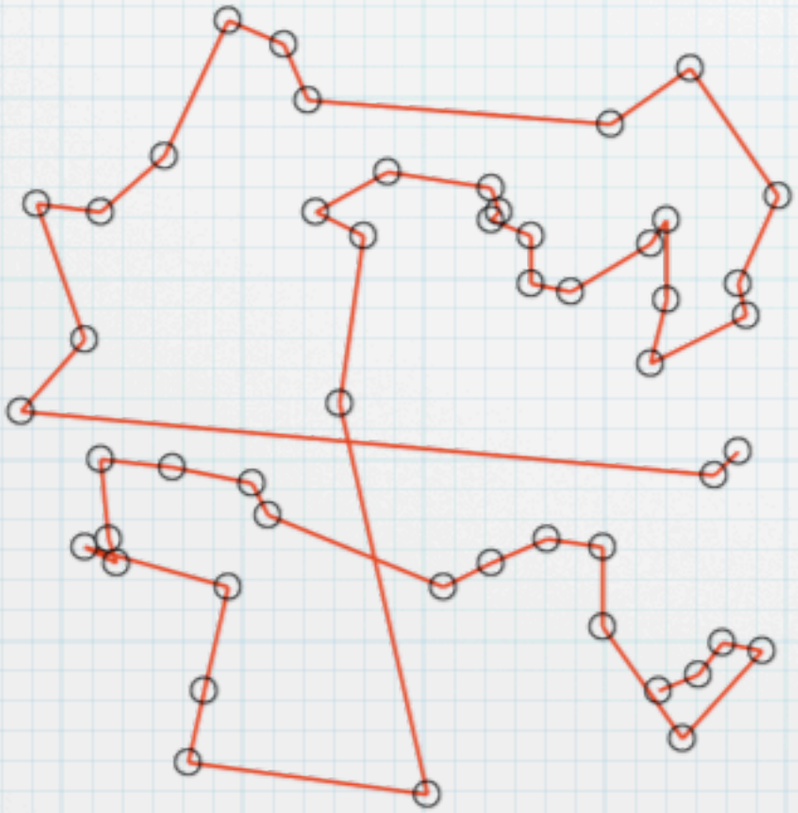
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



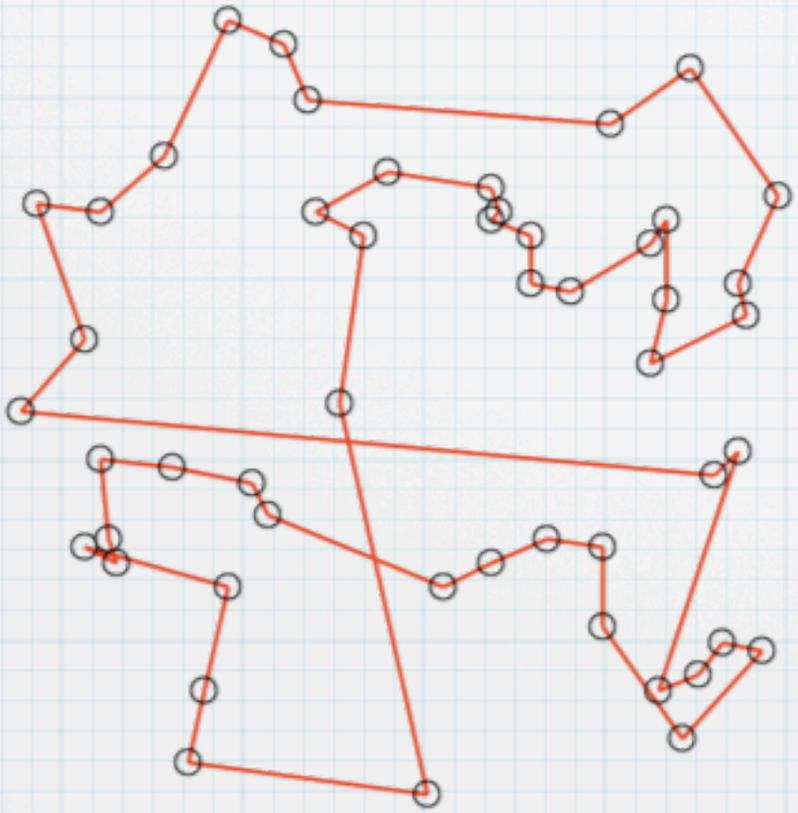
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



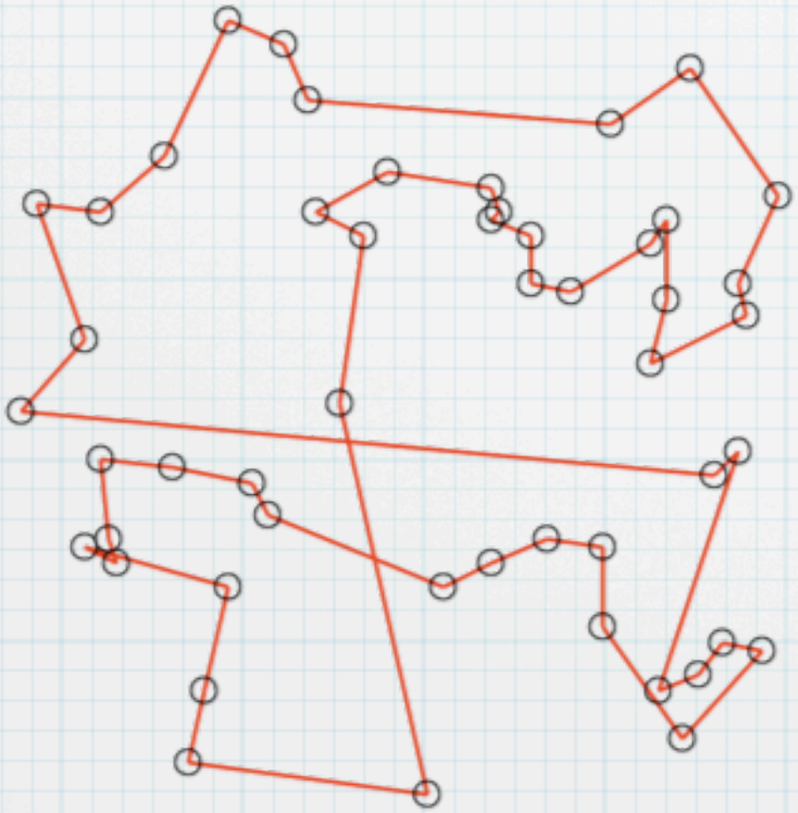
Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



Greedy-Heuristiken: Die nächster-Nachbar-Strategie

- Starte an einem beliebigen Knoten.
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und die kürzeste Distanz zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j} := c_{i,j}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $z^{greedy} = 677$



Parametrisches Greedy-Verfahren

Parametrisches Greedy-Verfahren

* Modifikation des nächster-Nachbar-Verfahrens

Parametrisches Greedy-Verfahren

* Modifikation des nächster-Nachbar-Verfahrens
* Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$

Parametrisches Greedy-Verfahren

* Modifikation des nächster-Nachbar-Verfahrens
* Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
* Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).

Parametrisches Greedy-Verfahren

* Modifikation des nächster-Nachbar-Verfahrens
* Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
* Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
* Füge einen Knoten k an den Endknoten i des Pfades P an, der

Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,

Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.

Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.

Parametrisches Greedy-Verfahren

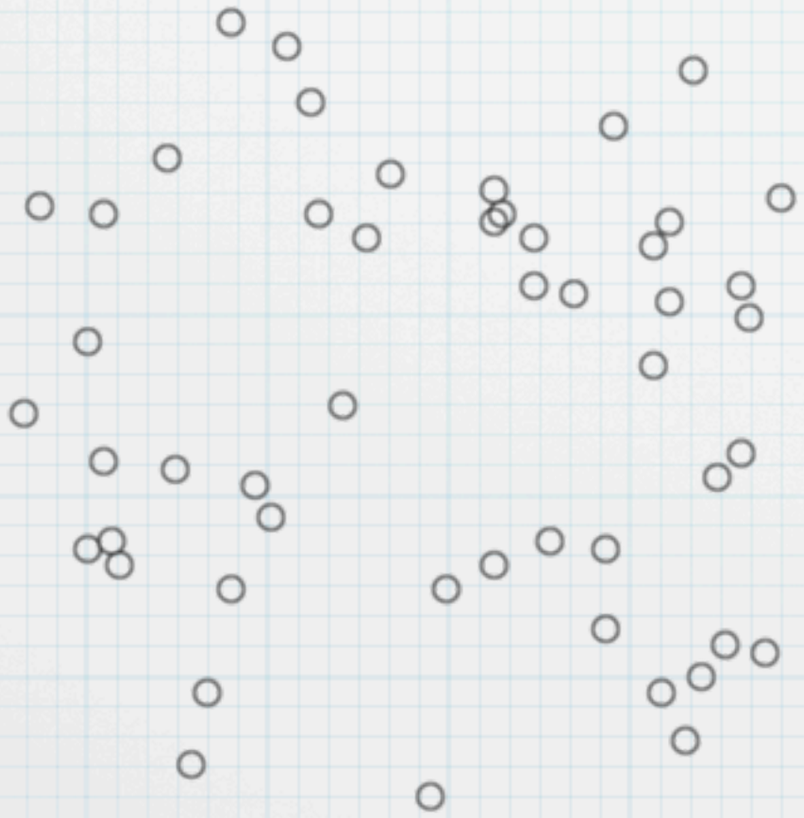
- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.

Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:

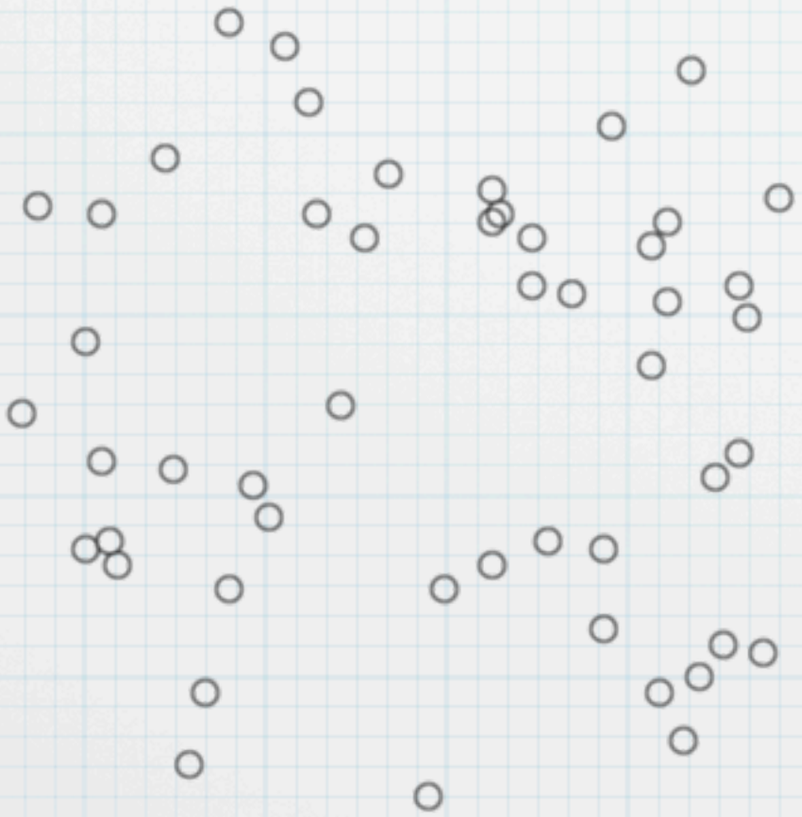
Parametrisches Greedy-Verfahren

- Modifikation des nächster–Nachbar–Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel:



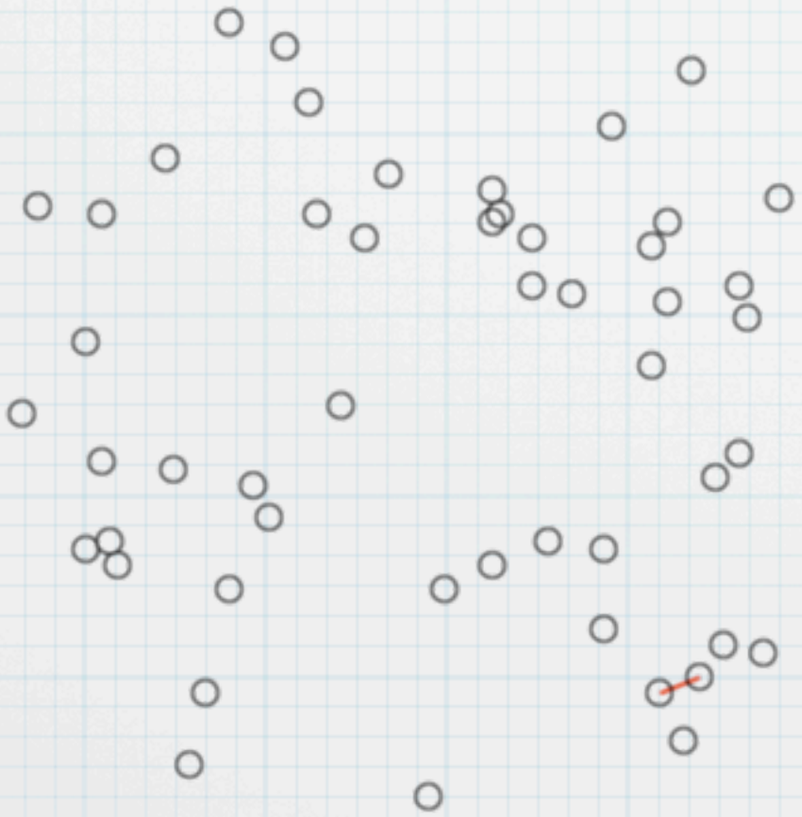
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



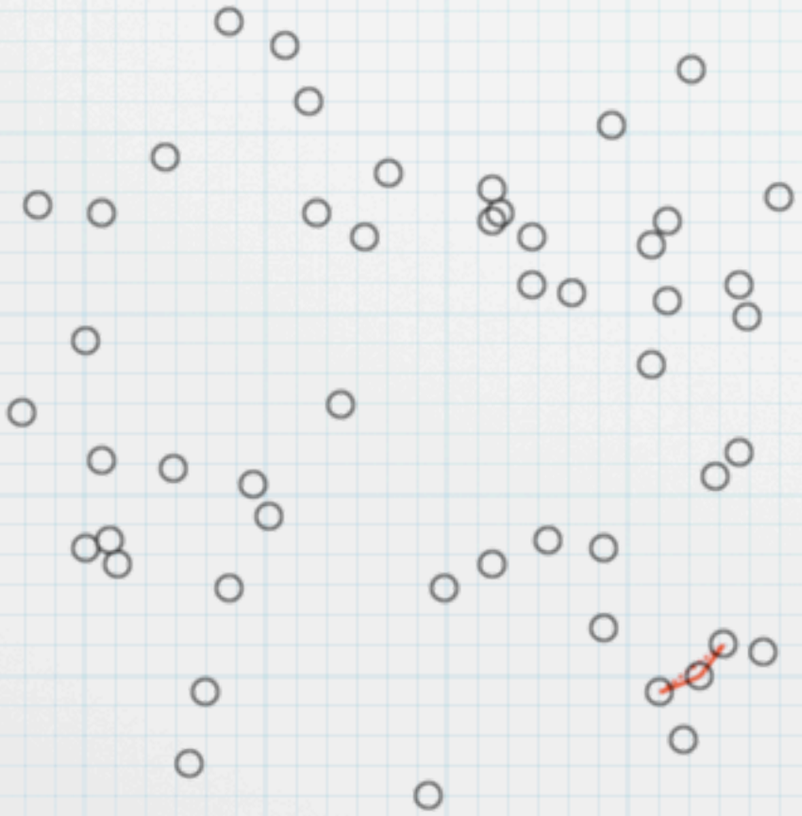
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



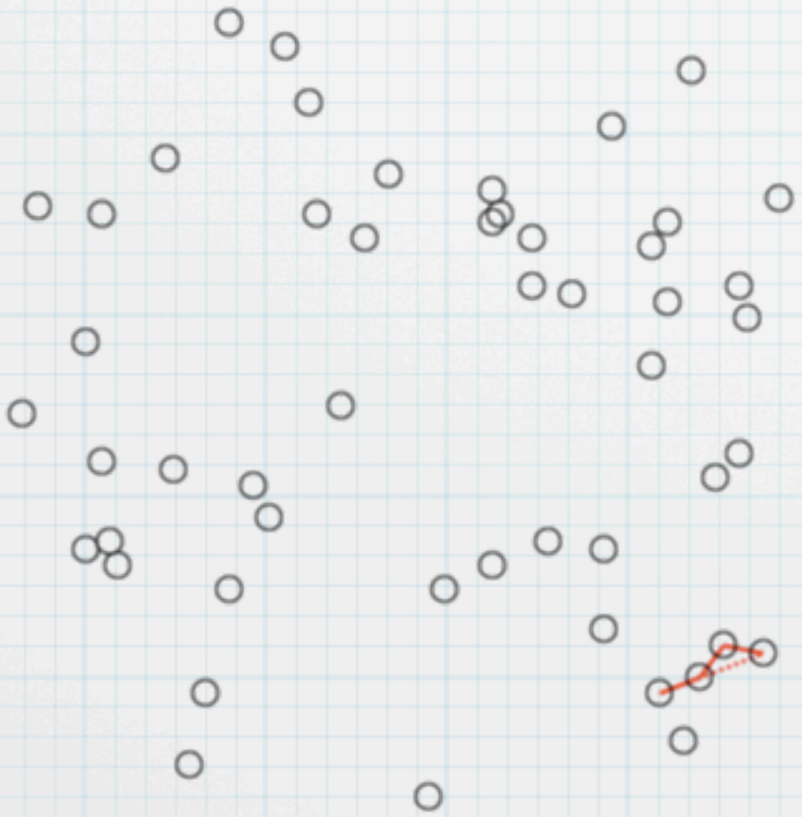
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



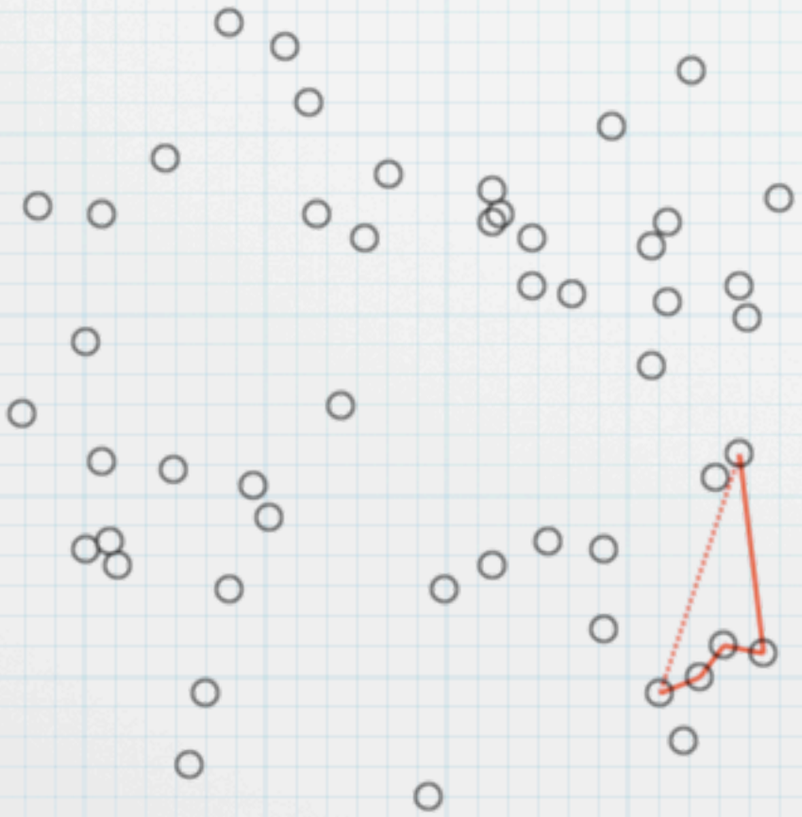
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



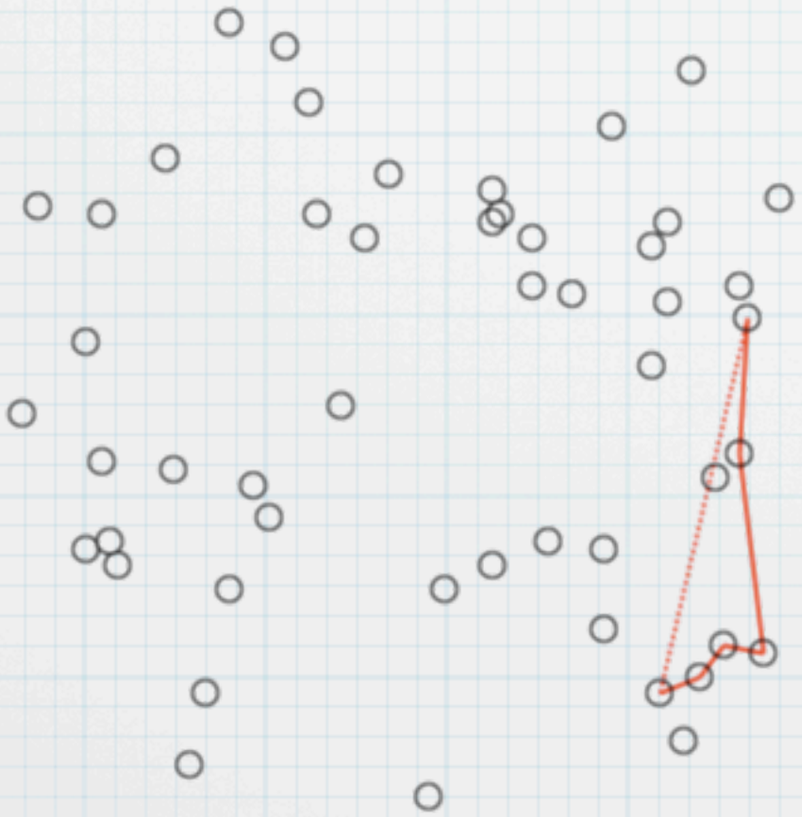
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



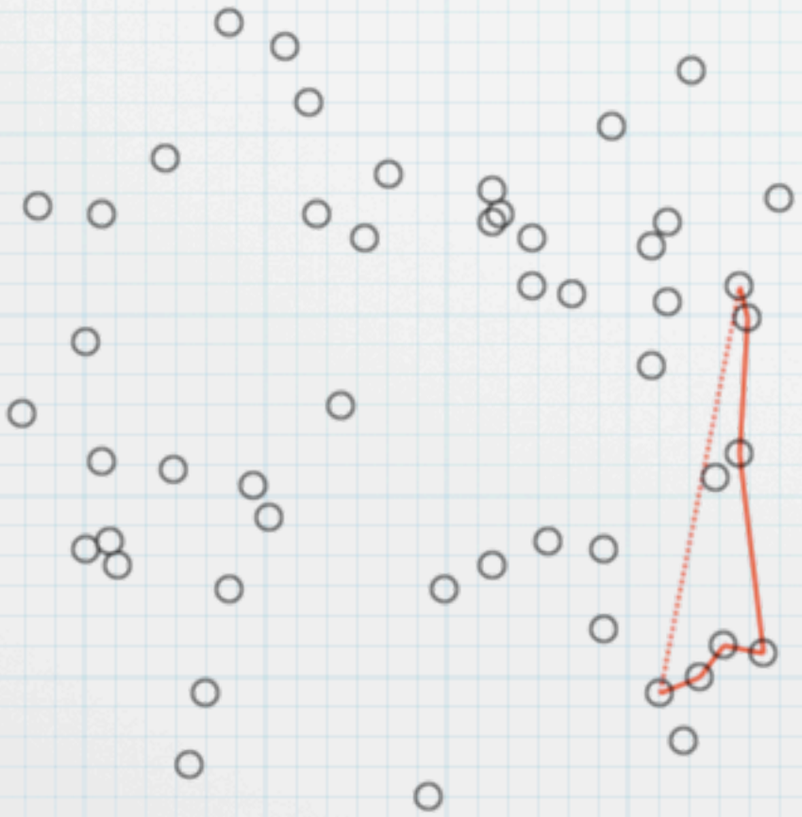
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



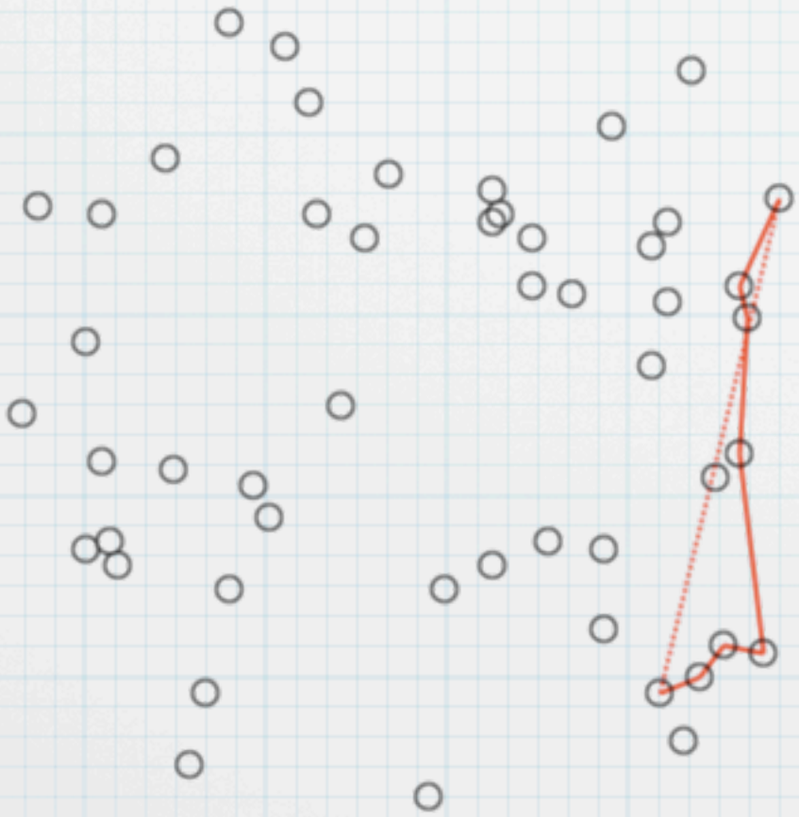
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



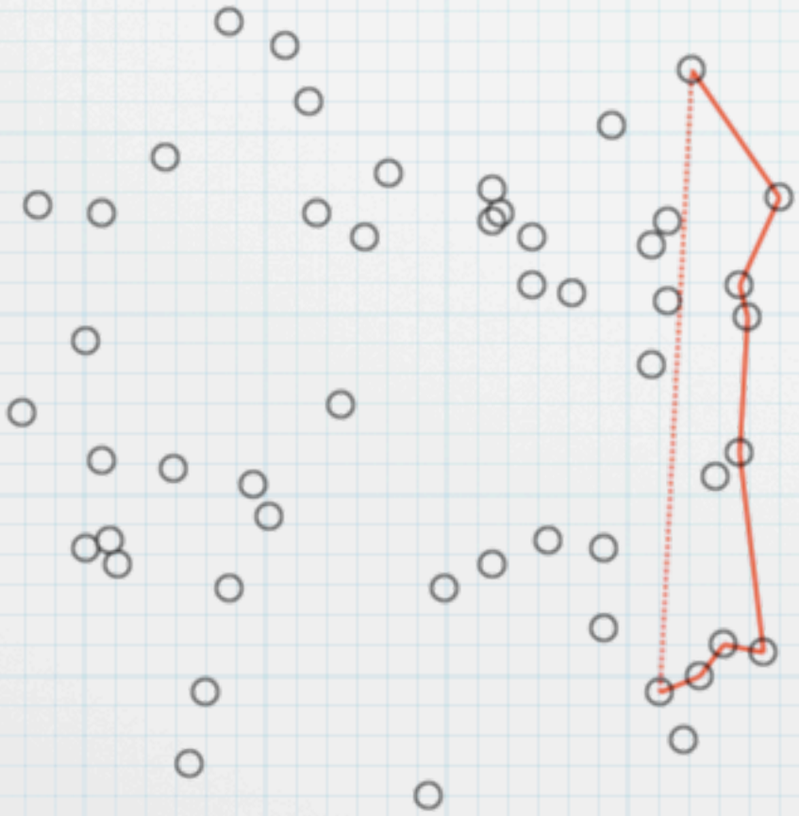
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



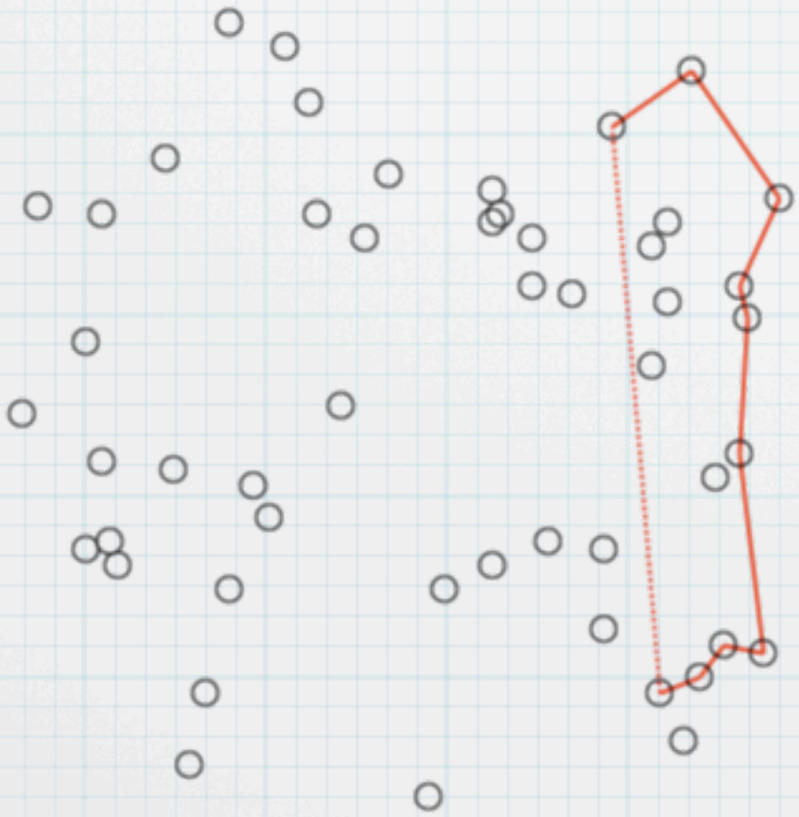
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



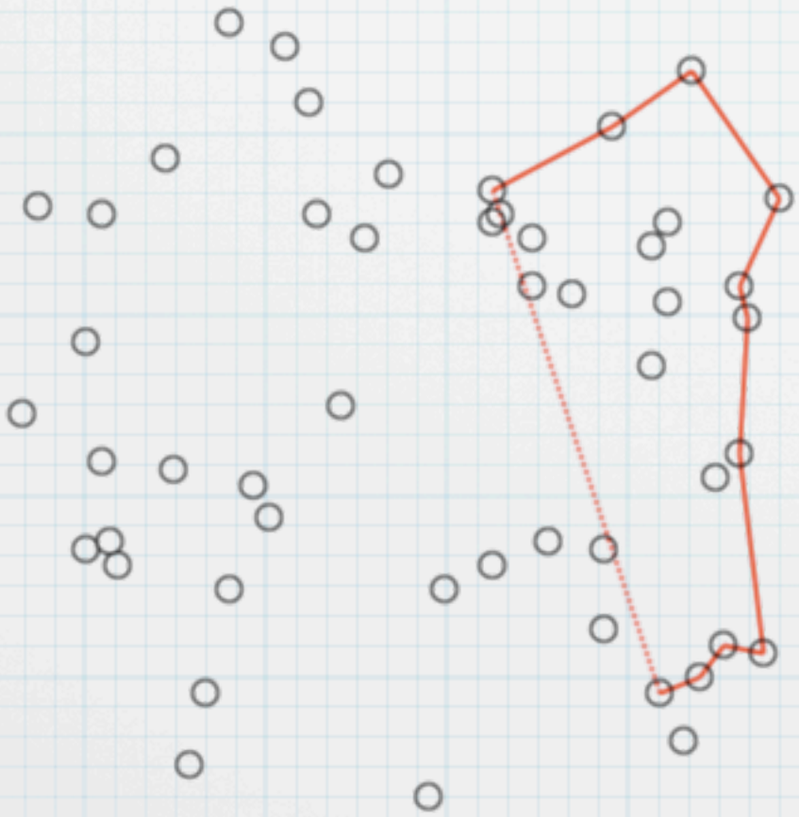
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



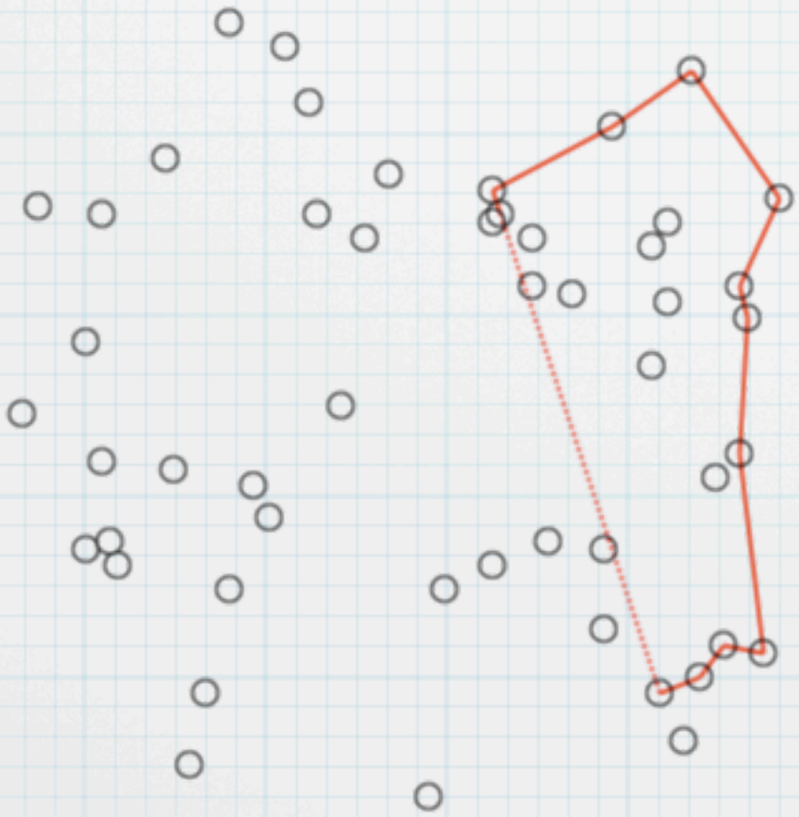
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



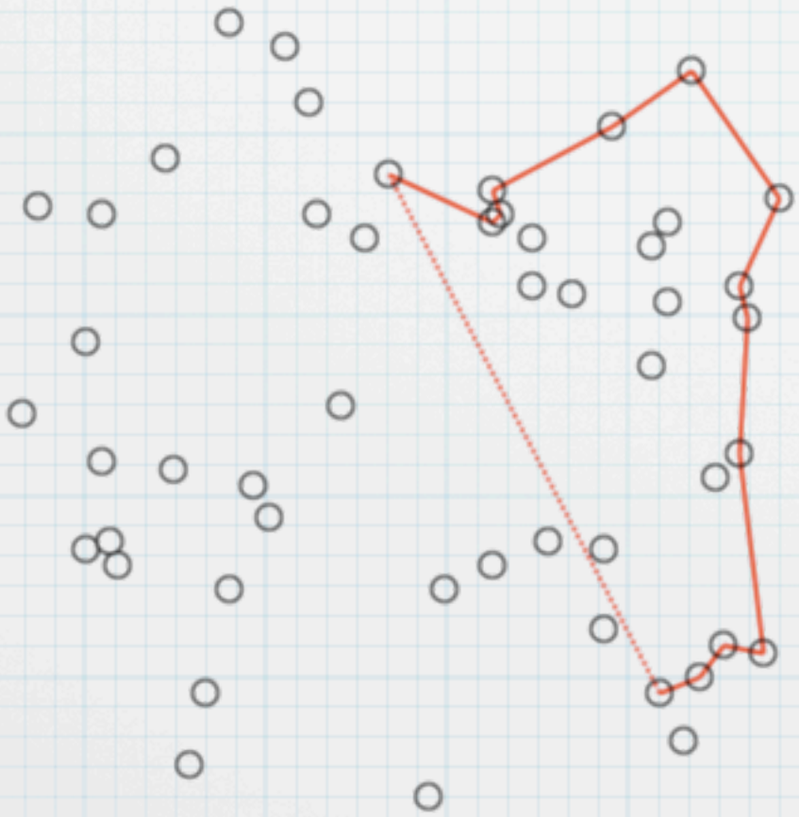
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



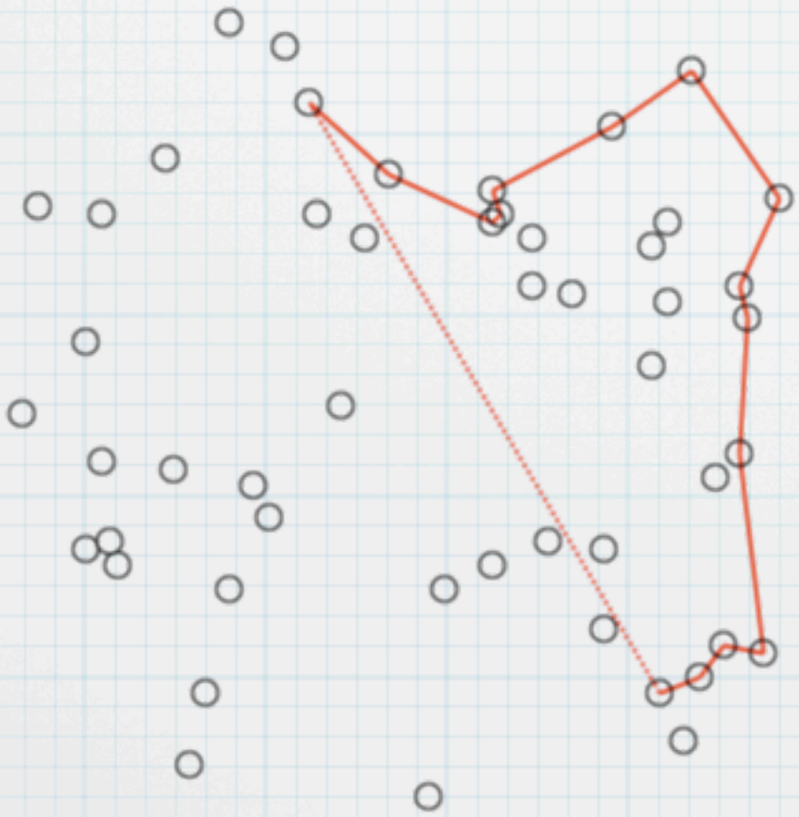
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



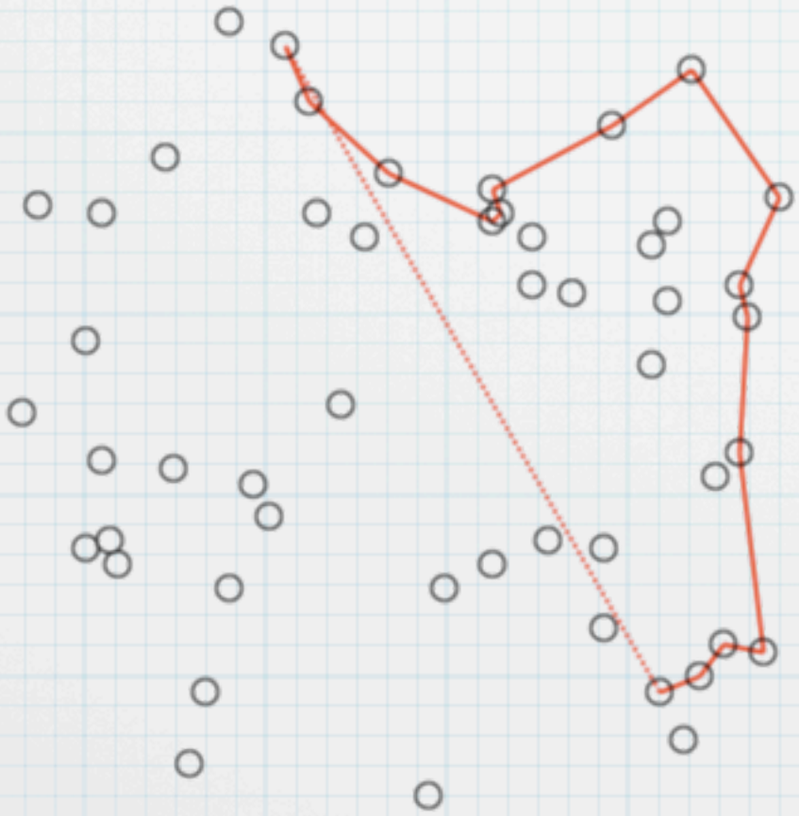
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



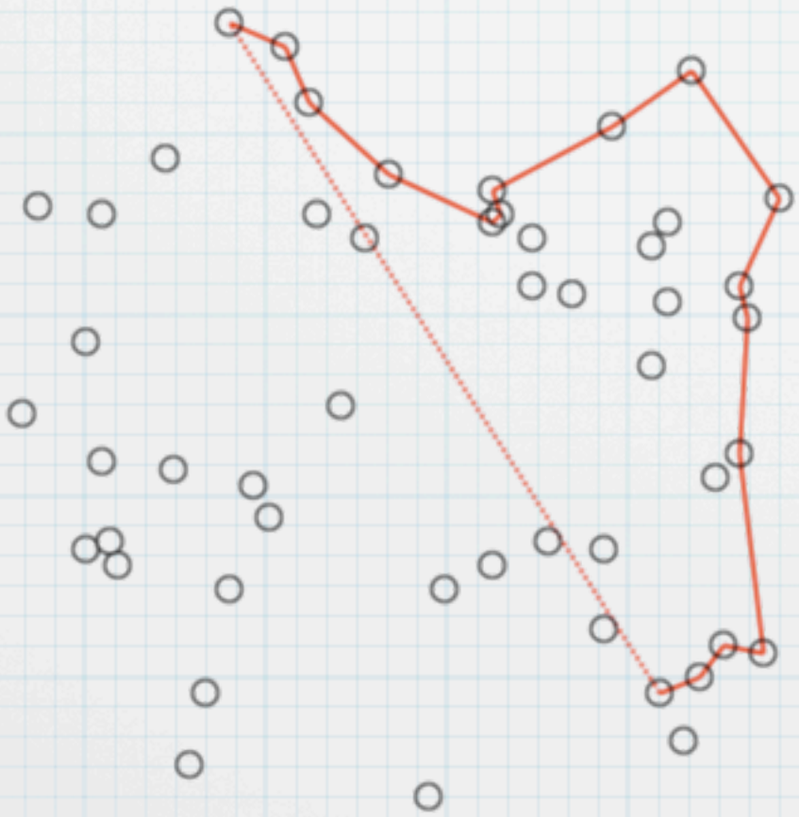
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



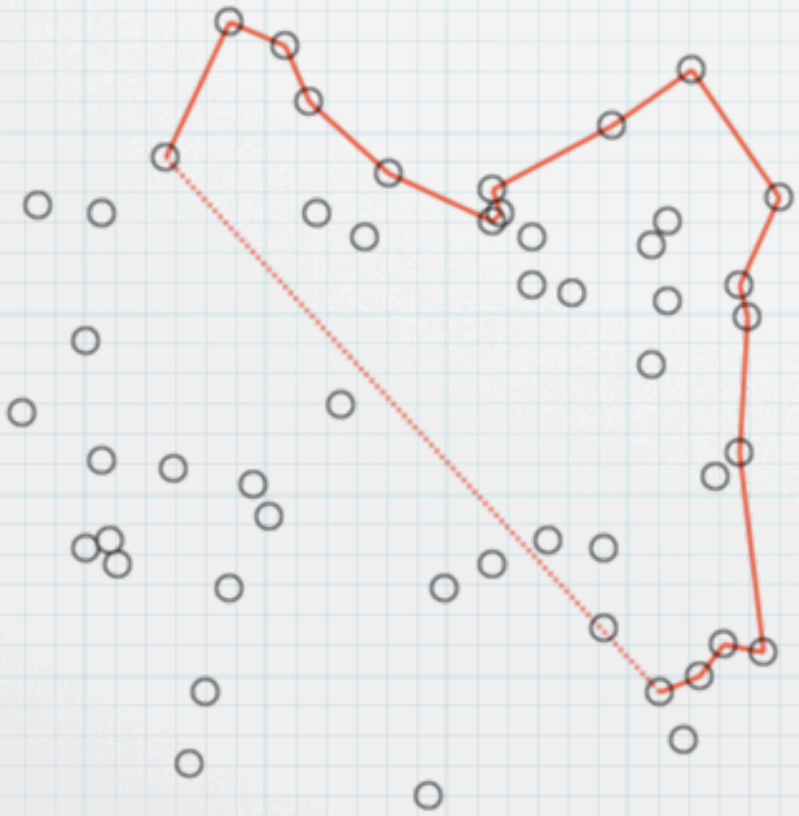
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



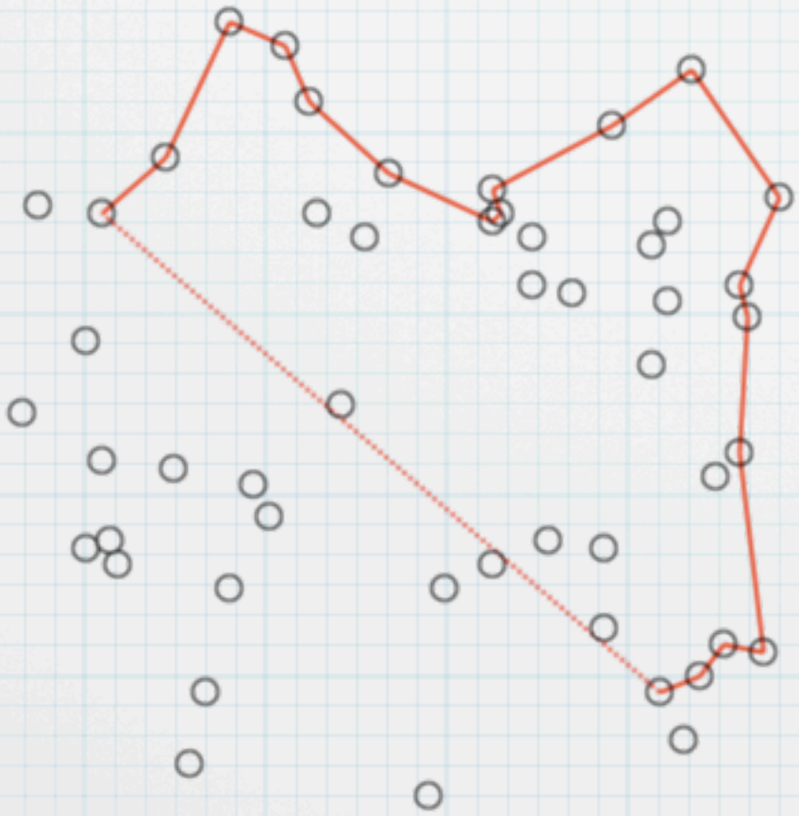
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



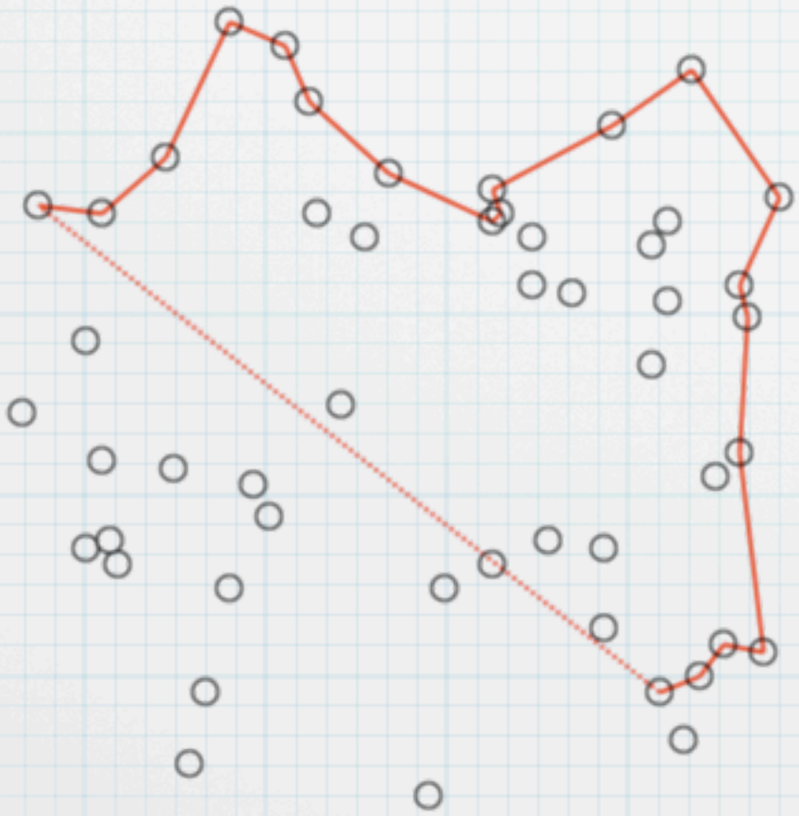
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



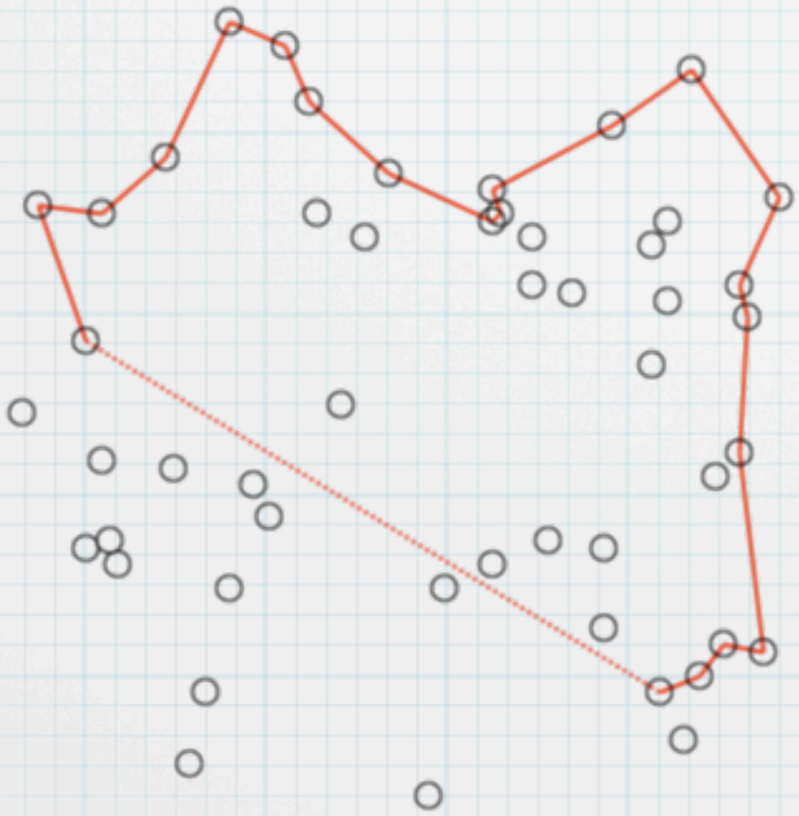
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



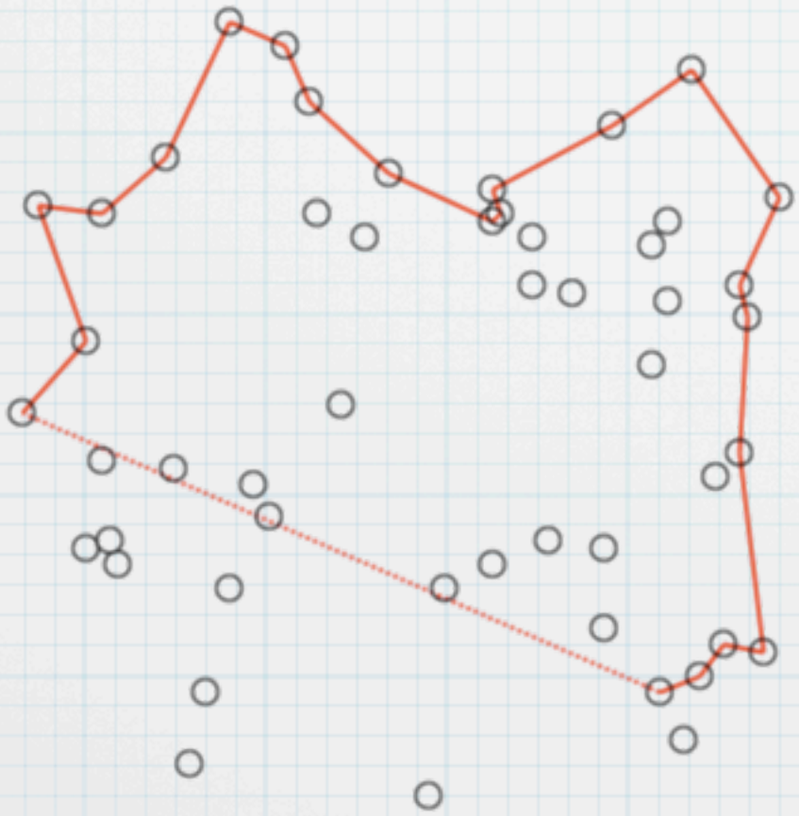
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



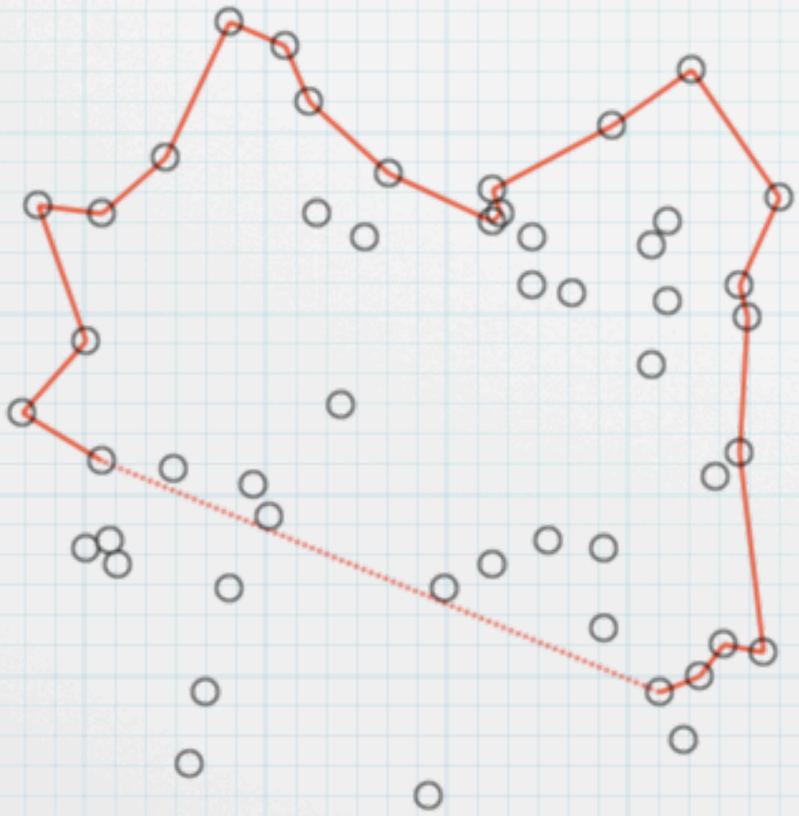
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



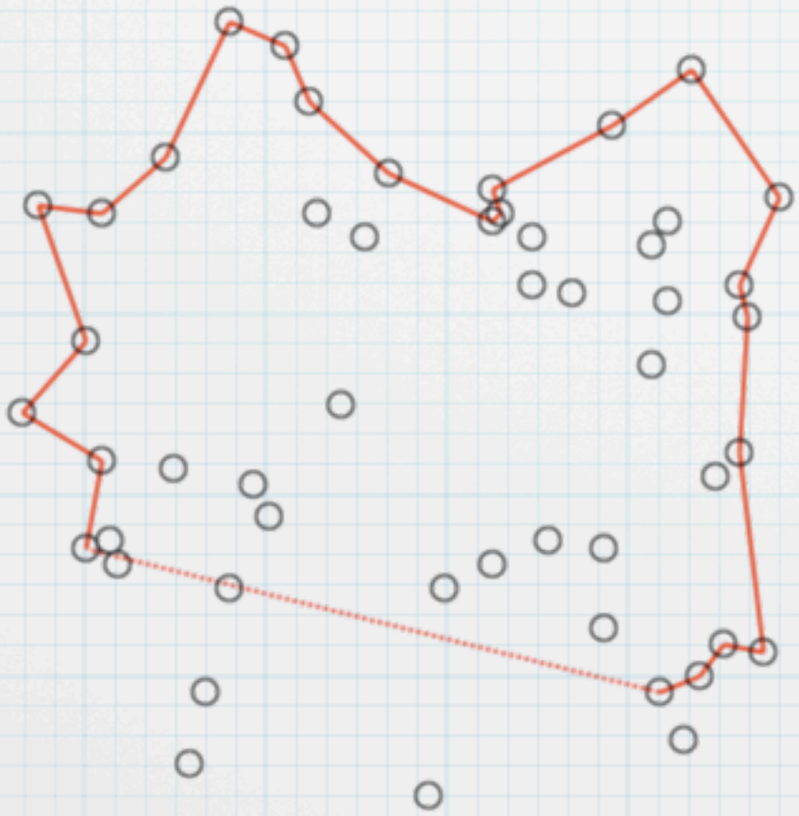
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



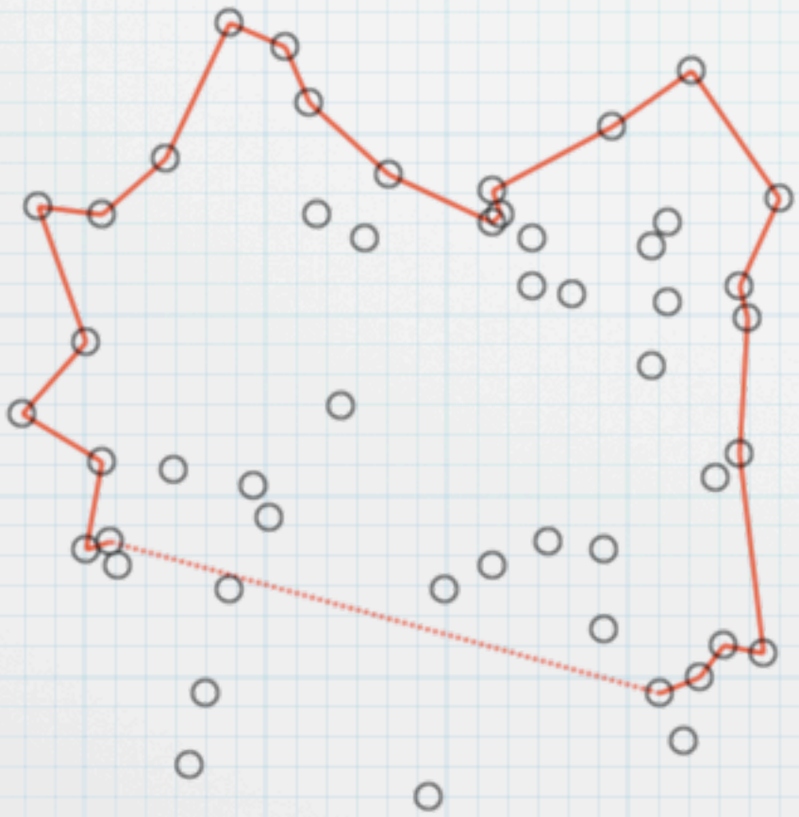
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



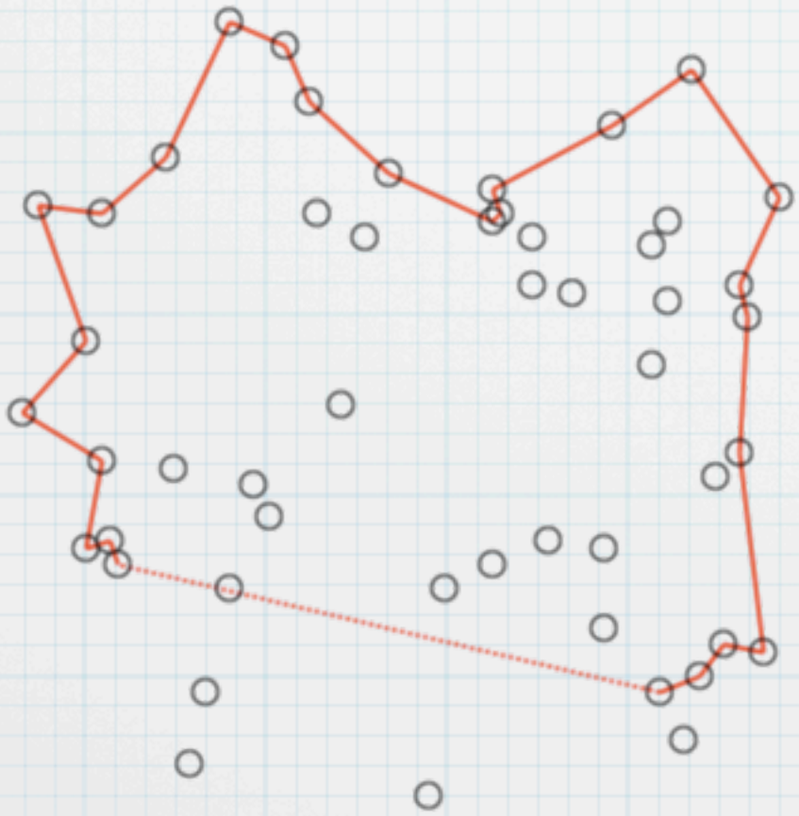
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



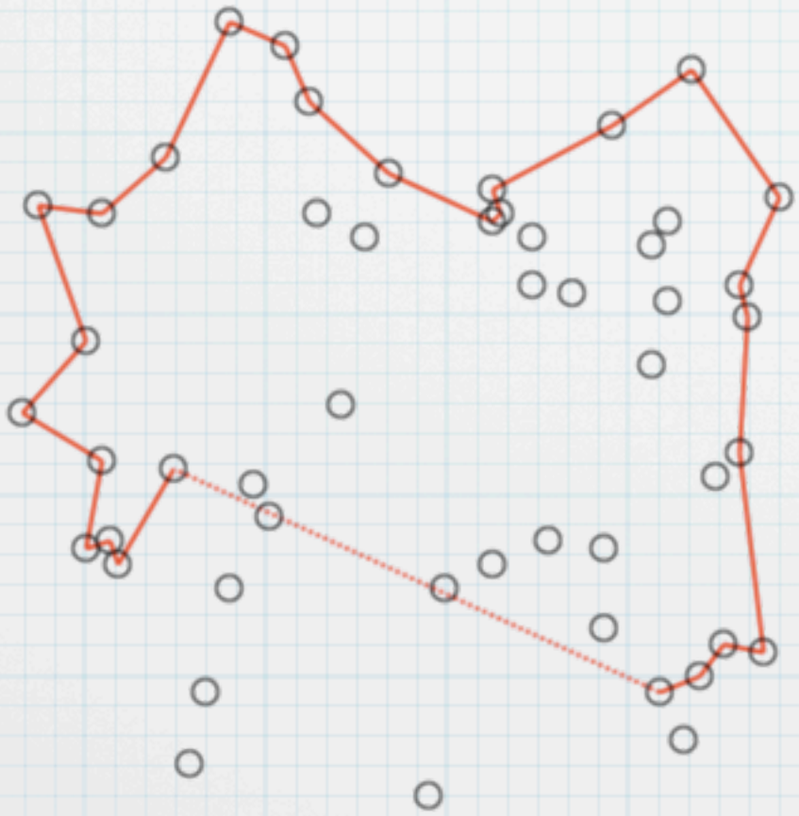
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



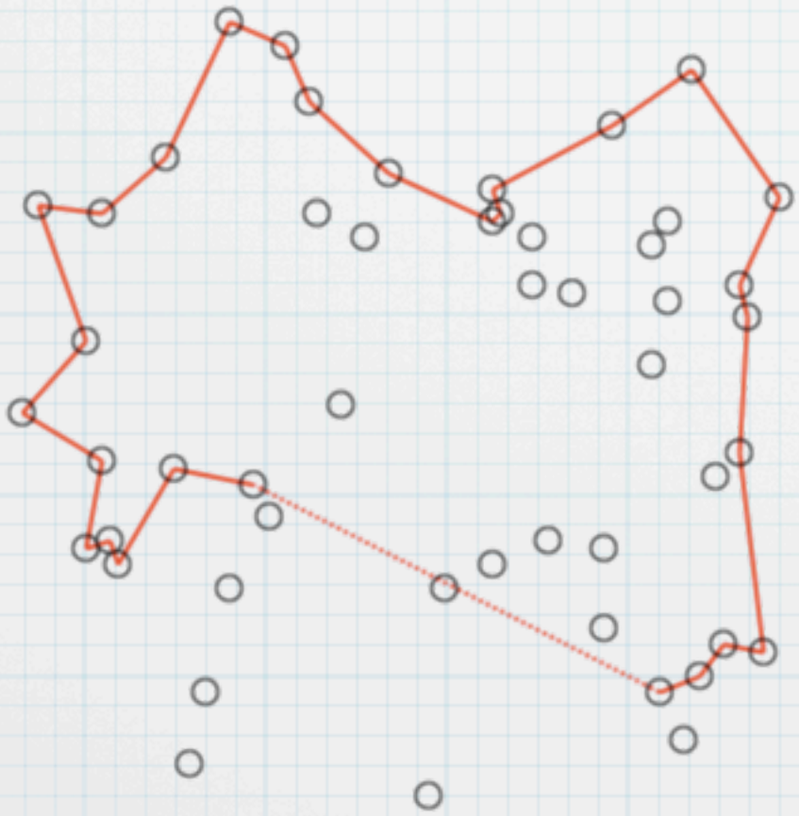
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



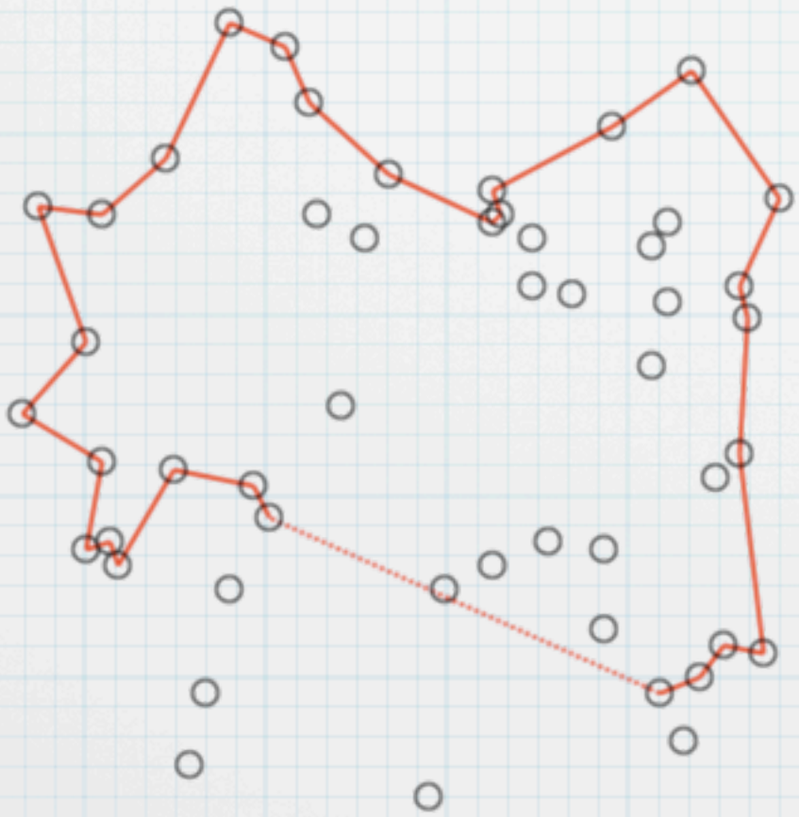
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



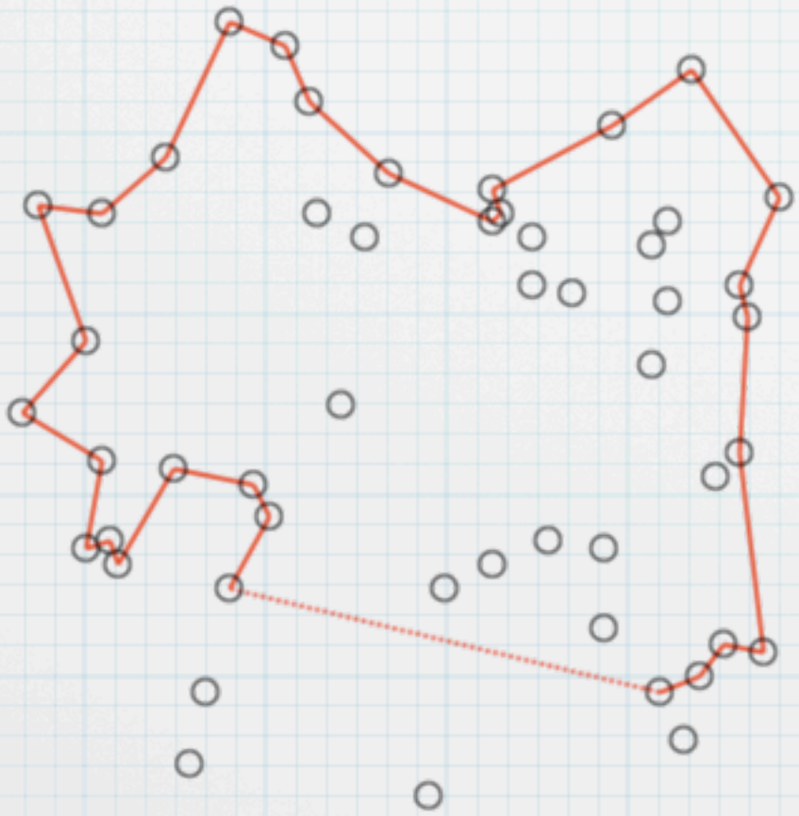
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



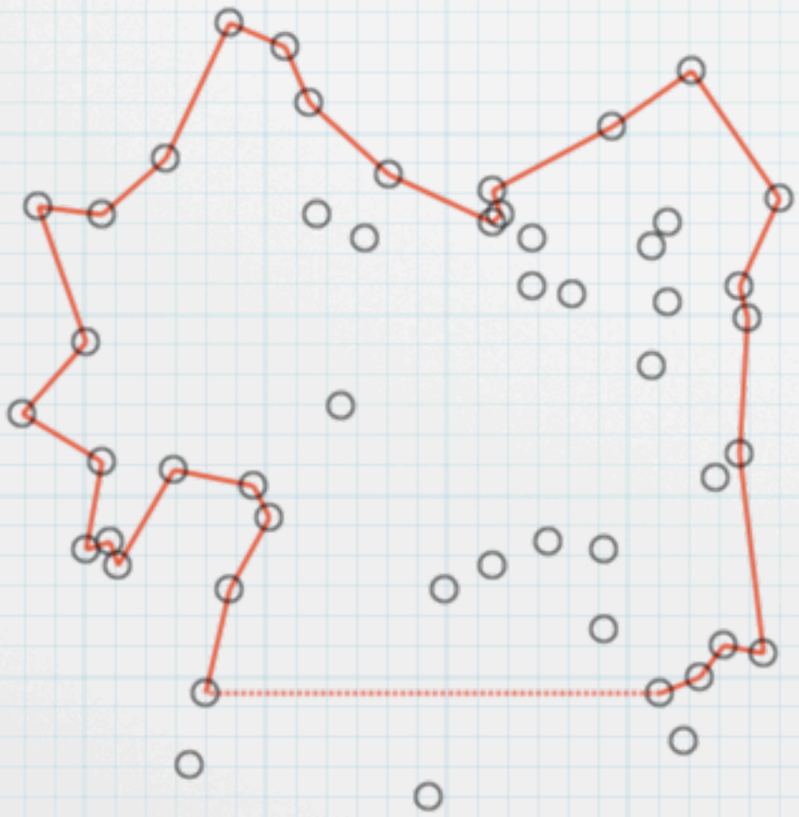
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



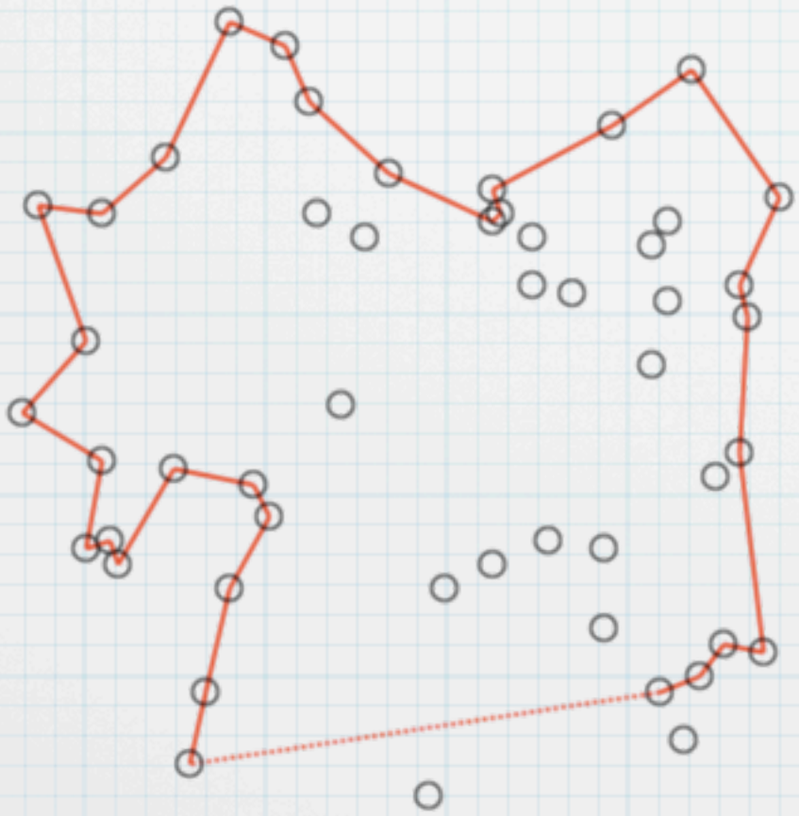
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



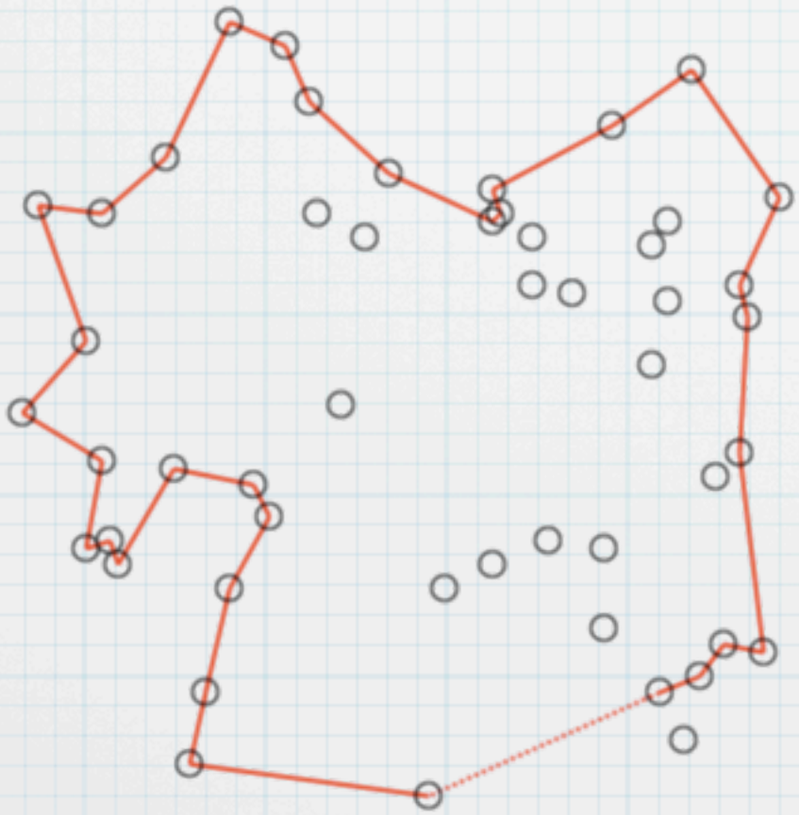
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



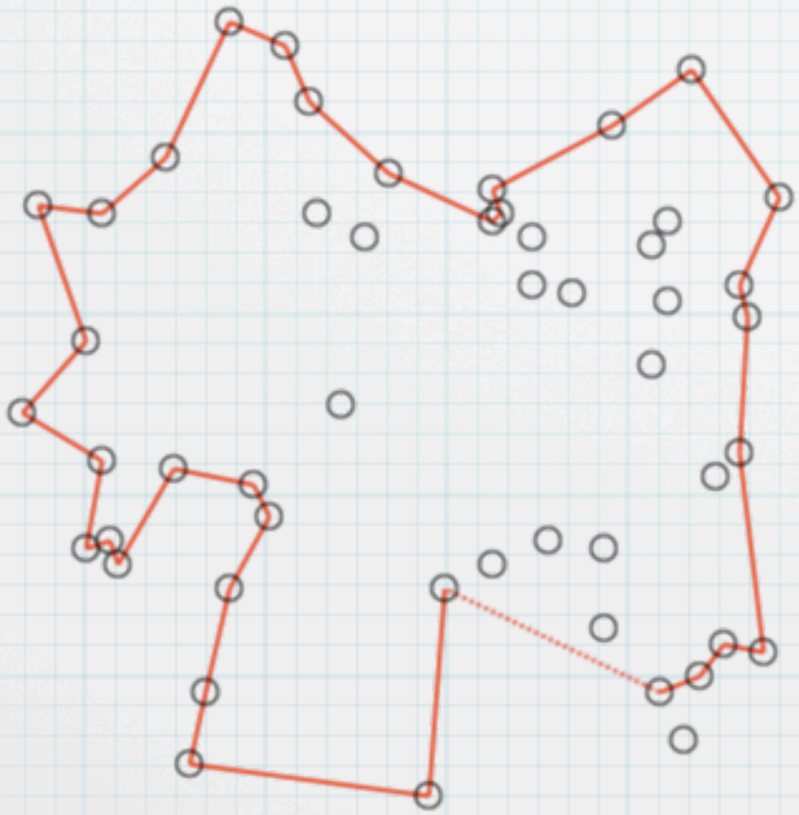
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



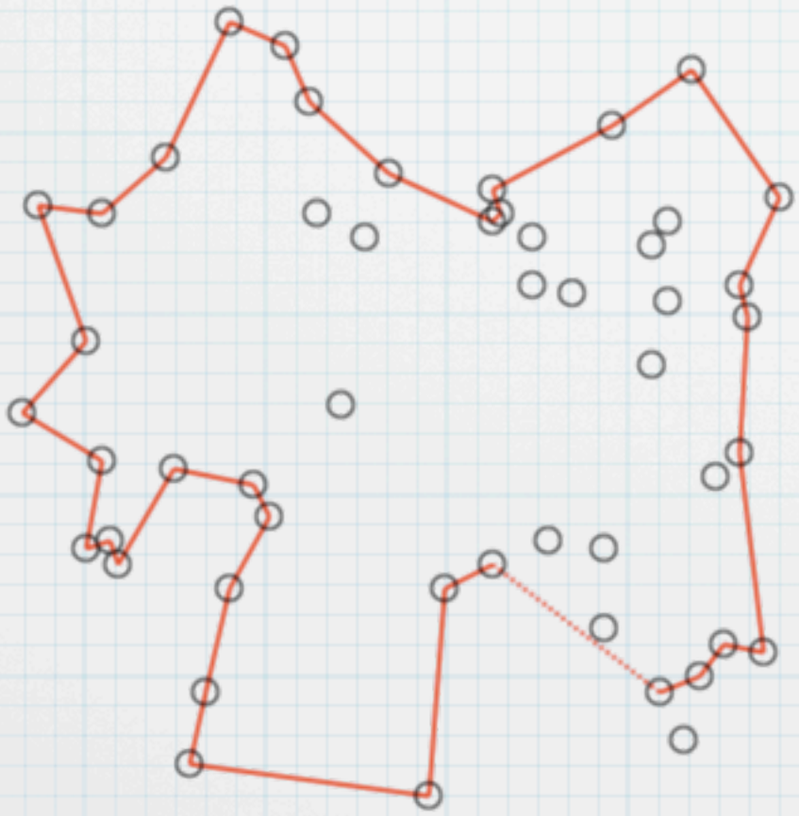
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



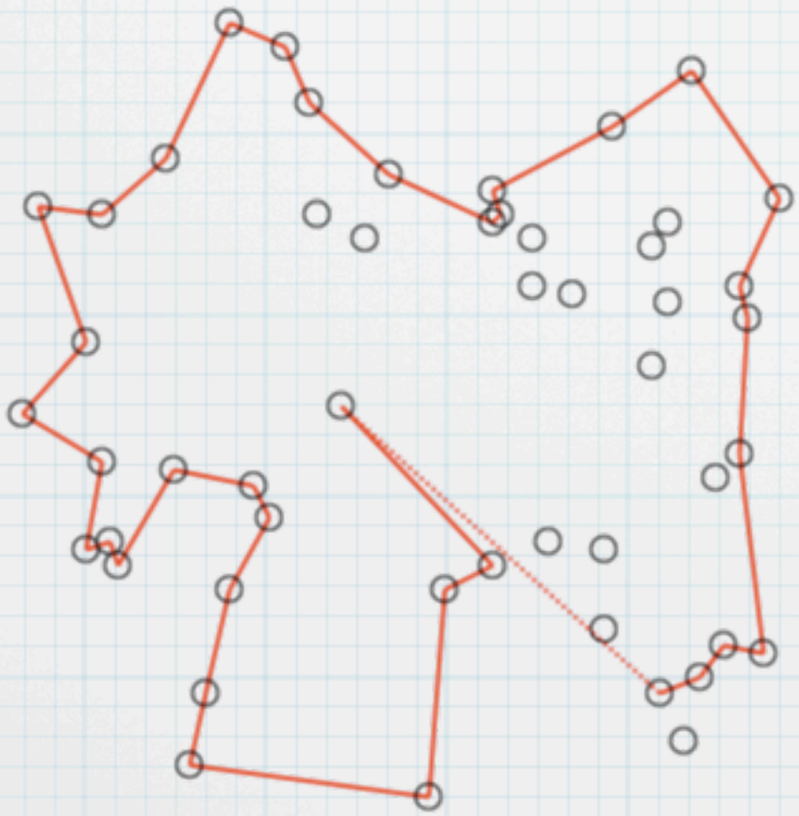
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



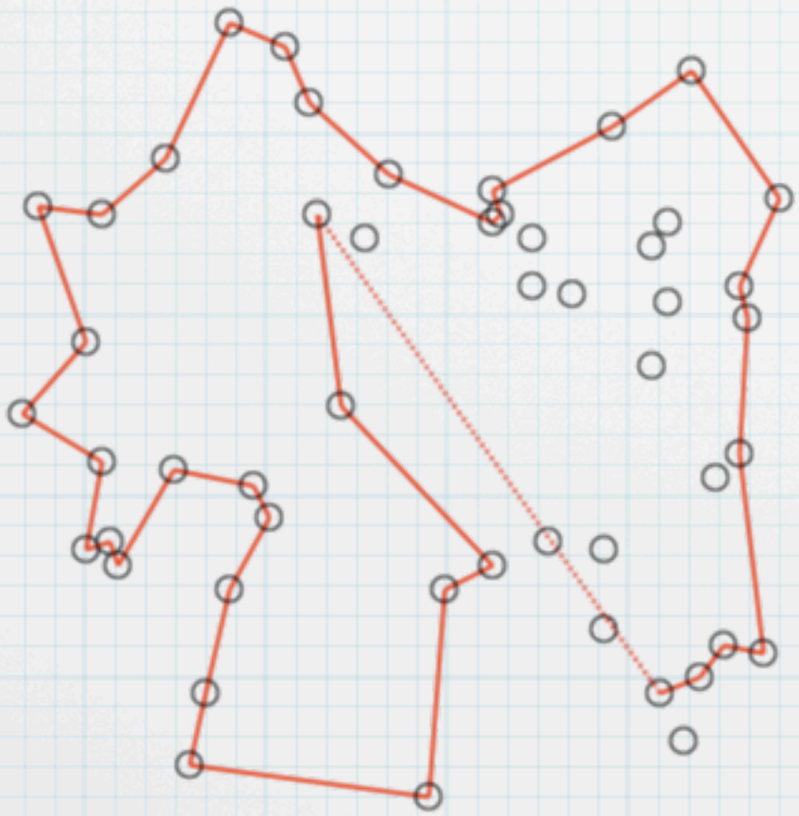
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



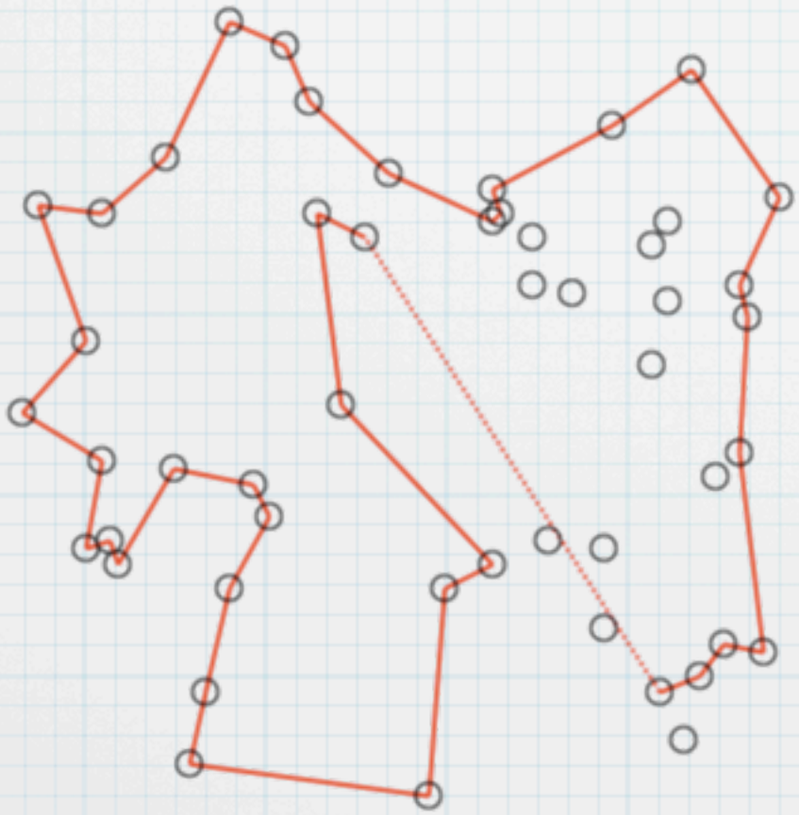
Parametrisches Greedy-Verfahren

- Modifikation des nächster–Nachbar–Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



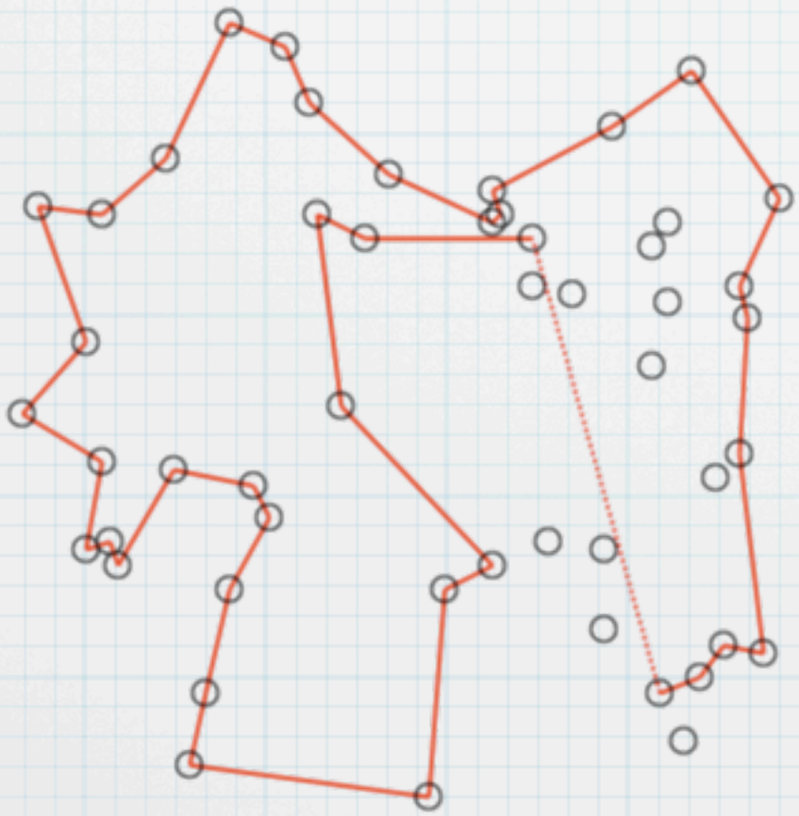
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



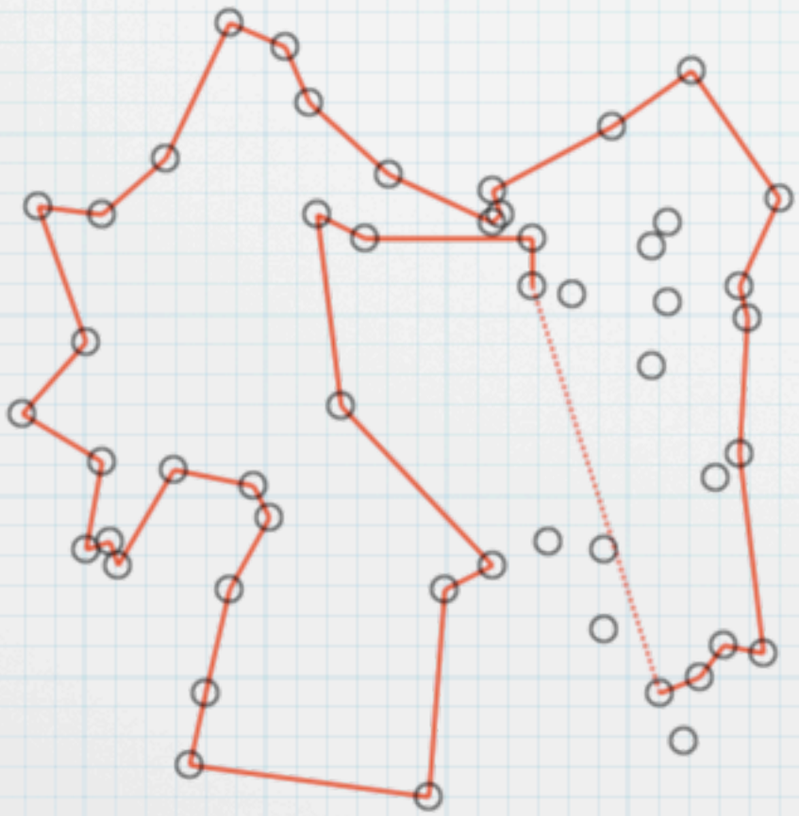
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



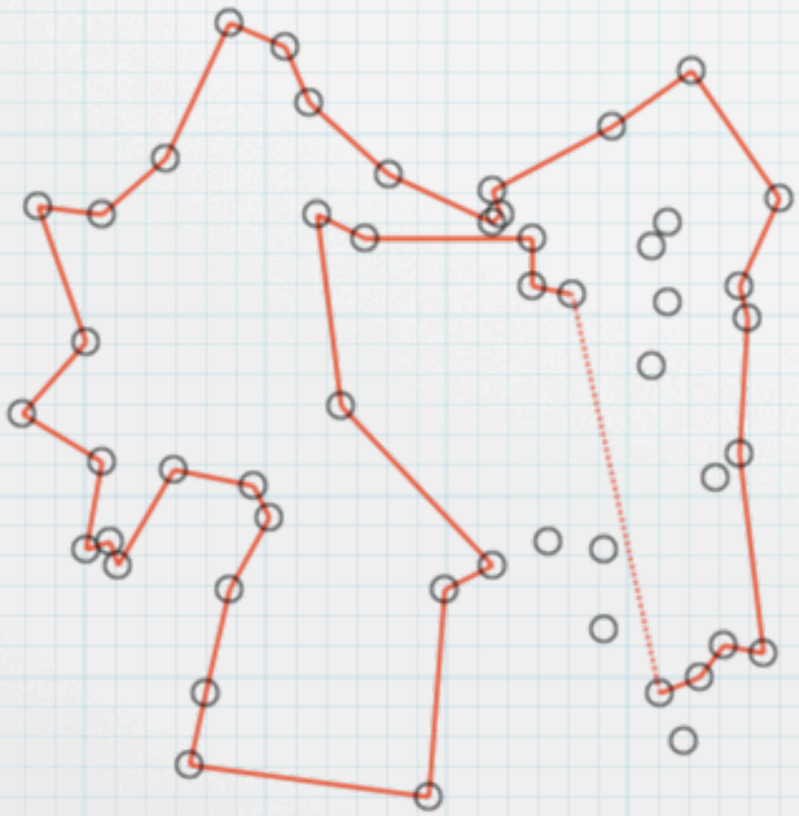
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



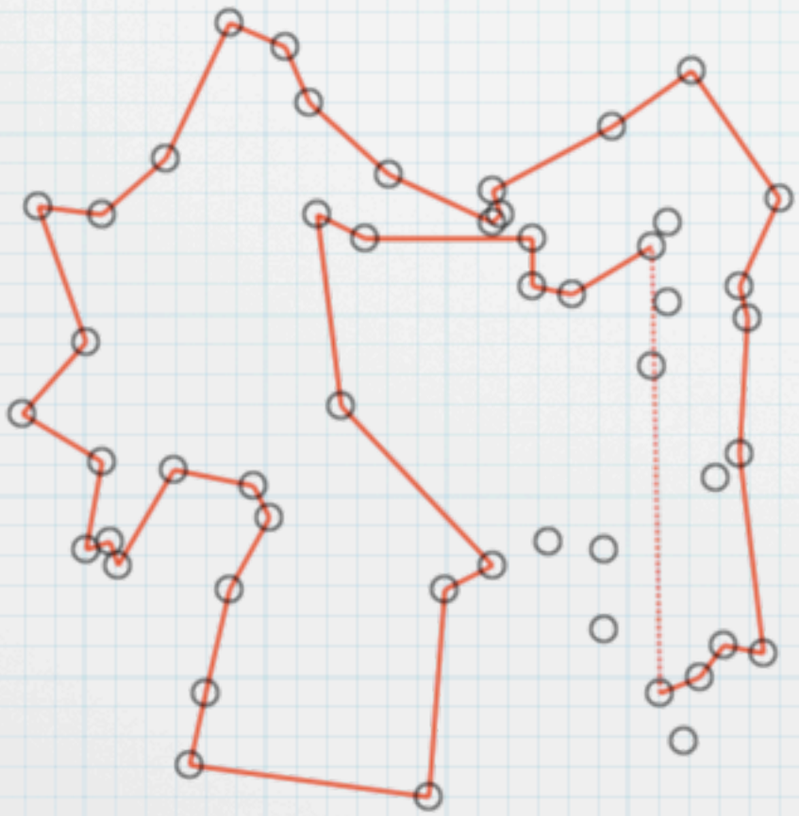
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



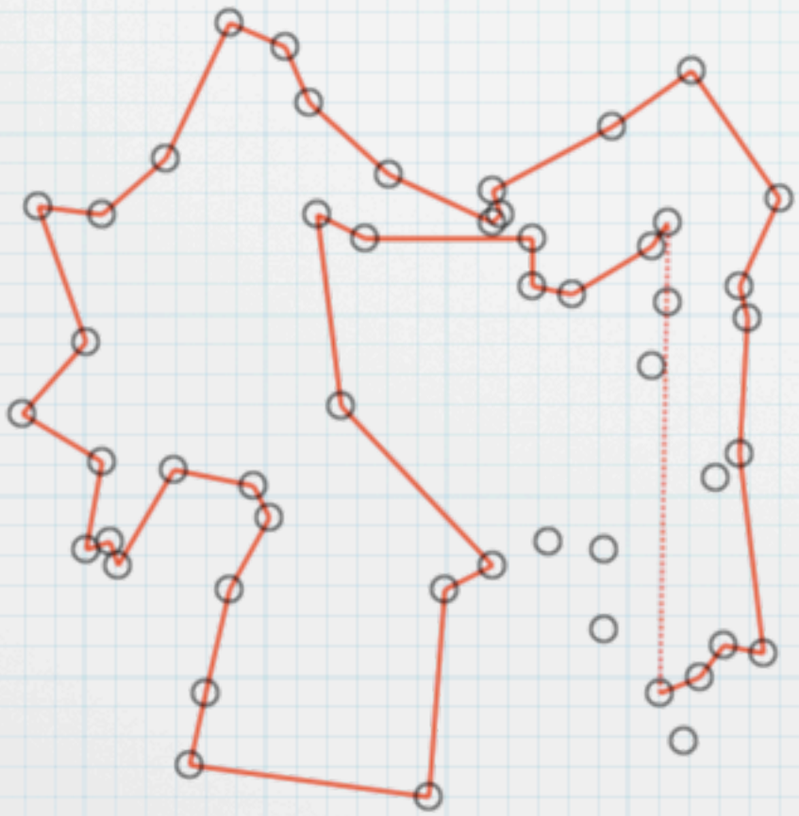
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



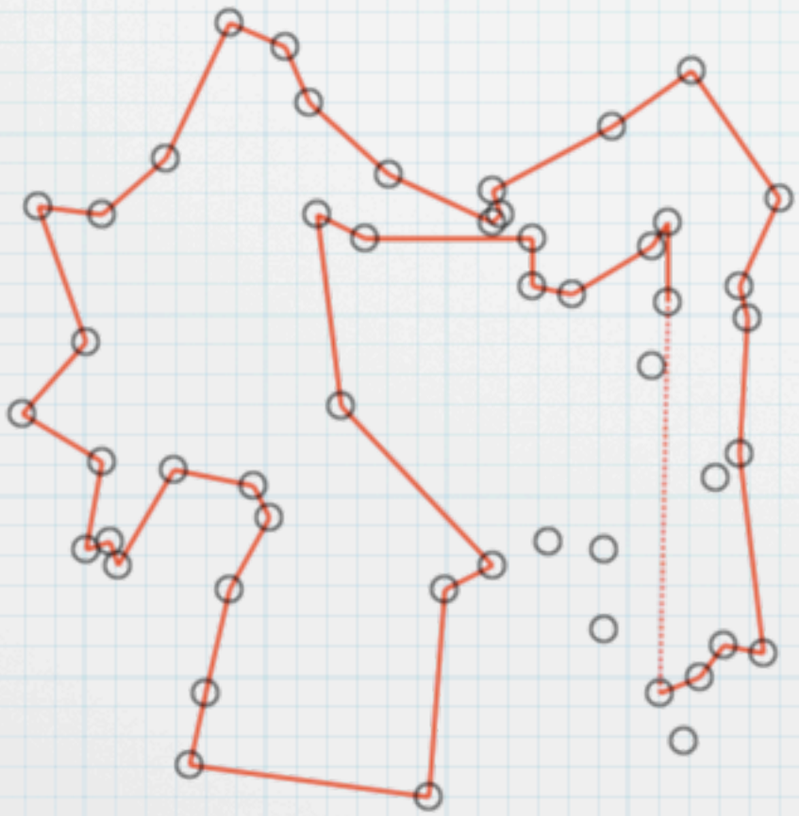
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



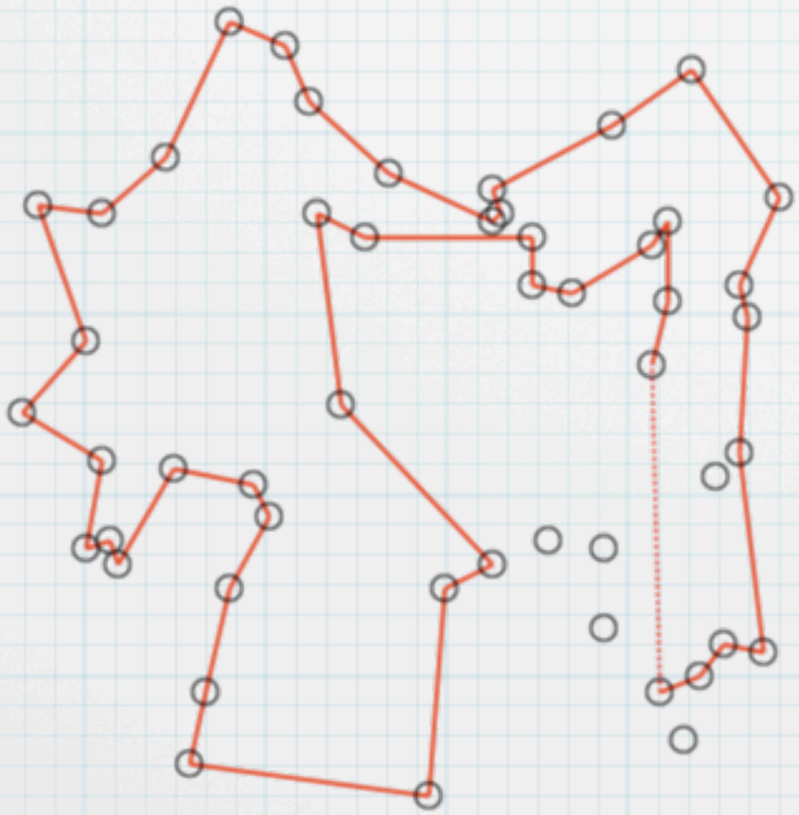
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



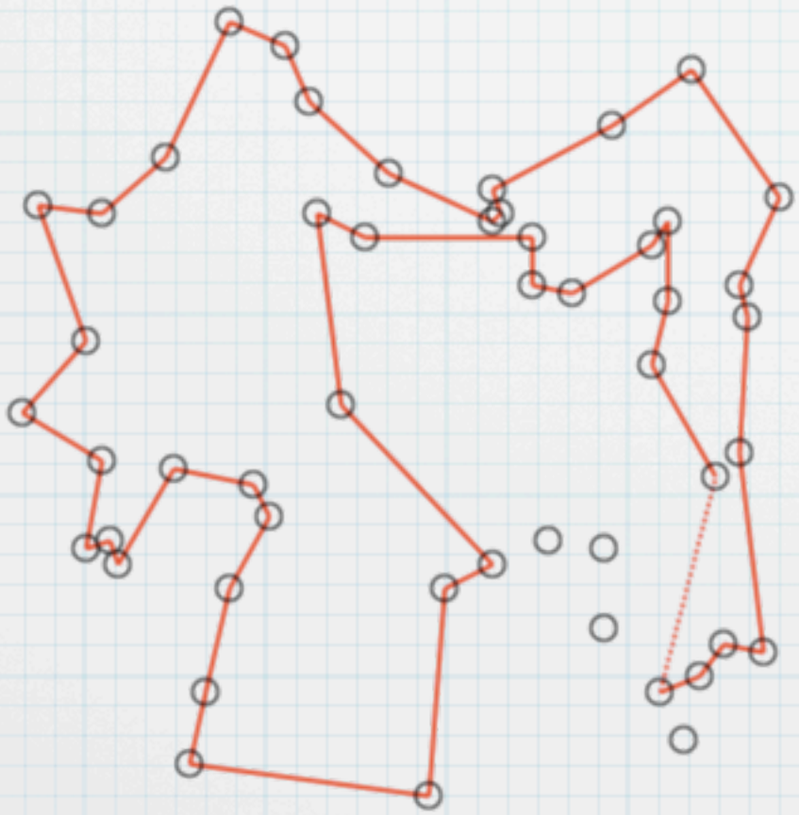
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



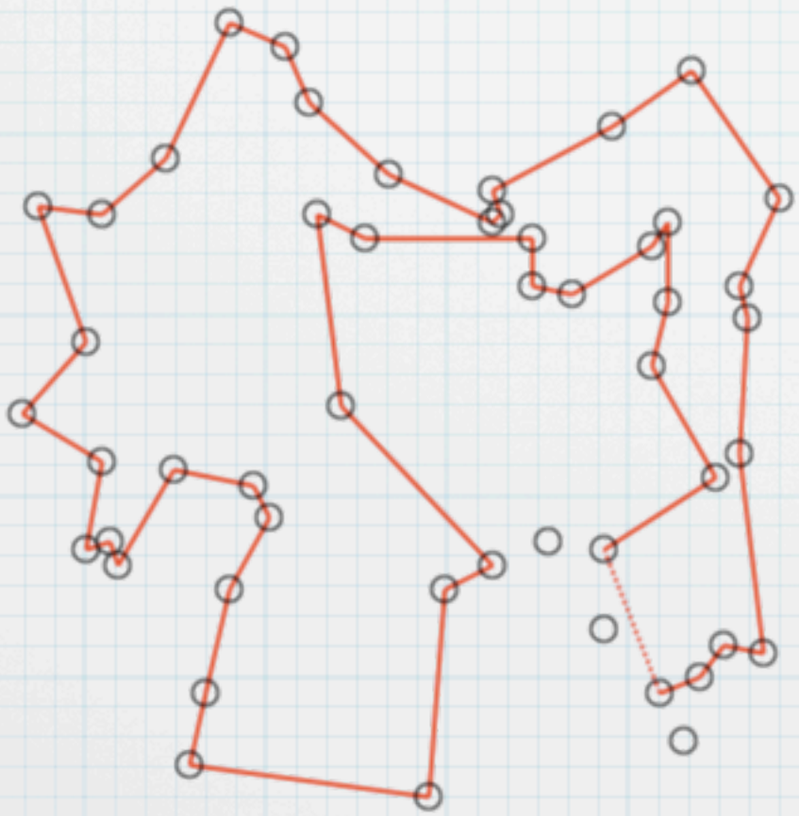
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



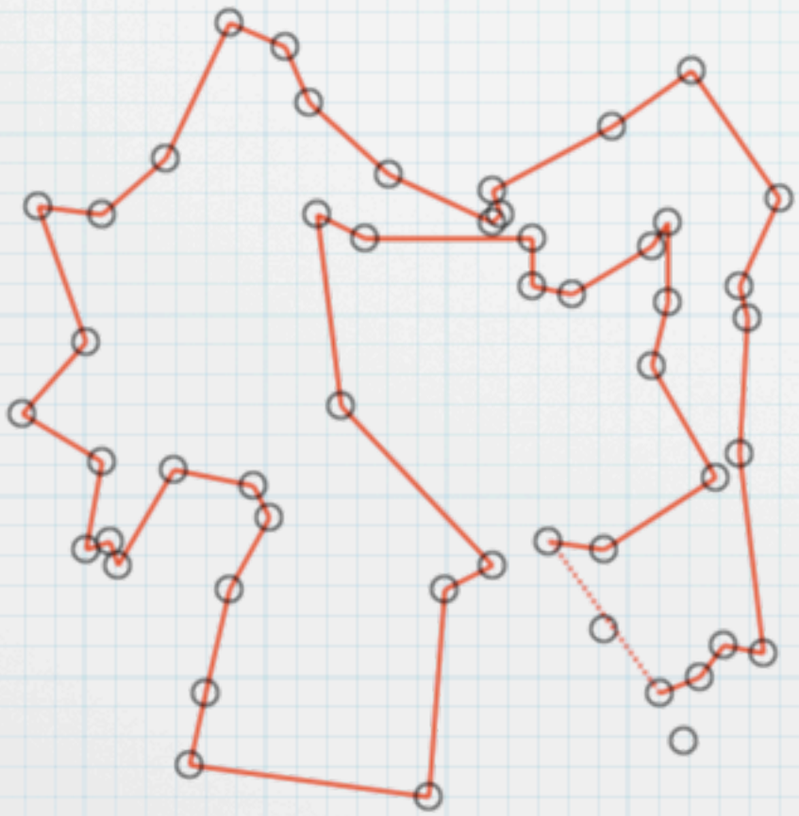
Parametrisches Greedy-Verfahren

- Modifikation des nächster–Nachbar–Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



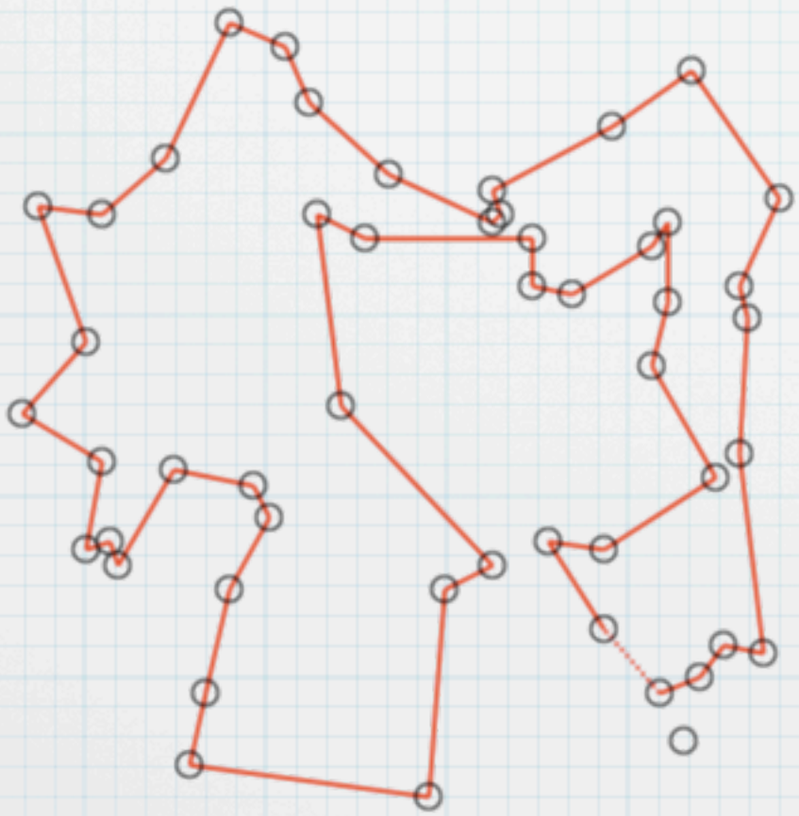
Parametrisches Greedy-Verfahren

- Modifikation des nächster–Nachbar–Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



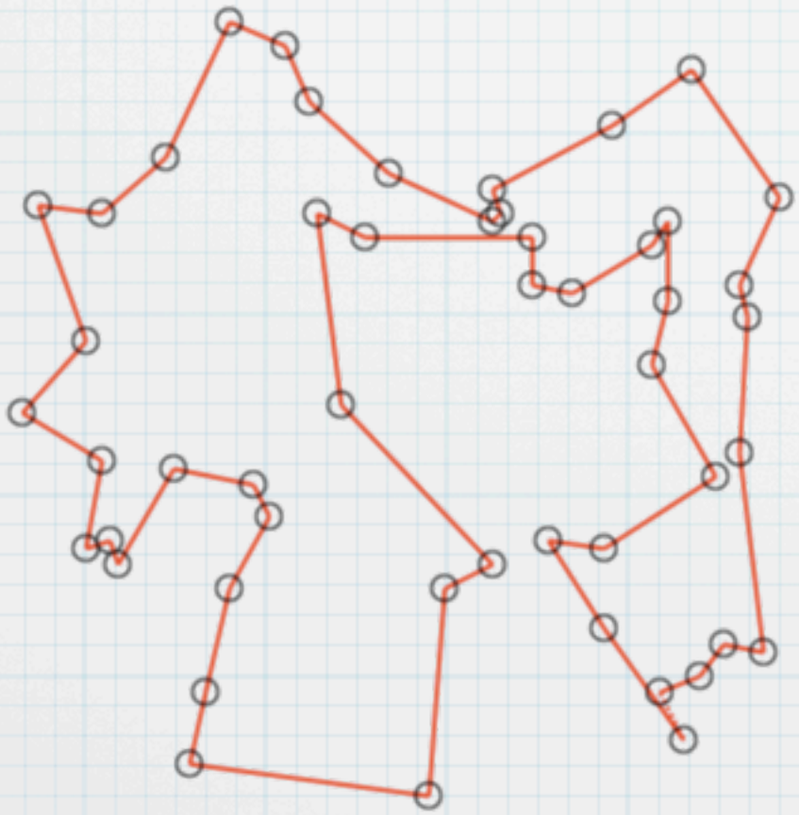
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



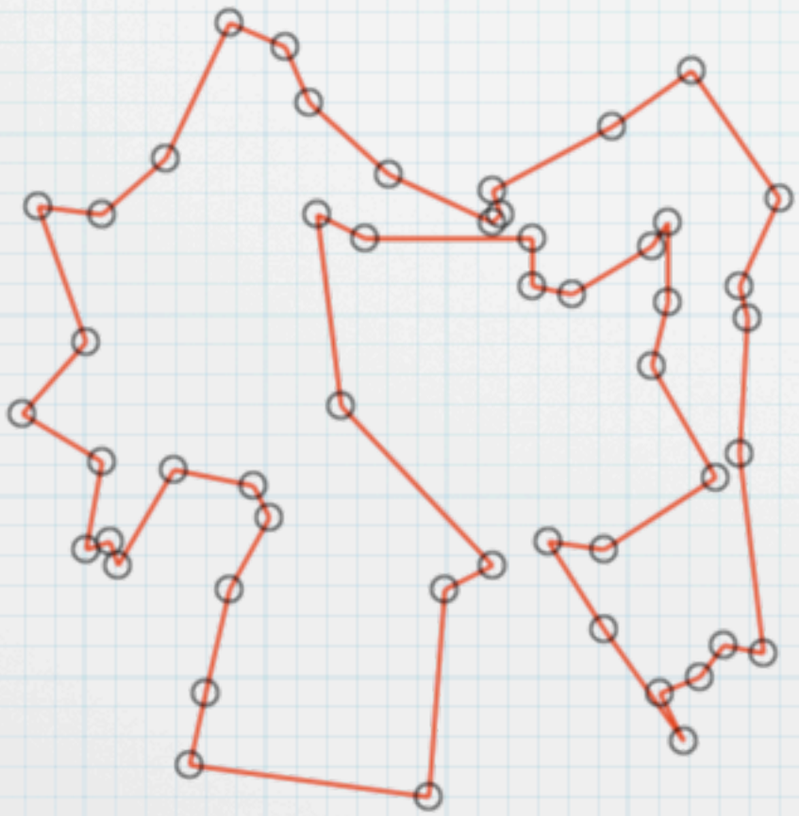
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



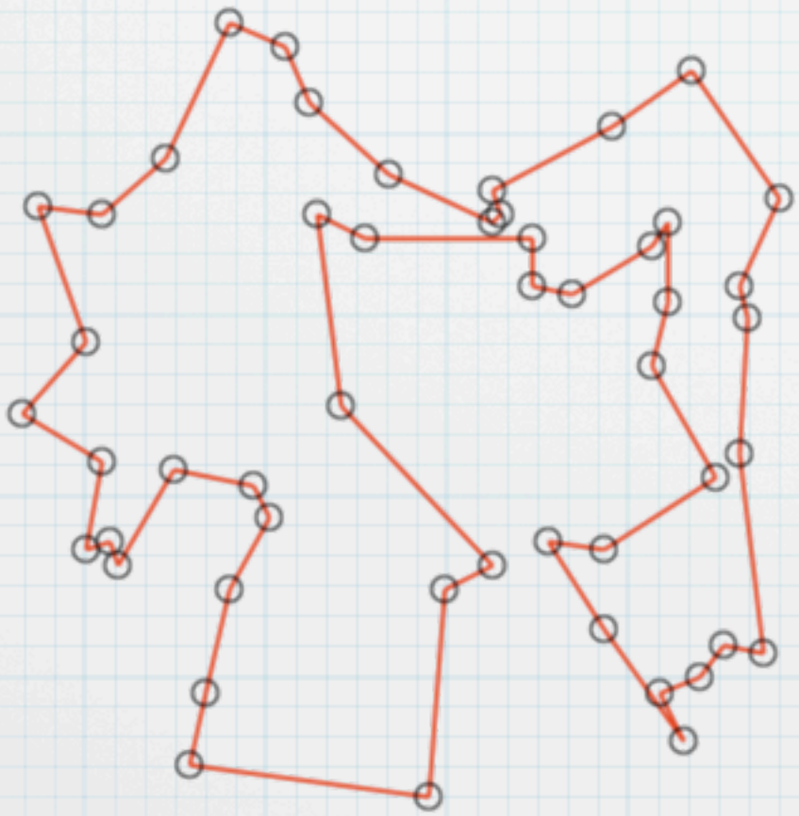
Parametrisches Greedy-Verfahren

- Modifikation des nächster-Nachbar-Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4$



Parametrisches Greedy-Verfahren

- Modifikation des nächster–Nachbar–Verfahrens
- Wähle ein $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$
- Starte an einem beliebigen Knoten (folgend mit „1“ bezeichnet).
- Füge einen Knoten k an den Endknoten i des Pfades P an, der
  - noch nicht besucht wurde,
  - und den kleinsten „Score“ $s_{i,j}$ zum Vorgängerknoten hat:
    $k = \arg\min\{s_{i,j} : j \notin P\}$, wobei $s_{i,j}(\lambda) := \lambda_1 \cdot c_{i,j} + \lambda_2 \cdot c_{j,1}$.
- Wiederhole, bis alle Knoten eingefügt sind.
- Verbinde den letzten Knoten mit dem Startknoten.
- Beispiel: $\lambda_1 := 0,6, \lambda_2 := -0,4 \Rightarrow z(\lambda) = 592$



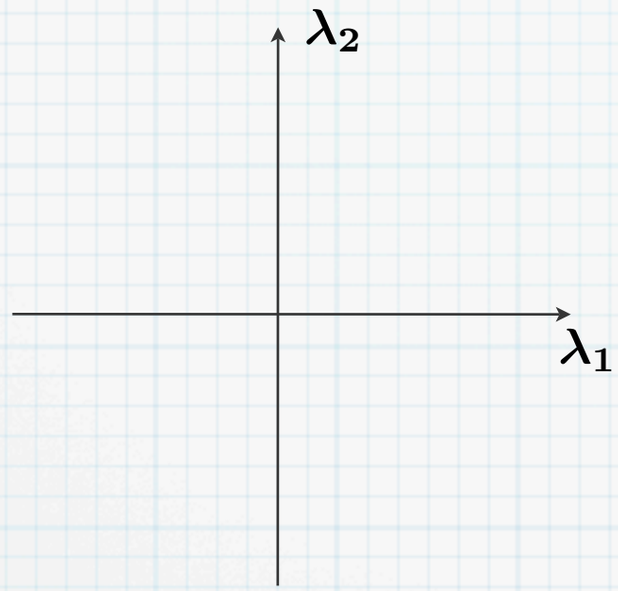
Abhängigkeit der Lösung von den Parametern

Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

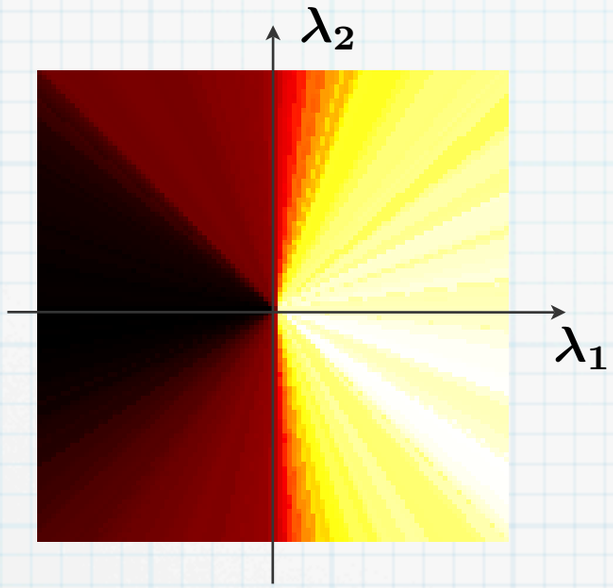
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):



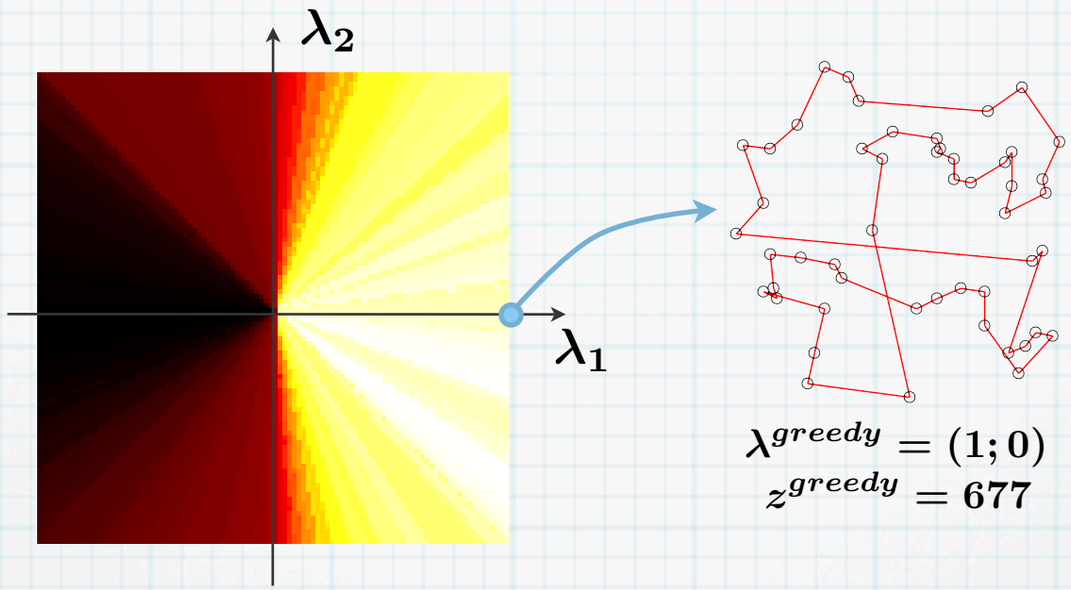
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):



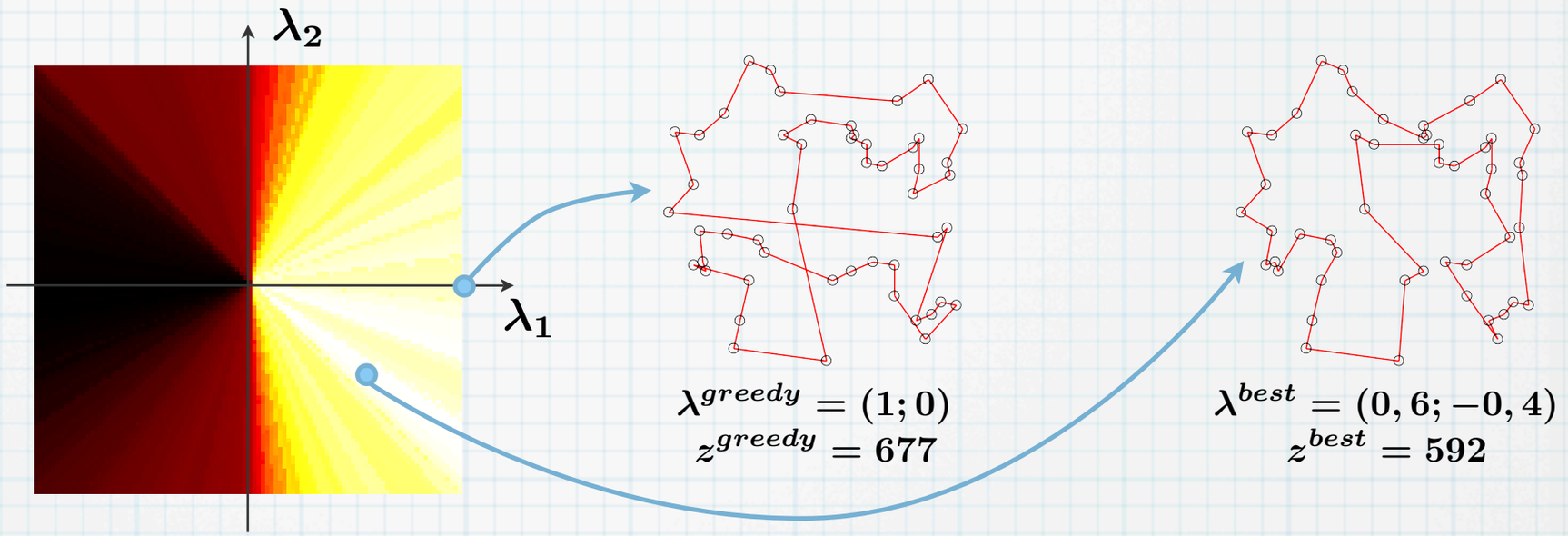
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):



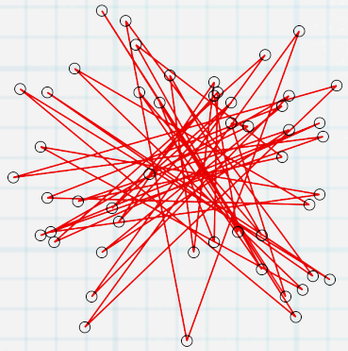
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

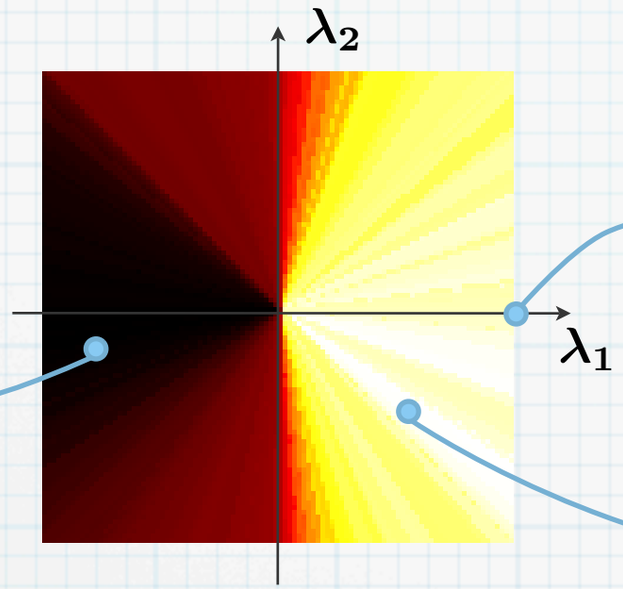


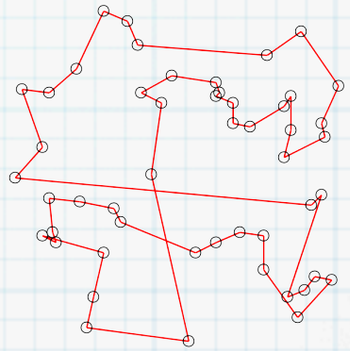
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

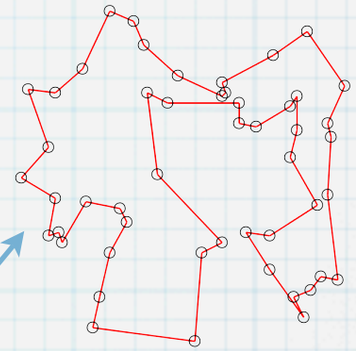


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





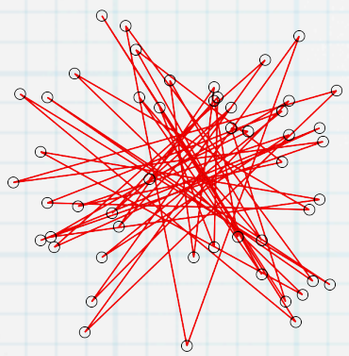
$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$



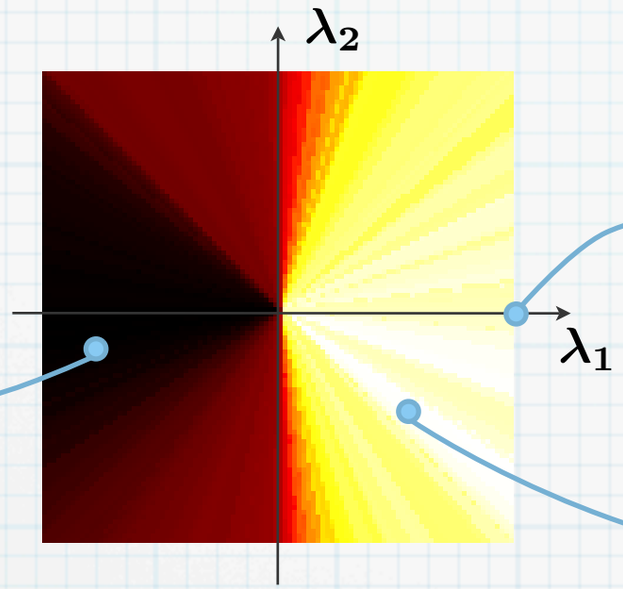
$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

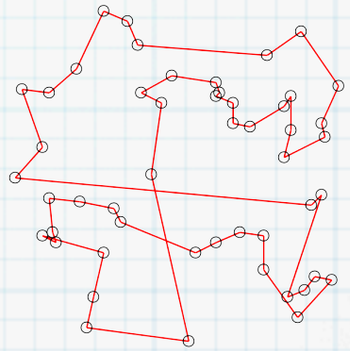
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

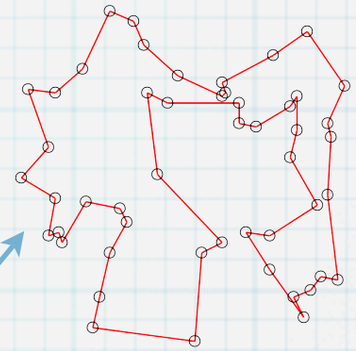


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

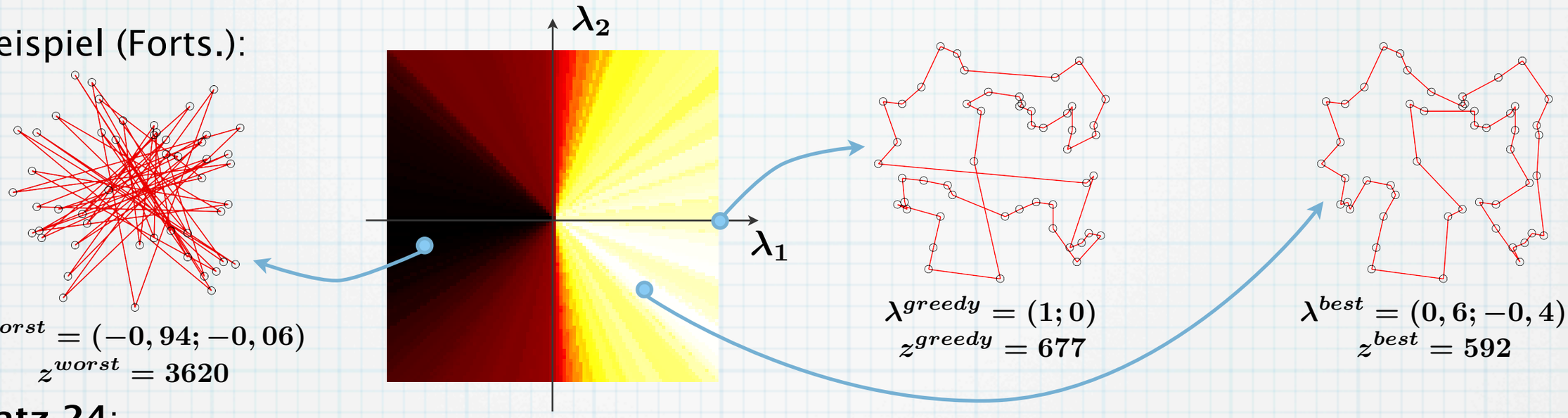


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.

Abhängigkeit der Lösung von den Parametern

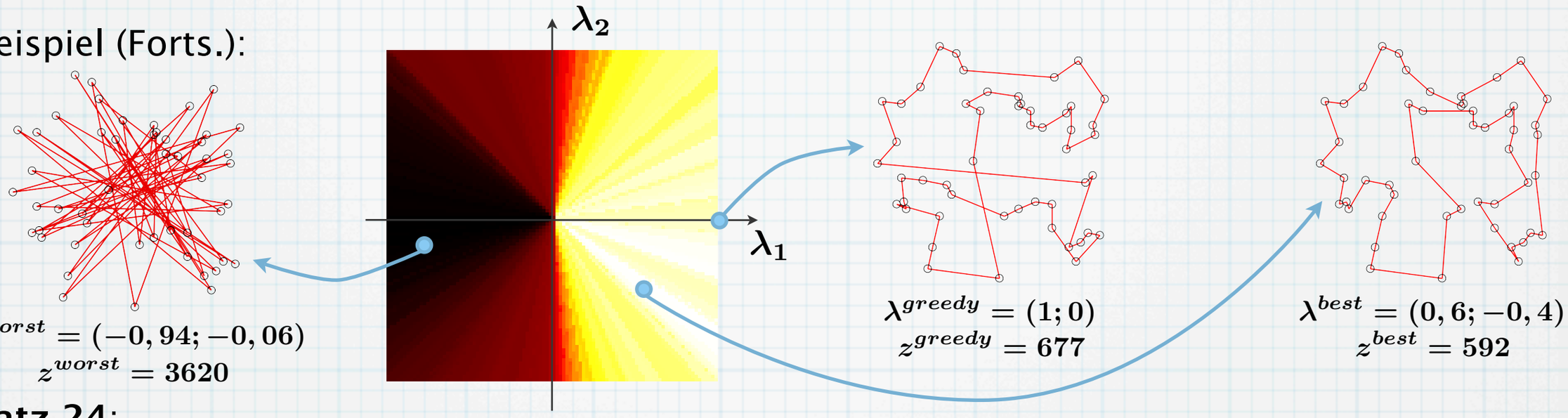
* Beispiel (Forts.):



* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:

Abhängigkeit der Lösung von den Parametern

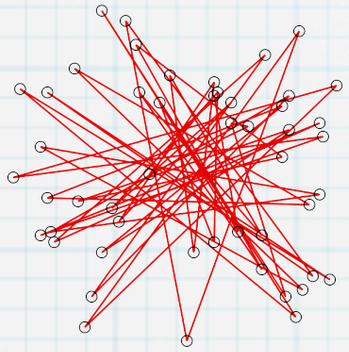
* Beispiel (Forts.):



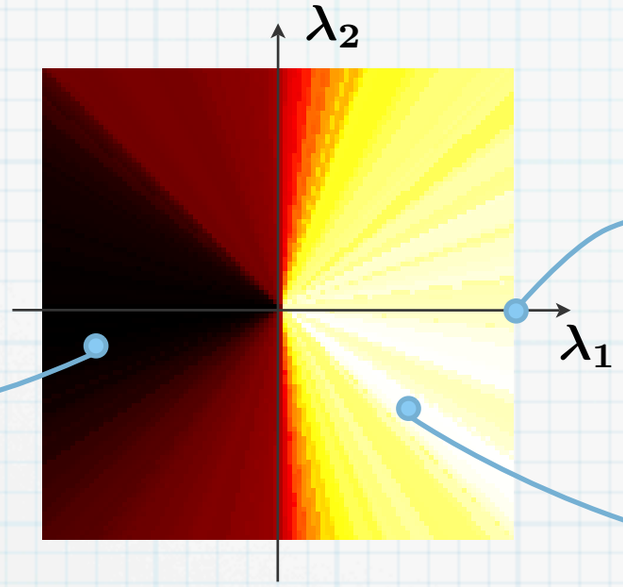
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.

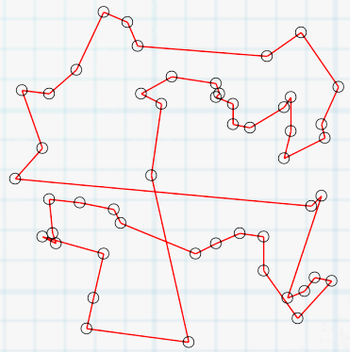
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

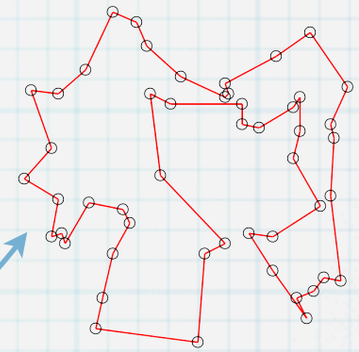


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

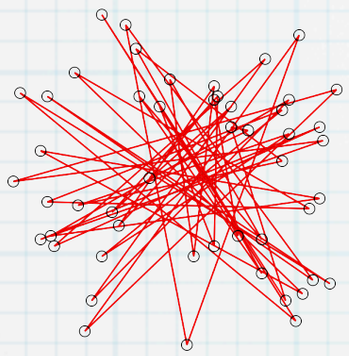


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

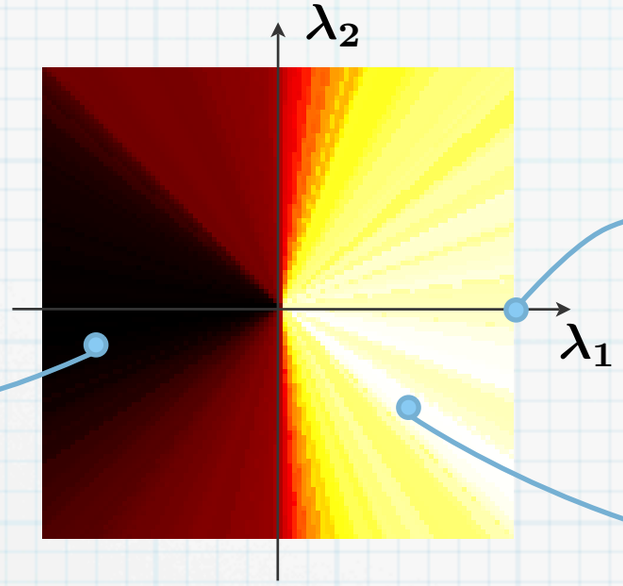
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.

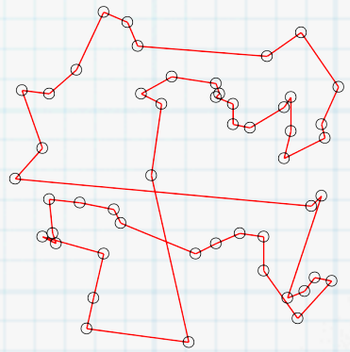
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

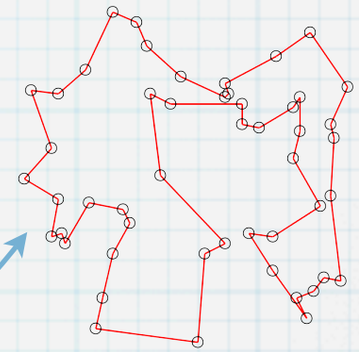


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

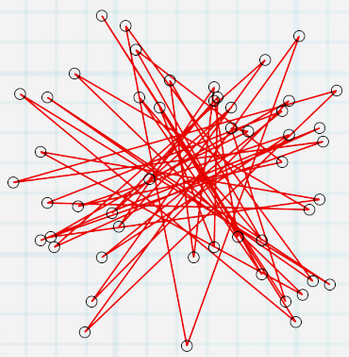


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

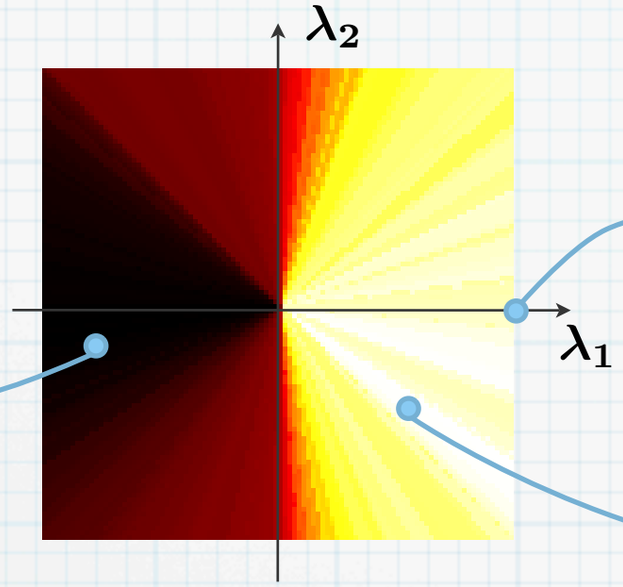
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.

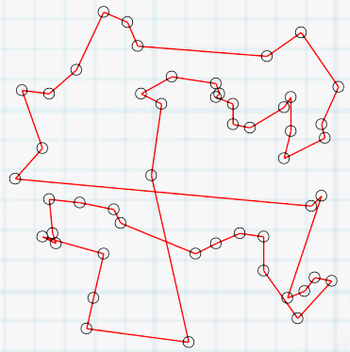
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

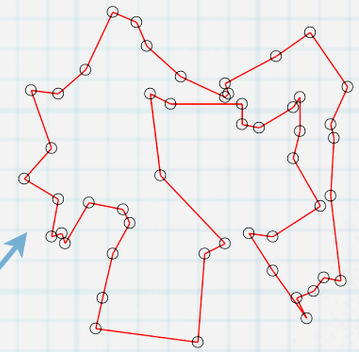


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

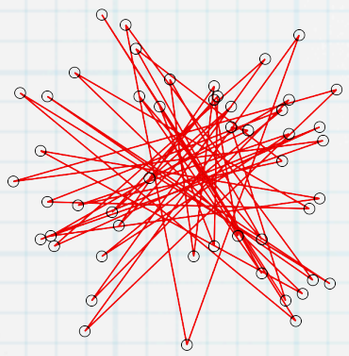


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

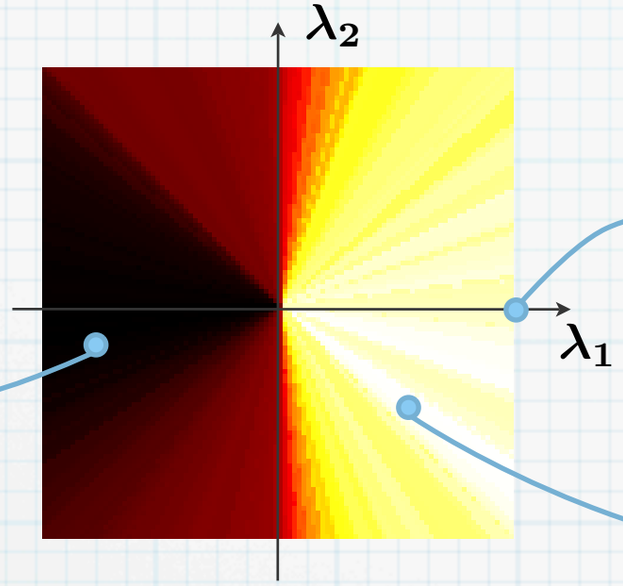
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.
  Somit werden die gleichen Knoten in jedem Schritt gewählt.

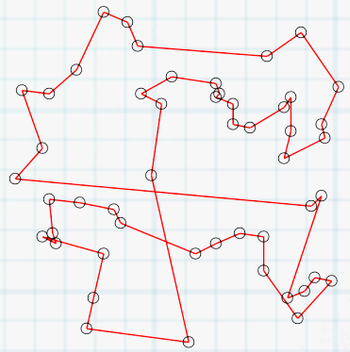
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

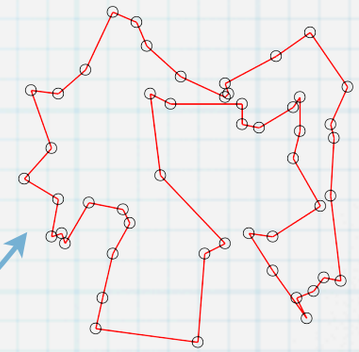


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

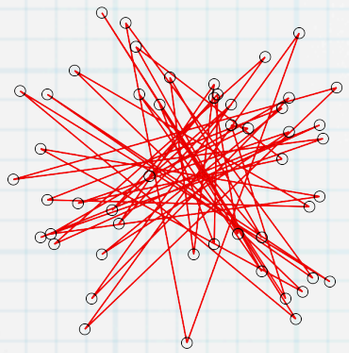


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

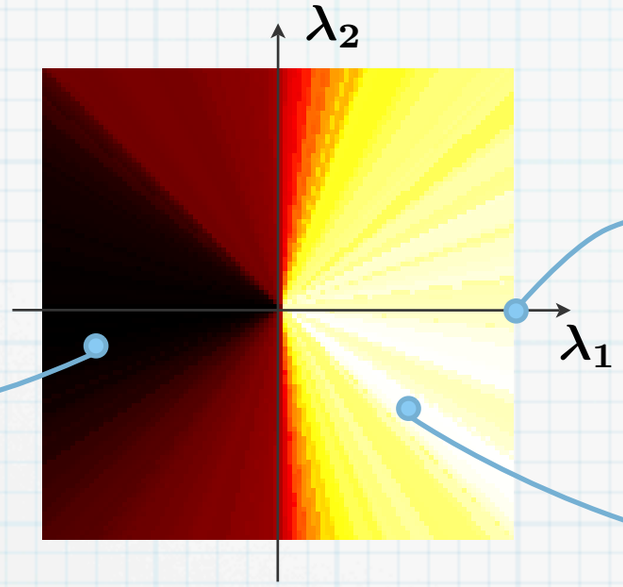
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.
  Somit werden die gleichen Knoten in jedem Schritt gewählt.
  Am Ende sind dann beide so konstruierten Lösungen identisch.

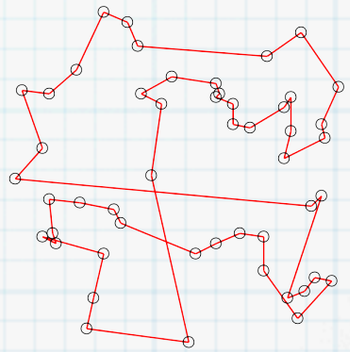
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

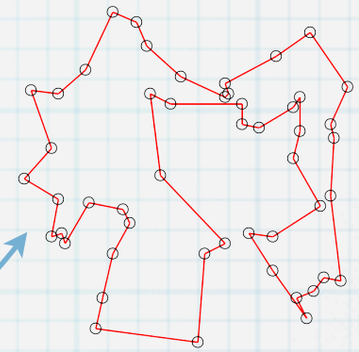


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

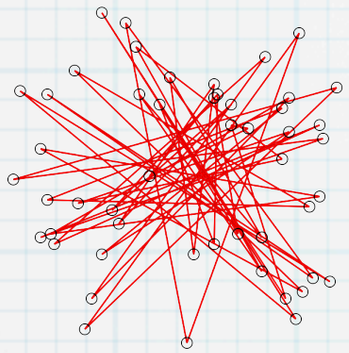


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

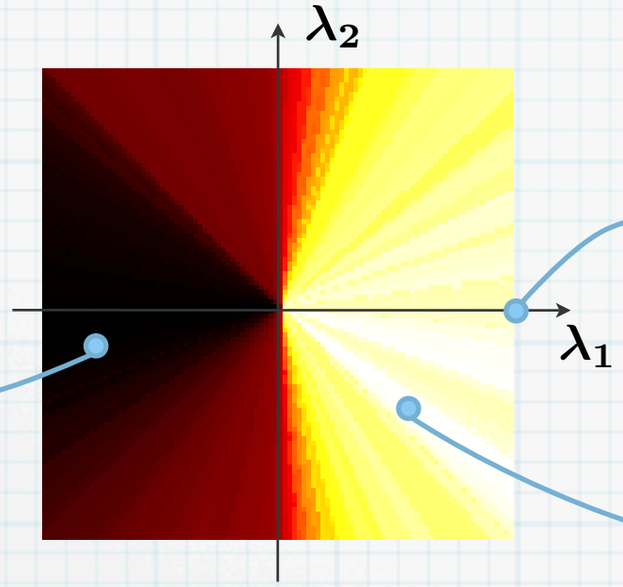
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.
  Somit werden die gleichen Knoten in jedem Schritt gewählt.
  Am Ende sind dann beide so konstruierten Lösungen identisch.
* **Korollar 25**:
$z^{pgreedy} := \min\{z(\lambda) : \lambda \in \mathbb{R}^p\} = \min\{z(\lambda) : \lambda \in \mathbb{R}^p, \|\lambda\| = 1\}$.

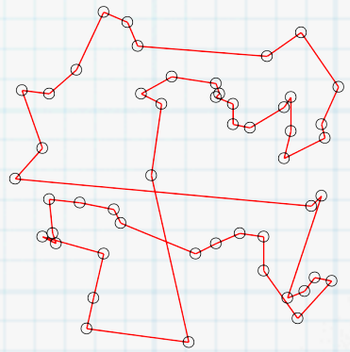
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

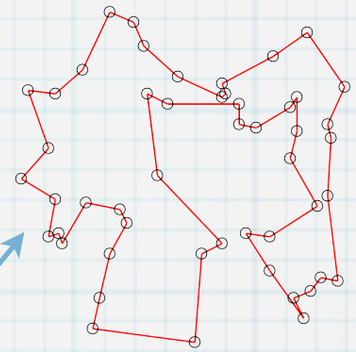


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

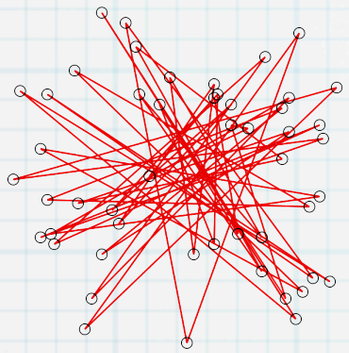


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

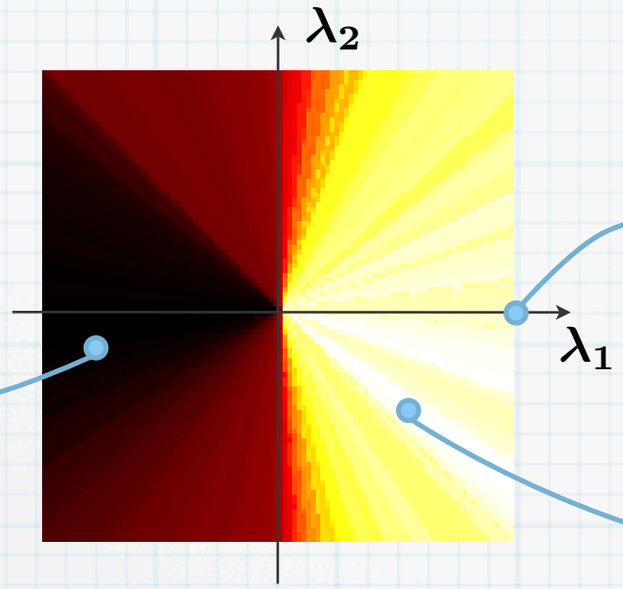
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.
  Somit werden die gleichen Knoten in jedem Schritt gewählt.
  Am Ende sind dann beide so konstruierten Lösungen identisch.
* **Korollar 25**:
$z^{pgreedy} := \min\{z(\lambda) : \lambda \in \mathbb{R}^p\} = \min\{z(\lambda) : \lambda \in \mathbb{R}^p, \|\lambda\| = 1\}$.
* Interpretation:

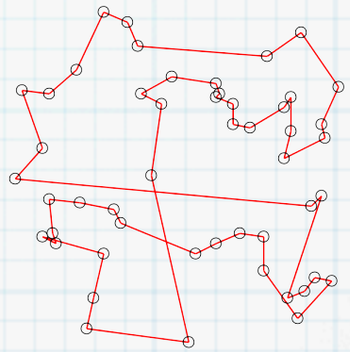
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

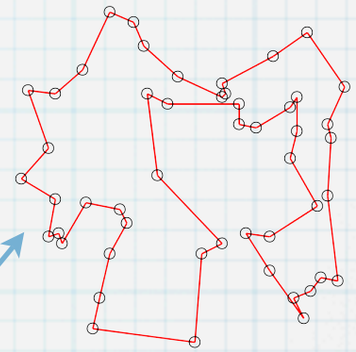


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

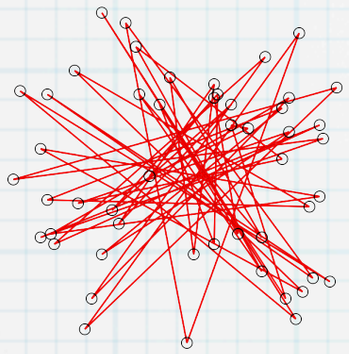


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

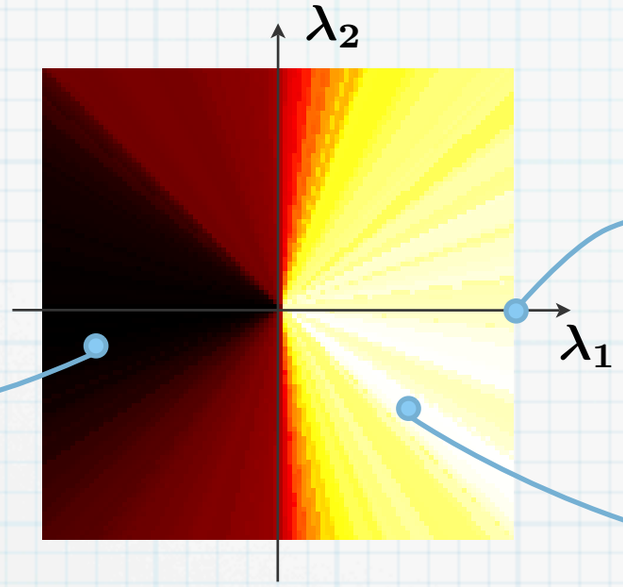
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.
  Somit werden die gleichen Knoten in jedem Schritt gewählt.
  Am Ende sind dann beide so konstruierten Lösungen identisch.
* **Korollar 25**:
$z^{pgreedy} := \min\{z(\lambda) : \lambda \in \mathbb{R}^p\} = \min\{z(\lambda) : \lambda \in \mathbb{R}^p, \|\lambda\| = 1\}$.
* Interpretation:
  * Die Dimension des Suchraums kann um 1 reduziert werden.

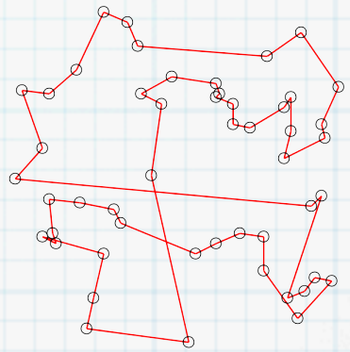
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

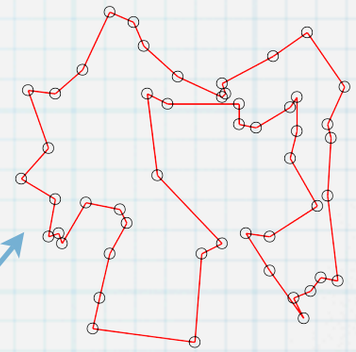


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$

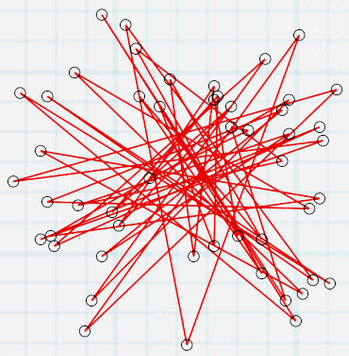


$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

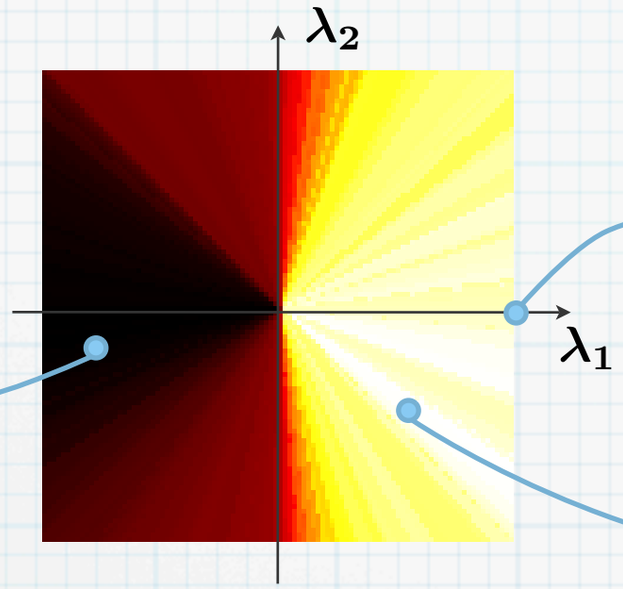
* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.
  Somit werden die gleichen Knoten in jedem Schritt gewählt.
  Am Ende sind dann beide so konstruierten Lösungen identisch.
* **Korollar 25**:
$z^{pgreedy} := \min\{z(\lambda) : \lambda \in \mathbb{R}^p\} = \min\{z(\lambda) : \lambda \in \mathbb{R}^p, \|\lambda\| = 1\}$.
* Interpretation:
  * Die Dimension des Suchraums kann um 1 reduziert werden.
  * Der Suchraum ist beschränkt.

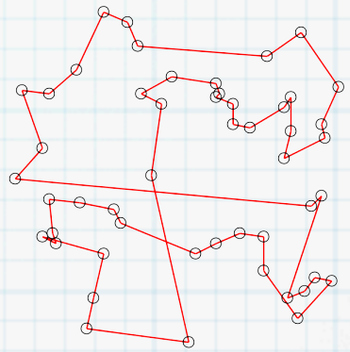
Abhängigkeit der Lösung von den Parametern

* Beispiel (Forts.):

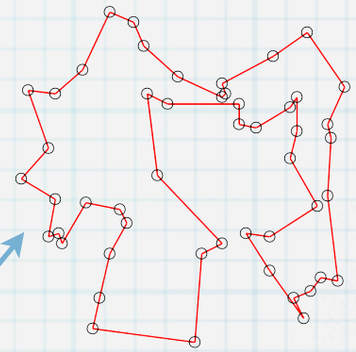


$\lambda^{worst} = (-0,94; -0,06)$
$z^{worst} = 3620$





$\lambda^{greedy} = (1; 0)$
$z^{greedy} = 677$



$\lambda^{best} = (0,6; -0,4)$
$z^{best} = 592$

* **Satz 24**:
Sei $r > 0$ und $\lambda \in \mathbb{R}^p$. Dann ist die Lösung, die PGreedy für λ erzeugt, gleich der Lösung für $r \cdot \lambda$. Insbesondere ist $z(\lambda) = z(r \cdot \lambda)$.
* Beweis:
  Es gilt $s_{i,j}(r \cdot \lambda) = r \cdot s_{i,j}(\lambda)$ für alle $r > 0$.
  Für alle i ist dann $\arg\min\{s_{i,j}(r \cdot \lambda) : j\} = \arg\min\{r \cdot s_{i,j}(\lambda) : j\} = \arg\min\{s_{i,j}(\lambda) : j\}$.
  In jedem Schritt wird ein Knoten mit kleinstem Argument gewählt.
  Somit werden die gleichen Knoten in jedem Schritt gewählt.
  Am Ende sind dann beide so konstruierten Lösungen identisch.
* **Korollar 25**:
$z^{pgreedy} := \min\{z(\lambda) : \lambda \in \mathbb{R}^p\} = \min\{z(\lambda) : \lambda \in \mathbb{R}^p, \|\lambda\| = 1\}$.
* Interpretation:
  * Die Dimension des Suchraums kann um 1 reduziert werden.
  * Der Suchraum ist beschränkt.
* Verfahren, um gute Parameter zu finden: Gittersuche oder zufallsbasiert.

Schranken durch minimale Spannbäume

Schranken durch minimale Spannbäume

* Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.

Schranken durch minimale Spannbäume

* Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
* Allerdings ist dieses ein sehr spezieller Baum: ein Weg.

Schranken durch minimale Spannbäume

* Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
* Allerdings ist dieses ein sehr spezieller Baum: ein Weg.
* Dieses gilt auch für einen minimalen Hamiltonschen Kreis.

Schranken durch minimale Spannbäume

* Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
* Allerdings ist dieses ein sehr spezieller Baum: ein Weg.
* Dieses gilt auch für einen minimalen Hamiltonschen Kreis.
* Also ist eine Lösung des minimalen Spannbaumproblems eine Abschätzung für das Handlungsreisendenproblem.

Schranken durch minimale Spannbäume

* Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
* Allerdings ist dieses ein sehr spezieller Baum: ein Weg.
* Dieses gilt auch für einen minimalen Hamiltonschen Kreis.
* Also ist eine Lösung des minimalen Spannbaumproblems eine Abschätzung für das Handlungsreisendenproblem.
* Beispiel:

Schranken durch minimale Spannbäume

- Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
- Allerdings ist dieses ein sehr spezieller Baum: ein Weg.
- Dieses gilt auch für einen minimalen Hamiltonschen Kreis.
- Also ist eine Lösung des minimalen Spannbaumproblems eine Abschätzung für das Handlungsreisendenproblem.
- Beispiel:

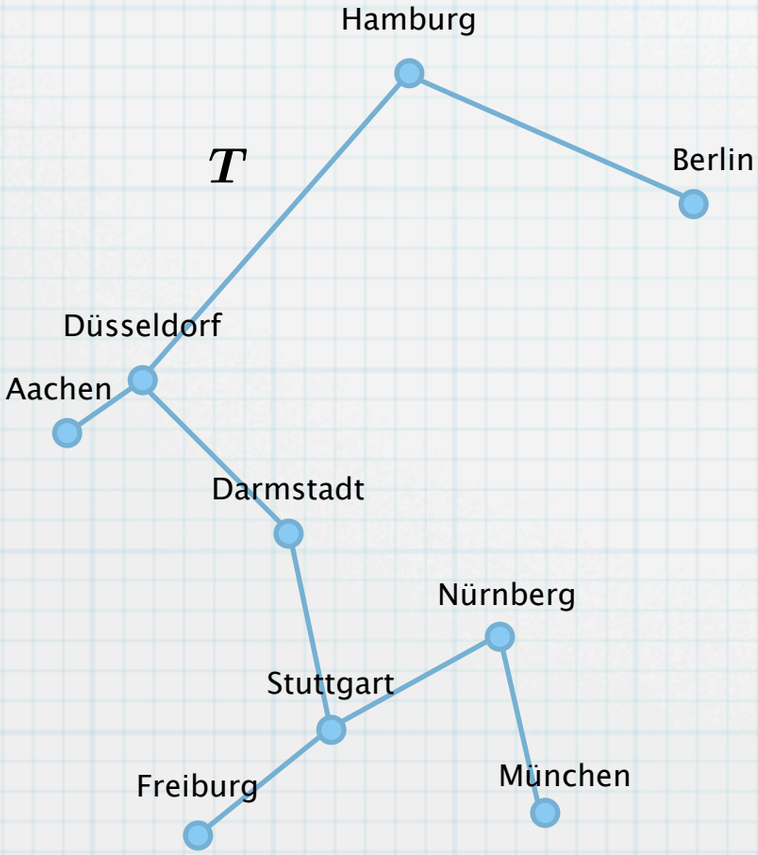
Schranken durch minimale Spannbäume

- Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
- Allerdings ist dieses ein sehr spezieller Baum: ein Weg.
- Dieses gilt auch für einen minimalen Hamiltonschen Kreis.
- Also ist eine Lösung des minimalen Spannbaumproblems eine Abschätzung für das Handlungsreisendenproblem.
- Beispiel:

	Aa	*Be*	*Da*	*Dü*	*Fr*	*Ha*	*Mü*	*Nü*	*St*
Aa	0	64	26	8	57	49	64	47	46
Be	64	0	56	57	88	29	60	44	63
Da	26	56	0	23	34	50	40	22	20
Dü	8	57	23	0	54	43	63	44	41
Fr	57	88	34	54	0	83	37	43	27
Ha	49	29	50	43	83	0	80	63	70
Mü	64	60	40	63	37	80	0	17	22
Nü	47	44	22	44	43	63	17	0	19
St	46	63	20	41	27	70	22	19	0

Schranken durch minimale Spannbäume

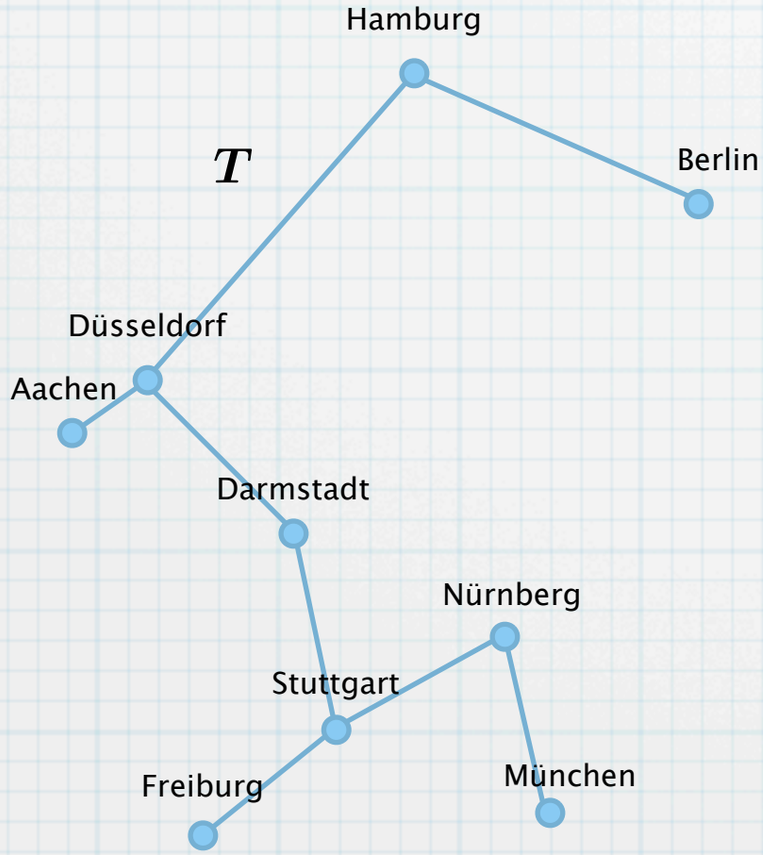
- Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
- Allerdings ist dieses ein sehr spezieller Baum: ein Weg.
- Dieses gilt auch für einen minimalen Hamiltonschen Kreis.
- Also ist eine Lösung des minimalen Spannbaumproblems eine Abschätzung für das Handlungsreisendenproblem.
- Beispiel:



	Aa	*Be*	*Da*	*Dü*	*Fr*	*Ha*	*Mü*	*Nü*	*St*
Aa	0	64	26	8	57	49	64	47	46
Be	64	0	56	57	88	29	60	44	63
Da	26	56	0	23	34	50	40	22	20
Dü	8	57	23	0	54	43	63	44	41
Fr	57	88	34	54	0	83	37	43	27
Ha	49	29	50	43	83	0	80	63	70
Mü	64	60	40	63	37	80	0	17	22
Nü	47	44	22	44	43	63	17	0	19
St	46	63	20	41	27	70	22	19	0

Schranken durch minimale Spannbäume

- Durch Entfernen einer beliebigen Kante aus einem Hamiltonschen Kreis erhält man einen aufspannenden Baum.
- Allerdings ist dieses ein sehr spezieller Baum: ein Weg.
- Dieses gilt auch für einen minimalen Hamiltonschen Kreis.
- Also ist eine Lösung des minimalen Spannbaumproblems eine Abschätzung für das Handlungsreisendenproblem.
- Beispiel:



	Aa	Be	Da	$Dü$	Fr	Ha	$Mü$	$Nü$	St
Aa	0	64	26	8	57	49	64	47	46
Be	64	0	56	57	88	29	60	44	63
Da	26	56	0	23	34	50	40	22	20
$Dü$	8	57	23	0	54	43	63	44	41
Fr	57	88	34	54	0	83	37	43	27
Ha	49	29	50	43	83	0	80	63	70
$Mü$	64	60	40	63	37	80	0	17	22
$Nü$	47	44	22	44	43	63	17	0	19
St	46	63	20	41	27	70	22	19	0

$$c(T) = 186$$

Schranken durch 1-Baum-Relaxation

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\ldots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\ldots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
* Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\ldots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
* Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
* Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
* Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
* Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  * Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
* Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
* Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  * Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  * Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
* Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
* Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  * Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  * Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  * Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
* Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
* Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  * Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  * Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  * Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).
  * Das Ergebnis (die Güte der Schranke) hängt von der Wahl des Knotens „1“ ab.

Schranken durch 1-Baum-Relaxation

* **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
* Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
* Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
* Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  * Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  * Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  * Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).
  * Das Ergebnis (die Güte der Schranke) hängt von der Wahl des Knotens „1“ ab.
  * Um die größte untere Schranke zu ermitteln, kann man es für alle n Knoten wiederholen. Dann ergibt sich eine Laufzeitkomplexität von $O(n^3)$.

Schranken durch 1-Baum-Relaxation

- **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
- Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
- Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
- Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  - Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  - Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  - Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).
  - Das Ergebnis (die Güte der Schranke) hängt von der Wahl des Knotens „1“ ab.
  - Um die größte untere Schranke zu ermitteln, kann man es für alle n Knoten wiederholen. Dann ergibt sich eine Laufzeitkomplexität von $O(n^3)$.
- Beispiel: Wählt man „Be“ als Knoten „1“, so erhält man:

Schranken durch 1-Baum-Relaxation

- **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
- Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
- Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
- Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  - Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  - Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  - Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).
  - Das Ergebnis (die Güte der Schranke) hängt von der Wahl des Knotens „1“ ab.
  - Um die größte untere Schranke zu ermitteln, kann man es für alle n Knoten wiederholen. Dann ergibt sich eine Laufzeitkomplexität von $O(n^3)$.
- Beispiel: Wählt man „Be“ als Knoten „1“, so erhält man:

Schranken durch 1-Baum-Relaxation

- **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
- Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
- Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
- Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  - Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  - Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  - Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).
  - Das Ergebnis (die Güte der Schranke) hängt von der Wahl des Knotens „1“ ab.
  - Um die größte untere Schranke zu ermitteln, kann man es für alle n Knoten wiederholen. Dann ergibt sich eine Laufzeitkomplexität von $O(n^3)$.
- Beispiel: Wählt man „Be“ als Knoten „1“, so erhält man:

Schranken durch 1-Baum-Relaxation

- **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
- Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
- Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
- Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  - Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  - Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  - Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).
  - Das Ergebnis (die Güte der Schranke) hängt von der Wahl des Knotens „1“ ab.
  - Um die größte untere Schranke zu ermitteln, kann man es für alle n Knoten wiederholen. Dann ergibt sich eine Laufzeitkomplexität von $O(n^3)$.
- Beispiel: Wählt man „Be“ als Knoten „1“, so erhält man:

Schranken durch 1-Baum-Relaxation

- **Definition 26**:
  Gegeben sei eine Instanz (K_n, c) des Handlungsreisendenproblems. Ein **1-Baum** ist ein aufspannender Baum auf dem Untergraphen $K_{\{2,\dots,n\}}$ zusammen mit zwei mit dem Knoten 1 inzidenten Kanten. Das **1-Baum-Problem** fragt nach der Konstruktion eines 1-Baumes minimalen Gewichts.
- Bemerke, dass jede TSP-Tour ein spezieller 1-Baum ist.
- Also liefert die Lösung des 1-Baum-Problems eine untere Schranke auf die minimale Tourlänge.
- Das 1-Baum-Problem kann einfach (polynomial) gelöst werden:
  - Bestimme einen minimalen Spannbaum auf $\{2, \dots, n\}$.
  - Füge zwei mit 1 inzidente Kanten minimalen Gewichts hinzu.
  - Die Komplexität dafür ist $O(n^2)$ (für Jarnik/Prim).
  - Das Ergebnis (die Güte der Schranke) hängt von der Wahl des Knotens „1“ ab.
  - Um die größte untere Schranke zu ermitteln, kann man es für alle n Knoten wiederholen. Dann ergibt sich eine Laufzeitkomplexität von $O(n^3)$.
- Beispiel: Wählt man „Be“ als Knoten „1“, so erhält man:

$$c(B) = 230$$

Approximationsalgorithmen

Approximationsalgorithmen

* **Definition 27**:
Sei $\varepsilon > 0$. Sei $\mathcal{A}$ ein Algorithmus, der für jede Instanz $\mathcal{I}$ eines Optimierungsproblems $\mathcal{P}$ eine zulässige Lösung erzeugt. Sei $c_{\mathcal{A}}(\mathcal{I})$ der Zielfunktionswert dieser Lösung und sei $c^*(\mathcal{I})$ der Zielfunktionswert einer Optimallösung von $\mathcal{I}$. Gilt $|c_{\mathcal{A}}(\mathcal{I})/c^*(\mathcal{I}) - 1| \leq \varepsilon$, so wird $\mathcal{A}$ als **ε-approximativer Algorithmus** für $\mathcal{P}$ bezeichnet.

Approximationsalgorithmen

* **Definition 27**:
  Sei $\varepsilon > 0$. Sei $\mathcal{A}$ ein Algorithmus, der für jede Instanz $\mathcal{I}$ eines Optimierungsproblems $\mathcal{P}$ eine zulässige Lösung erzeugt. Sei $c_{\mathcal{A}}(\mathcal{I})$ der Zielfunktionswert dieser Lösung und sei $c^*(\mathcal{I})$ der Zielfunktionswert einer Optimallösung von $\mathcal{I}$. Gilt $|c_{\mathcal{A}}(\mathcal{I})/c^*(\mathcal{I}) - 1| \leq \varepsilon$, so wird $\mathcal{A}$ als **ε-approximativer Algorithmus** für $\mathcal{P}$ bezeichnet.
* Beispiel: Ein 1-approximativer Algorithmus für TSP konstruiert Touren, deren Länge maximal doppelt so groß ist wie die einer optimalen Tour.

Approximationsalgorithmen

* **Definition 27**:
  Sei $\varepsilon > 0$. Sei $\mathcal{A}$ ein Algorithmus, der für jede Instanz $\mathcal{I}$ eines Optimierungsproblems $\mathcal{P}$ eine zulässige Lösung erzeugt. Sei $c_{\mathcal{A}}(\mathcal{I})$ der Zielfunktionswert dieser Lösung und sei $c^*(\mathcal{I})$ der Zielfunktionswert einer Optimallösung von $\mathcal{I}$. Gilt $|c_{\mathcal{A}}(\mathcal{I})/c^*(\mathcal{I}) - 1| \leq \varepsilon$, so wird $\mathcal{A}$ als **ε-approximativer Algorithmus** für $\mathcal{P}$ bezeichnet.
* Beispiel: Ein 1-approximativer Algorithmus für TSP konstruiert Touren, deren Länge maximal doppelt so groß ist wie die einer optimalen Tour.
* Besonders interessant sind polynomiale Approximationsalgorithmen mit kleinem ε.

Approximationsalgorithmen

* **Definition 27**:
  Sei $\varepsilon > 0$. Sei $\mathcal{A}$ ein Algorithmus, der für jede Instanz $\mathcal{I}$ eines Optimierungsproblems $\mathcal{P}$ eine zulässige Lösung erzeugt. Sei $c_{\mathcal{A}}(\mathcal{I})$ der Zielfunktionswert dieser Lösung und sei $c^*(\mathcal{I})$ der Zielfunktionswert einer Optimallösung von $\mathcal{I}$. Gilt $|c_{\mathcal{A}}(\mathcal{I})/c^*(\mathcal{I}) - 1| \leq \varepsilon$, so wird $\mathcal{A}$ als **ε-approximativer Algorithmus** für $\mathcal{P}$ bezeichnet.
* Beispiel: Ein 1-approximativer Algorithmus für TSP konstruiert Touren, deren Länge maximal doppelt so groß ist wie die einer optimalen Tour.
* Besonders interessant sind polynomiale Approximationsalgorithmen mit kleinem ε.
* Jedoch: Der folgende Satz deutet an, dass es „schwer“ ist, das Handlungsreisendenproblem mit einem Approximationsfehler von ε zu lösen (für beliebiges $\varepsilon > 0$).

Approximationsalgorithmen

- **Definition 27**:
  Sei $\varepsilon > 0$. Sei $\mathcal{A}$ ein Algorithmus, der für jede Instanz $\mathcal{I}$ eines Optimierungsproblems $\mathcal{P}$ eine zulässige Lösung erzeugt. Sei $c_{\mathcal{A}}(\mathcal{I})$ der Zielfunktionswert dieser Lösung und sei $c^*(\mathcal{I})$ der Zielfunktionswert einer Optimallösung von $\mathcal{I}$. Gilt $|c_{\mathcal{A}}(\mathcal{I})/c^*(\mathcal{I}) - 1| \leq \varepsilon$, so wird $\mathcal{A}$ als **ε-approximativer Algorithmus** für $\mathcal{P}$ bezeichnet.
- Beispiel: Ein 1-approximativer Algorithmus für TSP konstruiert Touren, deren Länge maximal doppelt so groß ist wie die einer optimalen Tour.
- Besonders interessant sind polynomiale Approximationsalgorithmen mit kleinem ε.
- Jedoch: Der folgende Satz deutet an, dass es „schwer“ ist, das Handlungsreisendenproblem mit einem Approximationsfehler von ε zu lösen (für beliebiges $\varepsilon > 0$).
- Wir benutzen im Folgenden (ohne Beweis), dass das Entscheidungsproblem, ob ein gegebener Graph einen Hamiltonschen Zyklus hat, ist ein einem bestimmten Sinne ein „schweres“ Problem ist.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.

  Dann gibt es eine Optimallösung von $\mathcal{I}$ mit $c^*(\mathcal{I}) = |V|$.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.
  Dann gibt es eine Optimallösung von $\mathcal{I}$ mit $c^*(\mathcal{I}) = |V|$.
  $\mathcal{A}$ findet eine Tour vom Gewicht $c_{\mathcal{A}}(\mathcal{I}) \leq (1 + \varepsilon) \cdot |V|$.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.

  Dann gibt es eine Optimallösung von $\mathcal{I}$ mit $c^*(\mathcal{I}) = |V|$.

  $\mathcal{A}$ findet eine Tour vom Gewicht $c_{\mathcal{A}}(\mathcal{I}) \leq (1 + \varepsilon) \cdot |V|$.

  Diese Tour besteht nur aus Kanten aus E.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.

  Dann gibt es eine Optimallösung von $\mathcal{I}$ mit $c^*(\mathcal{I}) = |V|$.

  $\mathcal{A}$ findet eine Tour vom Gewicht $c_{\mathcal{A}}(\mathcal{I}) \leq (1 + \varepsilon) \cdot |V|$.

  Diese Tour besteht nur aus Kanten aus E.

    Anderfalls wäre $c_{\mathcal{A}}(\mathcal{I}) \geq (|V| - 1) + (2 + \varepsilon \cdot |V|) = (1 + \varepsilon) \cdot |V| + 1$. Widerspruch.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.

  Dann gibt es eine Optimallösung von $\mathcal{I}$ mit $c^*(\mathcal{I}) = |V|$.

  $\mathcal{A}$ findet eine Tour vom Gewicht $c_{\mathcal{A}}(\mathcal{I}) \leq (1 + \varepsilon) \cdot |V|$.

  Diese Tour besteht nur aus Kanten aus E.

  Anderfalls wäre $c_{\mathcal{A}}(\mathcal{I}) \geq (|V| - 1) + (2 + \varepsilon \cdot |V|) = (1 + \varepsilon) \cdot |V| + 1$. Widerspruch.

  Also entspricht die Tour in K_V einem Hamiltonschen Kreis in G.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.

  Dann gibt es eine Optimallösung von $\mathcal{I}$ mit $c^*(\mathcal{I}) = |V|$.

  $\mathcal{A}$ findet eine Tour vom Gewicht $c_{\mathcal{A}}(\mathcal{I}) \leq (1 + \varepsilon) \cdot |V|$.

  Diese Tour besteht nur aus Kanten aus E.

  Anderfalls wäre $c_{\mathcal{A}}(\mathcal{I}) \geq (|V| - 1) + (2 + \varepsilon \cdot |V|) = (1 + \varepsilon) \cdot |V| + 1$. Widerspruch.

  Also entspricht die Tour in K_V einem Hamiltonschen Kreis in G.

  Somit hat G genau dann einen Hamiltonschen Kreis, wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ berechnet.

Nicht-Approximierbarkeit des Handlungsreisendenproblem

* **Satz 28** (Sahni und Gonzales, 1976):
  Wenn es einen ε-approximativen polynomialen Algorithmus $\mathcal{A}$ für das Handlungsreisendenproblem in K_V gibt, dann gibt es einen polynomialen Algorithmus zur Bestimmung eines Hamiltonschen Zyklus in einem Graphen $G = (V, E)$.
* Beweis:

  Sei $G = (V, E)$ ein (o.B.d.A.) zusammenhängender Graph.

  Dann ist $\mathcal{I} := (K_V, c)$ mit $c_{i,j} := 1$ für $\{i, j\} \in E$ und $c_{i,j} := 2 + \varepsilon \cdot |V|$ für $\{i, j\} \notin E$ eine Instanz des Handlungsreisendenproblems.

  Wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ findet, dann hat G einen Hamiltonschen Zyklus.

  Umgekehrt nehmen wir an, dass G einen Hamiltonschen Zyklus besitzt.

  Dann gibt es eine Optimallösung von $\mathcal{I}$ mit $c^*(\mathcal{I}) = |V|$.

  $\mathcal{A}$ findet eine Tour vom Gewicht $c_{\mathcal{A}}(\mathcal{I}) \leq (1 + \varepsilon) \cdot |V|$.

  Diese Tour besteht nur aus Kanten aus E.

  Anderfalls wäre $c_{\mathcal{A}}(\mathcal{I}) \geq (|V| - 1) + (2 + \varepsilon \cdot |V|) = (1 + \varepsilon) \cdot |V| + 1$. Widerspruch.

  Also entspricht die Tour in K_V einem Hamiltonschen Kreis in G.

  Somit hat G genau dann einen Hamiltonschen Kreis, wenn $\mathcal{A}$ eine Tour mit Gewicht $|V|$ berechnet.

  Ist letzteres in polynomialer Laufzeit möglich, so wäre auch ersteres einfach.

Erzeugende Eulersche Multigraphen

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei $i_1, i_2, \ldots, i_n$ eine Permutation der $n := |V|$ Knoten von V ist, und

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei $i_1, i_2, \ldots, i_n$ eine Permutation der $n := |V|$ Knoten von V ist, und $P_1, \ldots, P_n$ (evtl. leere) Knotenfolgen auf V sind.

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei $i_1, i_2, \ldots, i_n$ eine Permutation der $n := |V|$ Knoten von V ist, und $P_1, \ldots, P_n$ (evtl. leere) Knotenfolgen auf V sind.

  Dann ist $C = (i_1, i_2, \ldots, i_n, i_1)$ eine TSP–Tour.

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei $i_1, i_2, \ldots, i_n$ eine Permutation der $n := |V|$ Knoten von V ist, und $P_1, \ldots, P_n$ (evtl. leere) Knotenfolgen auf V sind.

  Dann ist $C = (i_1, i_2, \ldots, i_n, i_1)$ eine TSP–Tour.

  Es gilt die Abschätzung: $c(C) = \sum_{k=1}^{n} c_{i_k, i_{k+1}} \leq c(K)$, wobei $i_{n+1} := i_1$,

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei $i_1, i_2, \ldots, i_n$ eine Permutation der $n := |V|$ Knoten von V ist, und $P_1, \ldots, P_n$ (evtl. leere) Knotenfolgen auf V sind.

  Dann ist $C = (i_1, i_2, \ldots, i_n, i_1)$ eine TSP–Tour.

  Es gilt die Abschätzung: $c(C) = \sum_{k=1}^{n} c_{i_k, i_{k+1}} \leq c(K)$, wobei $i_{n+1} := i_1$,

  da wegen der Dreiecksungleichung $c_{i_k, i_{k+1}} \leq c(i_k, P_k, i_{k+1})$.

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei $i_1, i_2, \ldots, i_n$ eine Permutation der $n := |V|$ Knoten von V ist, und $P_1, \ldots, P_n$ (evtl. leere) Knotenfolgen auf V sind.

  Dann ist $C = (i_1, i_2, \ldots, i_n, i_1)$ eine TSP–Tour.

  Es gilt die Abschätzung: $c(C) = \sum_{k=1}^{n} c_{i_k, i_{k+1}} \leq c(K)$, wobei $i_{n+1} := i_1$, da wegen der Dreiecksungleichung $c_{i_k, i_{k+1}} \leq c(i_k, P_k, i_{k+1})$.

  Jede Kante von E kommt in K genau ein Mal vor. Also ist $c(K) = c(E)$.

Erzeugende Eulersche Multigraphen

* **Definition 29**:
  Sei V eine Knotenmenge. Ein zusammenhängender Eulerscher Multigraph auf V heißt **erzeugender Eulerscher Multigraph** für K_V.
* **Lemma 30**:
  Sei $\mathcal{I}$ eine Instanz des metrischen TSP (K_V, c). Sei $G = (V, E)$ ein erzeugender Eulerscher Multigraph für K_V. Dann kann eine Tour C mit $c(C) \leq c(E)$ in polynomialer Laufzeit konstruiert werden.
* Beweis:

  Da G Eulersch ist, kann eine Eulertour K mittels des Algorithmus von Fleury in polynomialer Laufzeit gefunden werden.

  Die Knotenfolge von K ist gegeben durch $K = (i_1, P_1, i_2, P_2, \ldots, i_n, P_n, i_1)$, wobei $i_1, i_2, \ldots, i_n$ eine Permutation der $n := |V|$ Knoten von V ist, und $P_1, \ldots, P_n$ (evtl. leere) Knotenfolgen auf V sind.

  Dann ist $C = (i_1, i_2, \ldots, i_n, i_1)$ eine TSP–Tour.

  Es gilt die Abschätzung: $c(C) = \sum_{k=1}^{n} c_{i_k, i_{k+1}} \leq c(K)$, wobei $i_{n+1} := i_1$, da wegen der Dreiecksungleichung $c_{i_k, i_{k+1}} \leq c(i_k, P_k, i_{k+1})$.

  Jede Kante von E kommt in K genau ein Mal vor. Also ist $c(K) = c(E)$.

  Es folgt $c(C) \leq c(E)$.

Ein 1-approximativer Algorithmus für mTSP

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

Ein 1-approximativer Algorithmus für mTSP

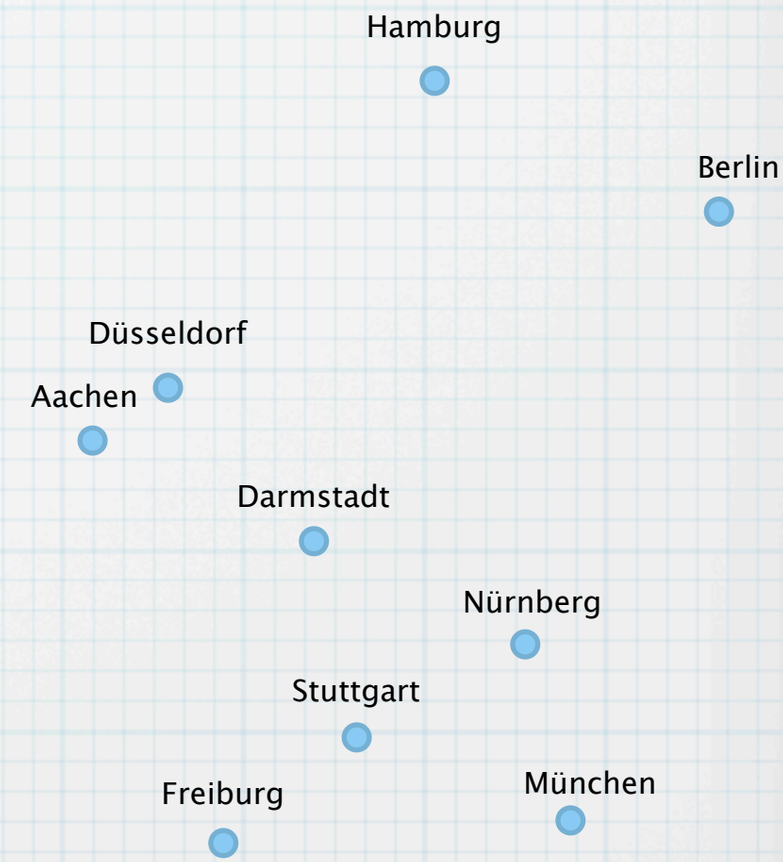
* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

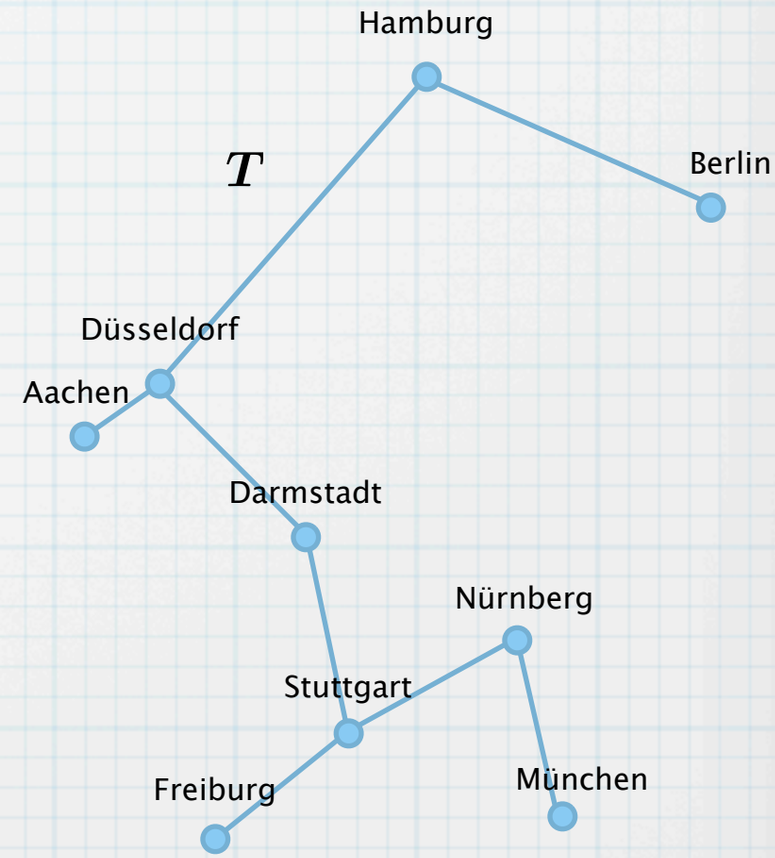
(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)



Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

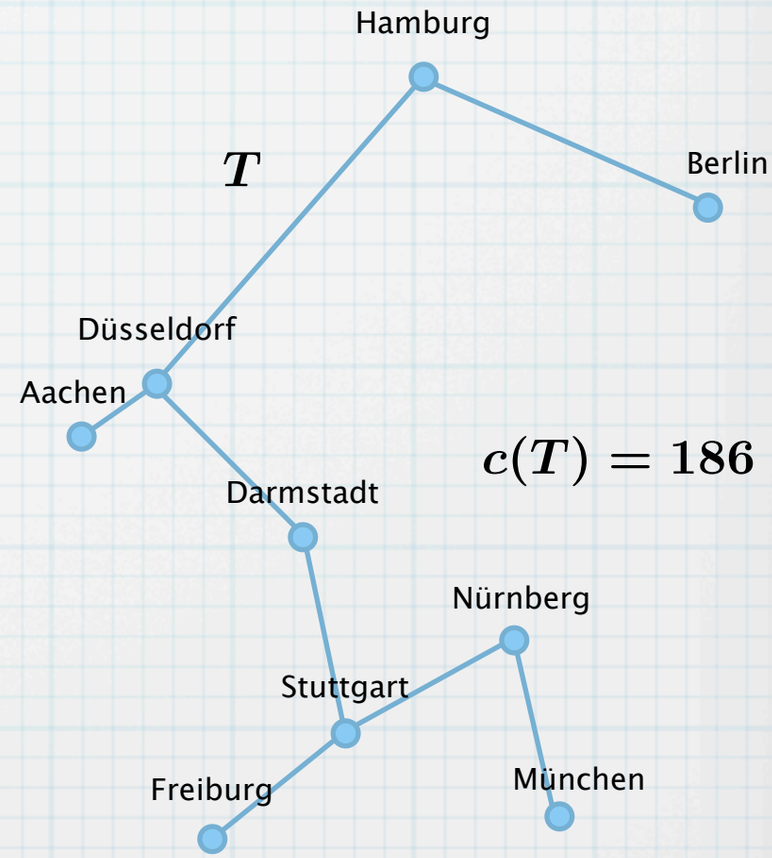
(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)



Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)

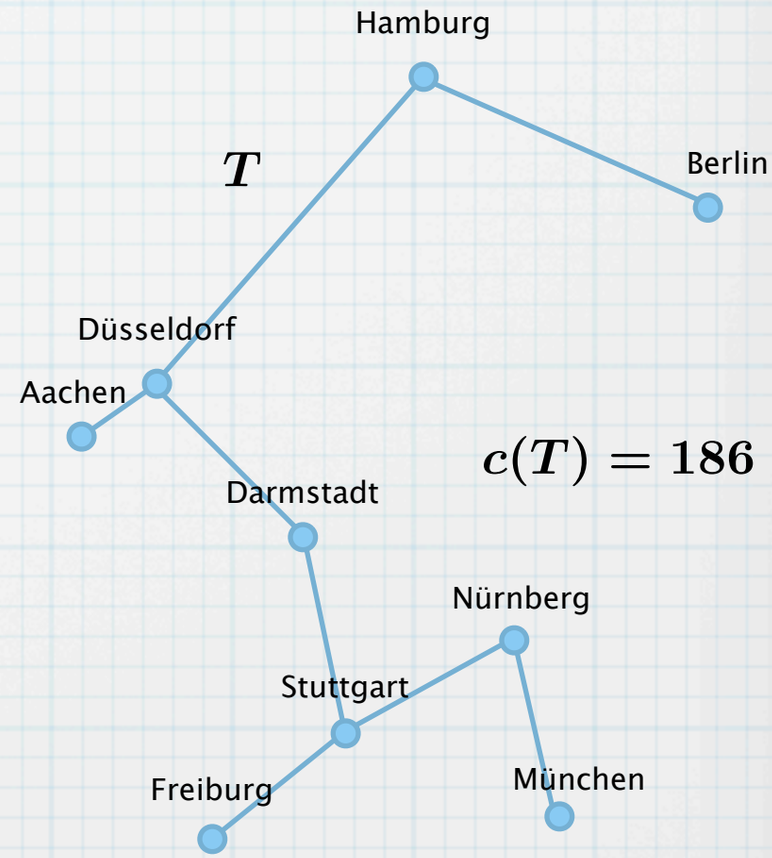


$c(T) = 186$

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht

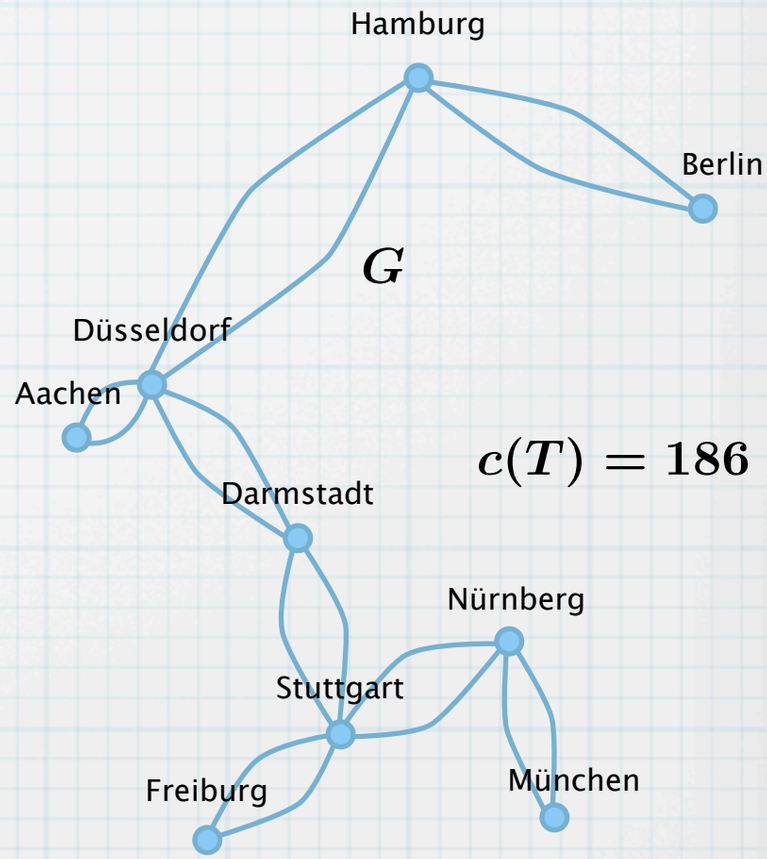


$c(T) = 186$

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

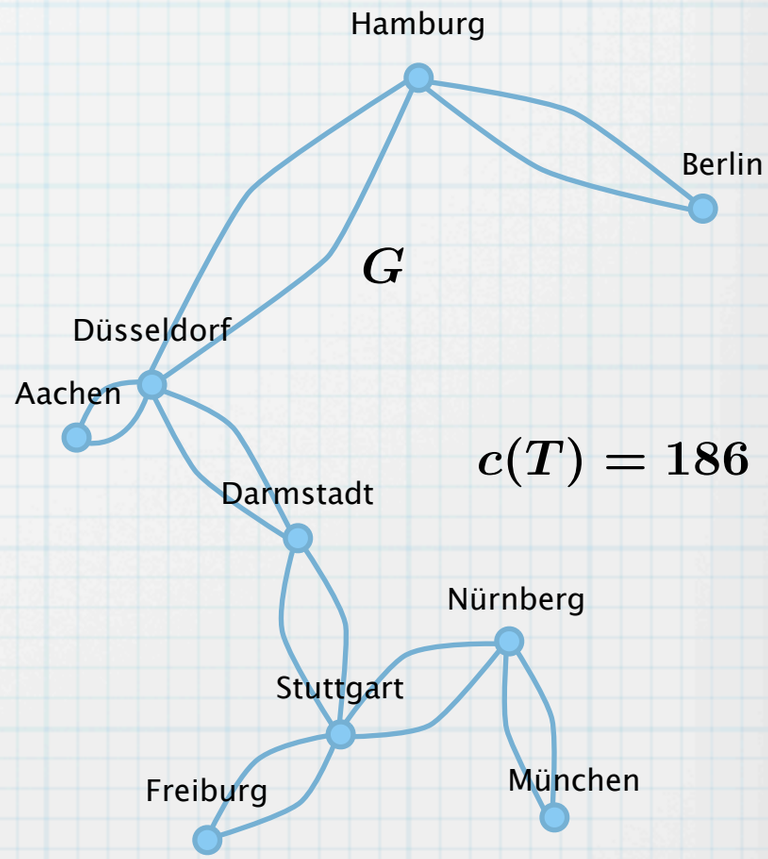
(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht



Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

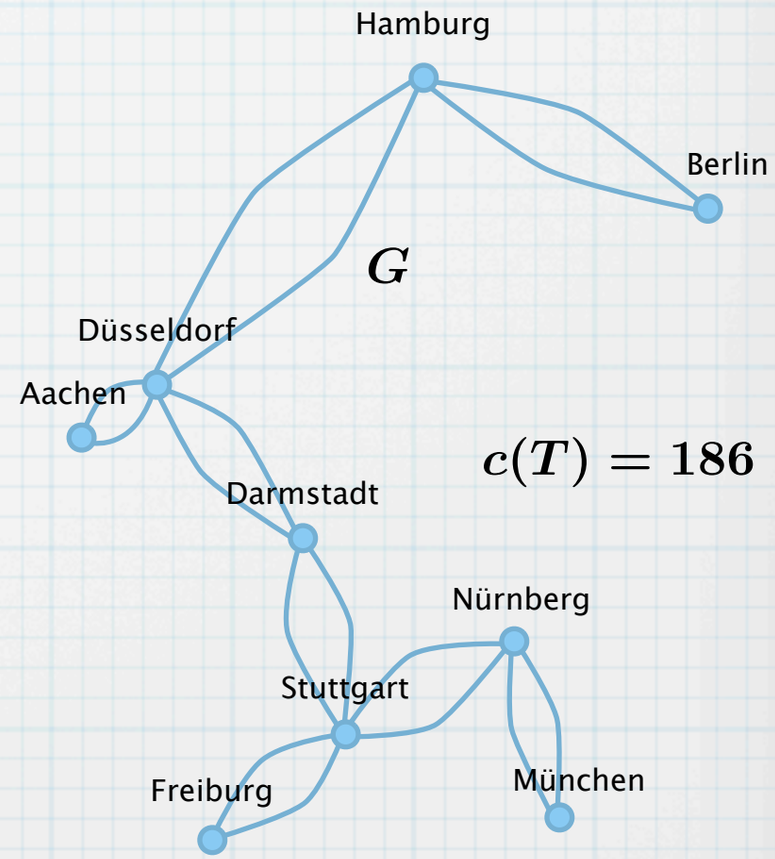
(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G



Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G

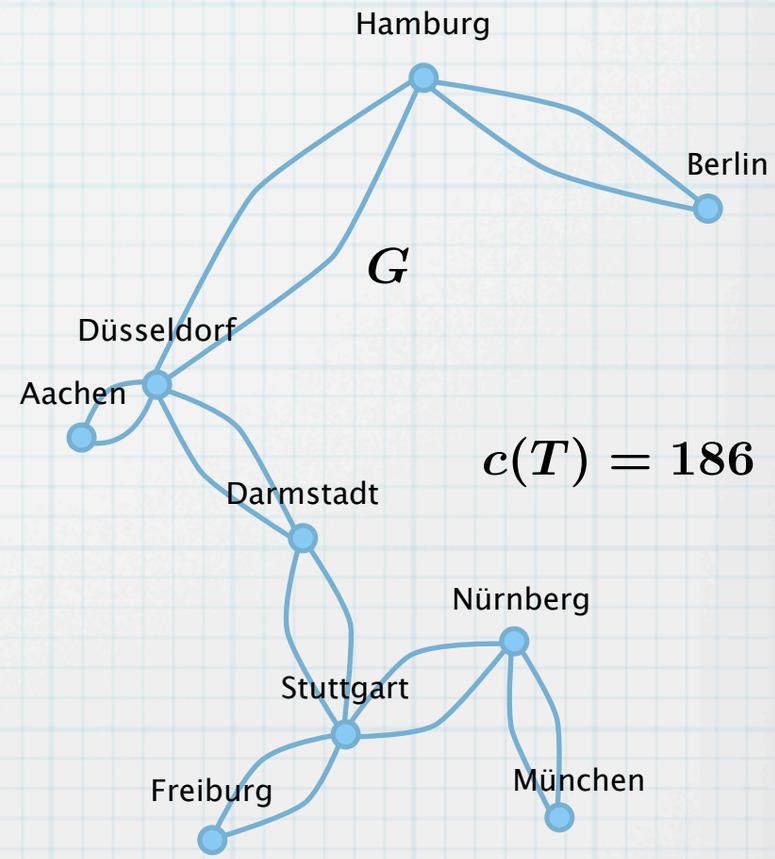


K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C

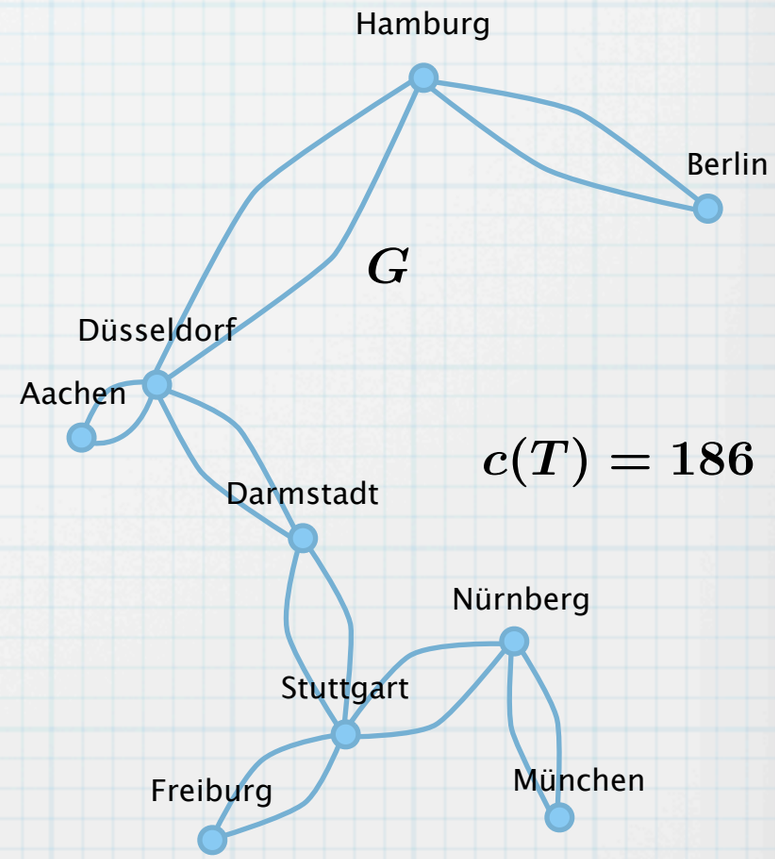


K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C

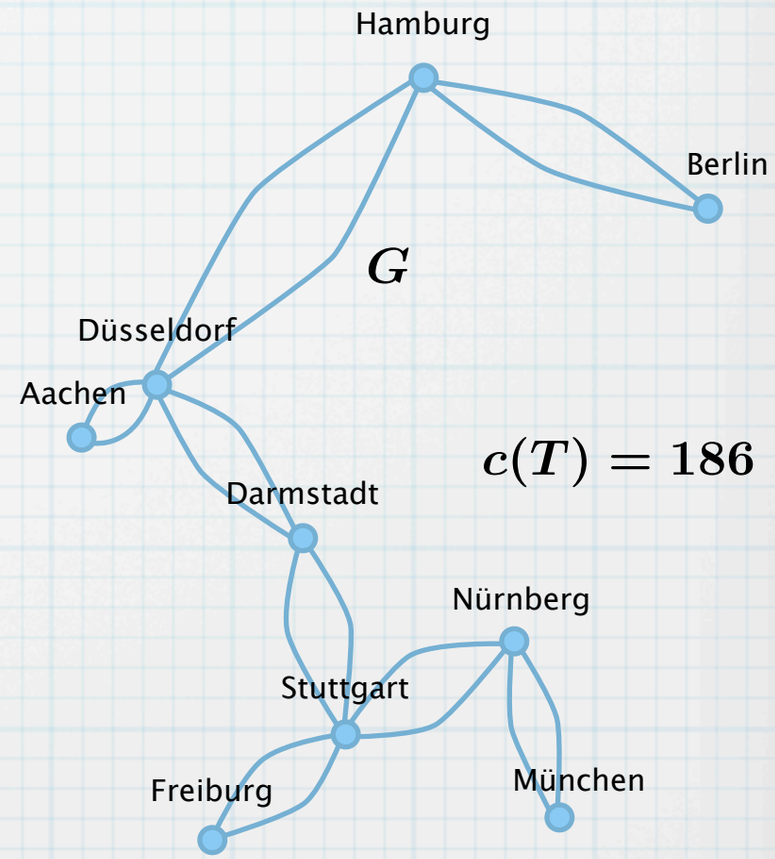


K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C



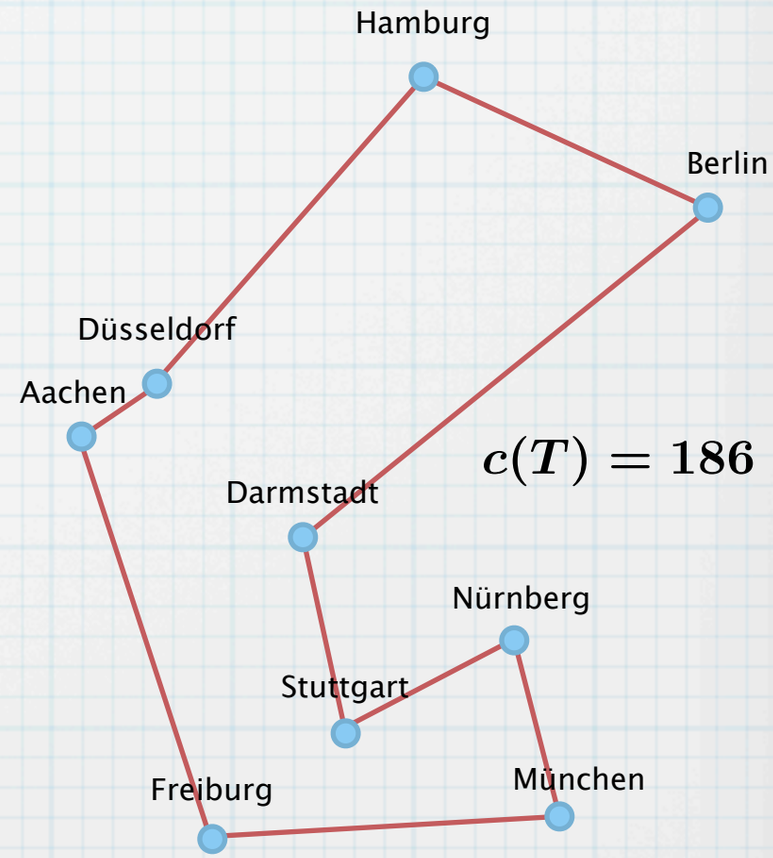
K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C



$c(T) = 186$

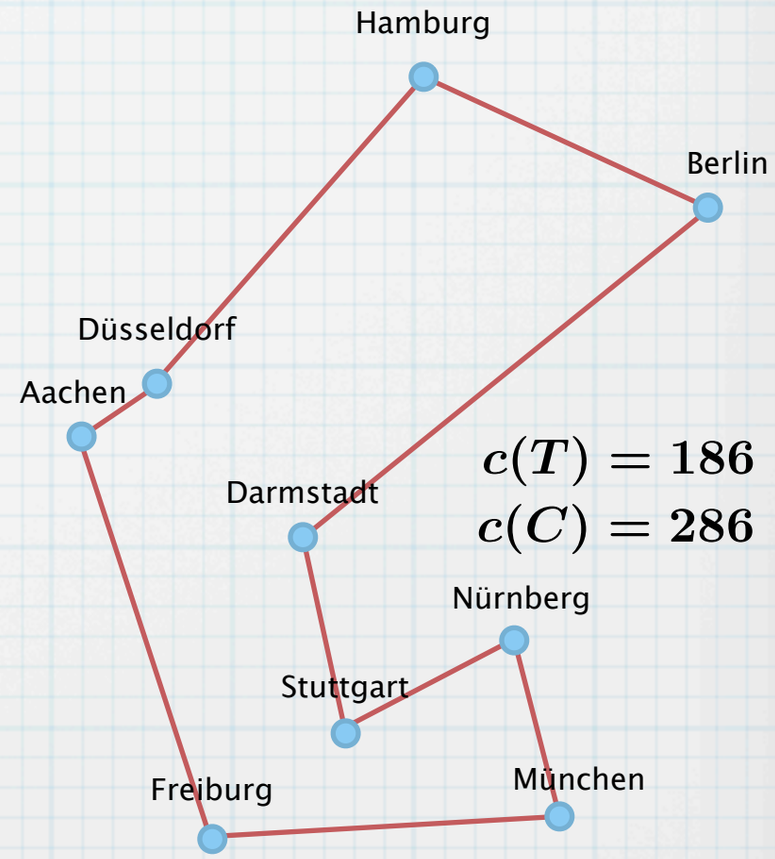
K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C



$c(T) = 186$
$c(C) = 286$

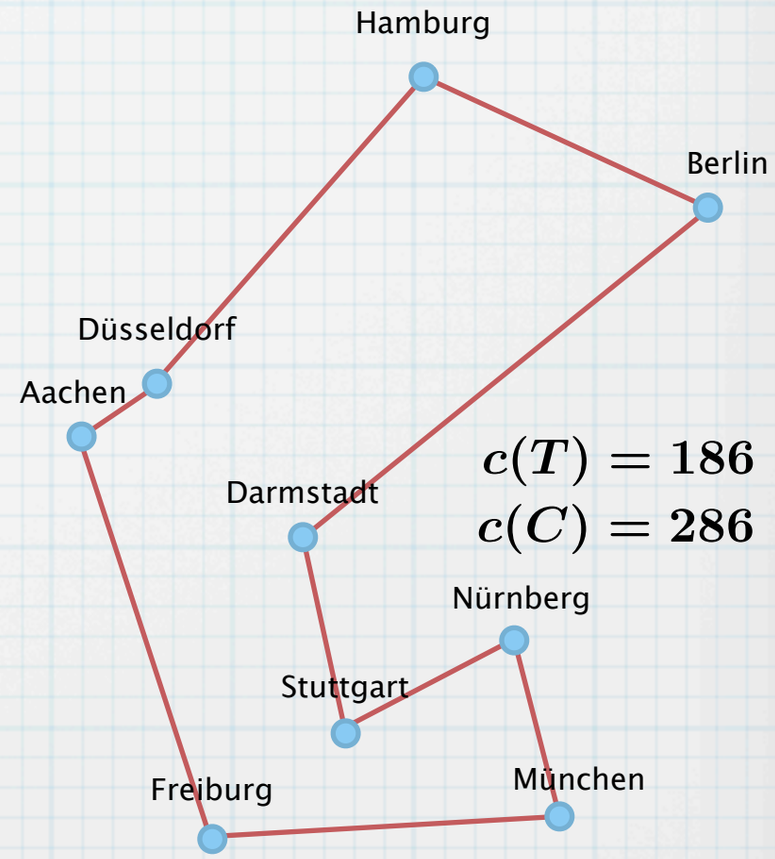
K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**



K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

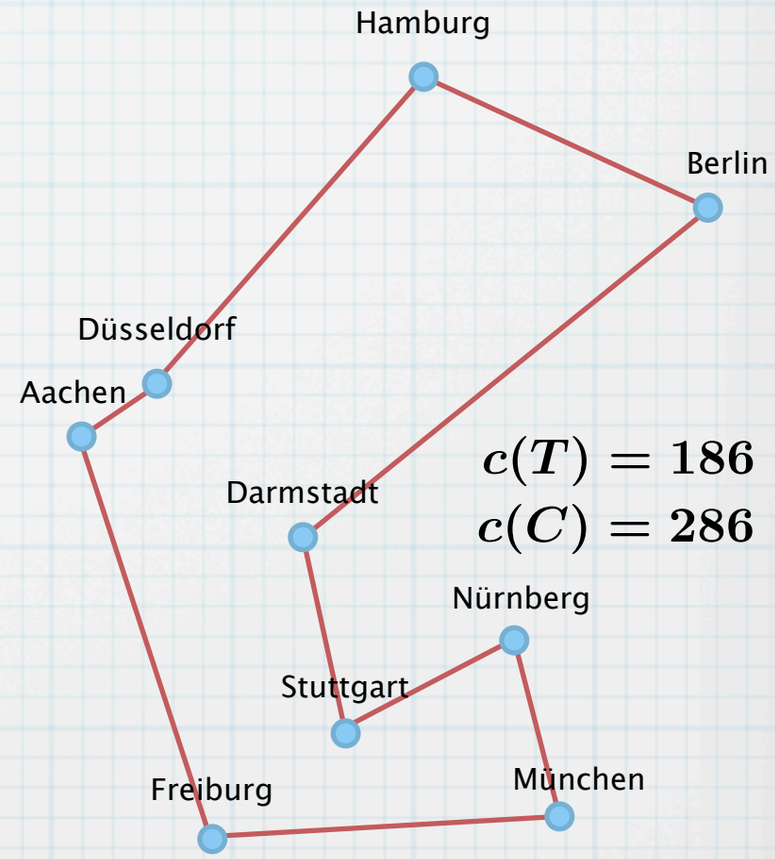
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.



K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

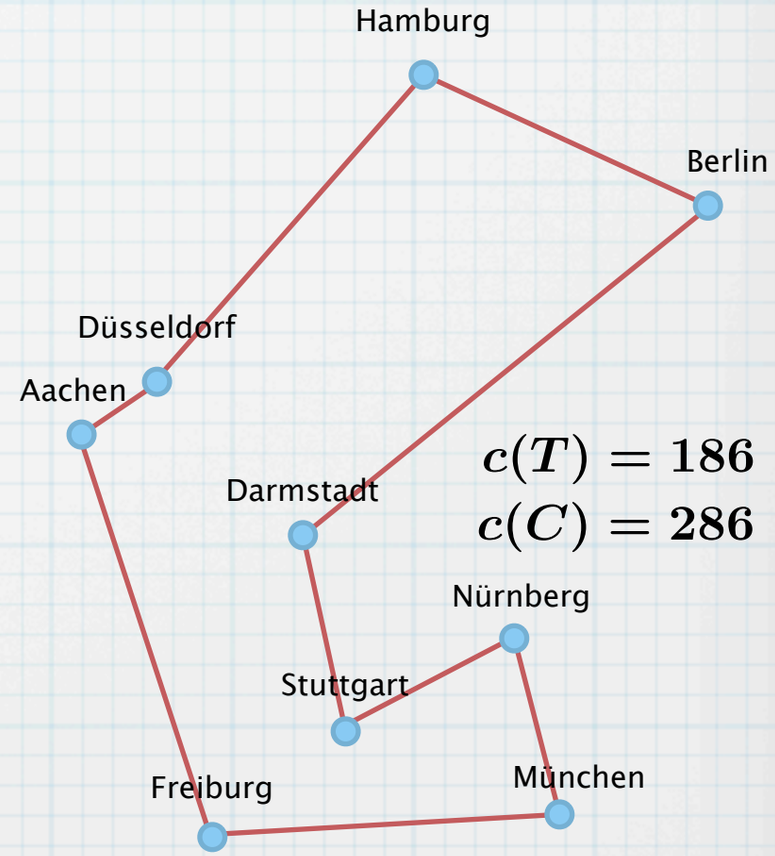
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:



K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

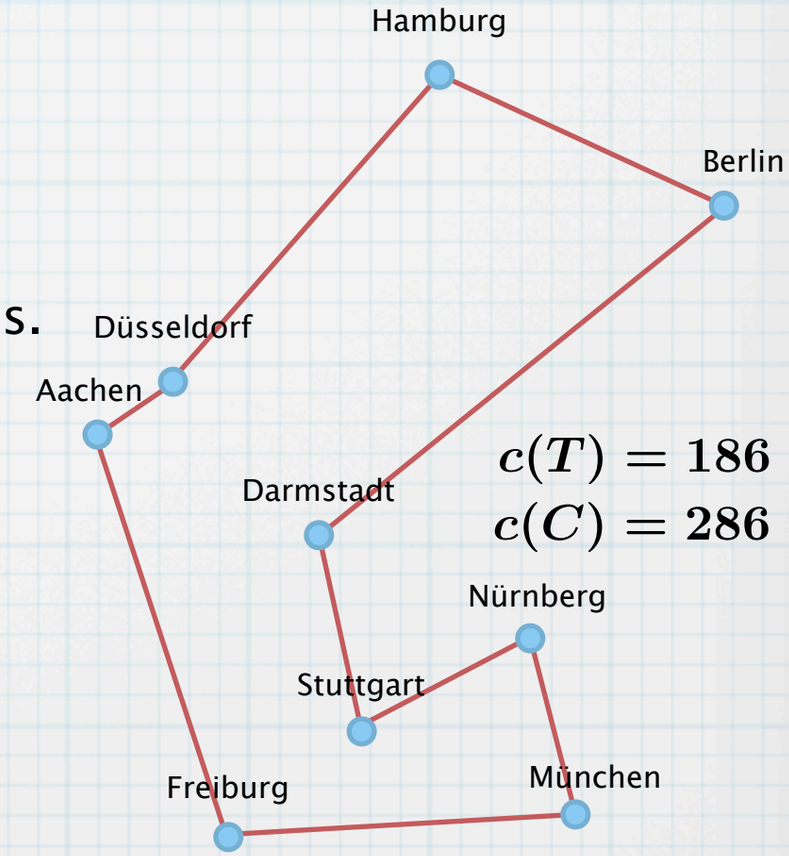
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.



K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

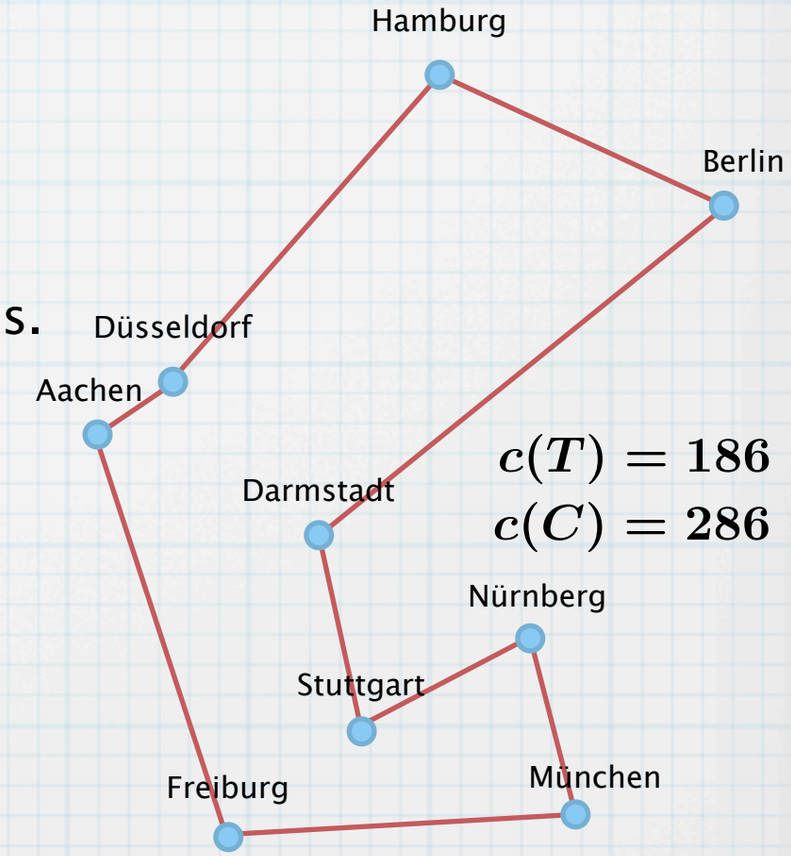
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.
  $O(n)$ für (3), die Erzeugung von G.



K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

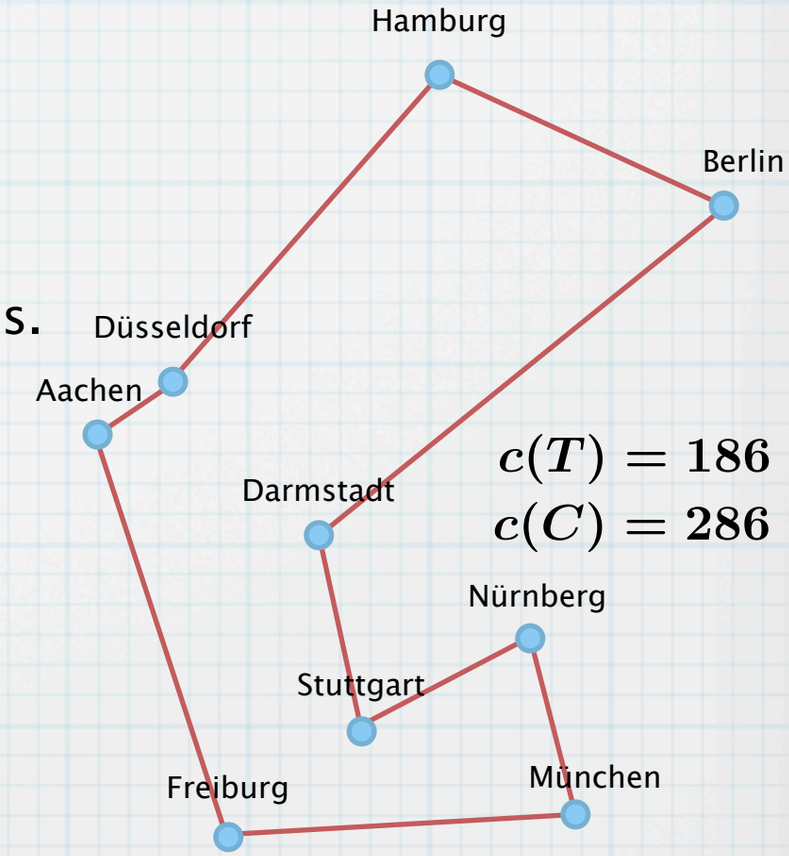
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.
  $O(n)$ für (3), die Erzeugung von G.
  $O(n)$ für (4), die Bestimmung einer Eulertour.



$c(T) = 186$
$c(C) = 286$

K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

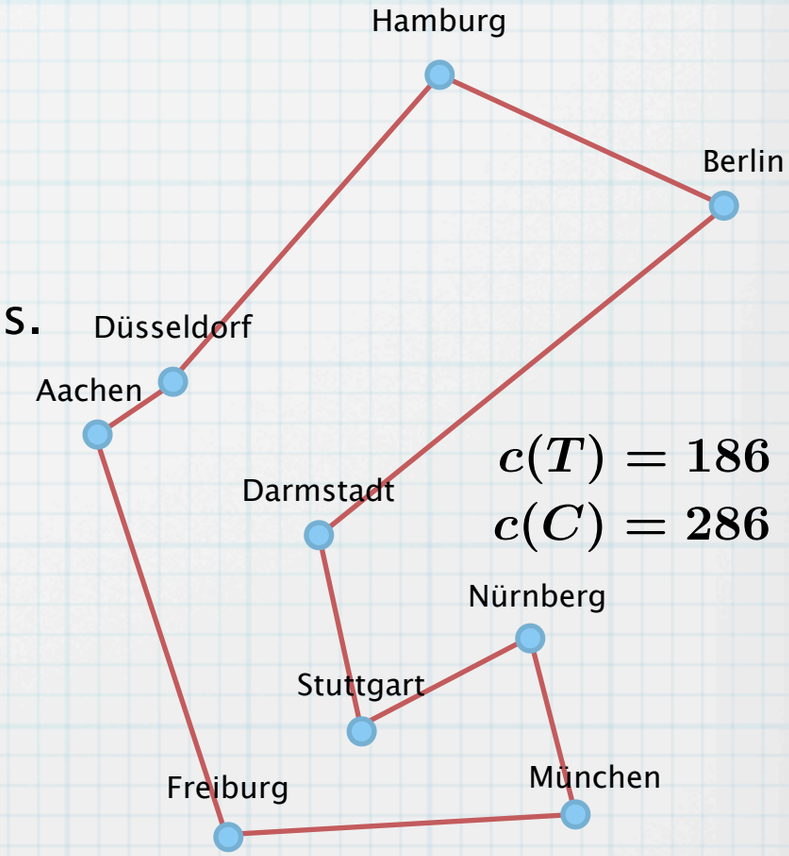
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.
  $O(n)$ für (3), die Erzeugung von G.
  $O(n)$ für (4), die Bestimmung einer Eulertour.
  $O(n)$ für (5), die Wahl einer TSP-Tour gemäß Lemma 30.



$c(T) = 186$
$c(C) = 286$

K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

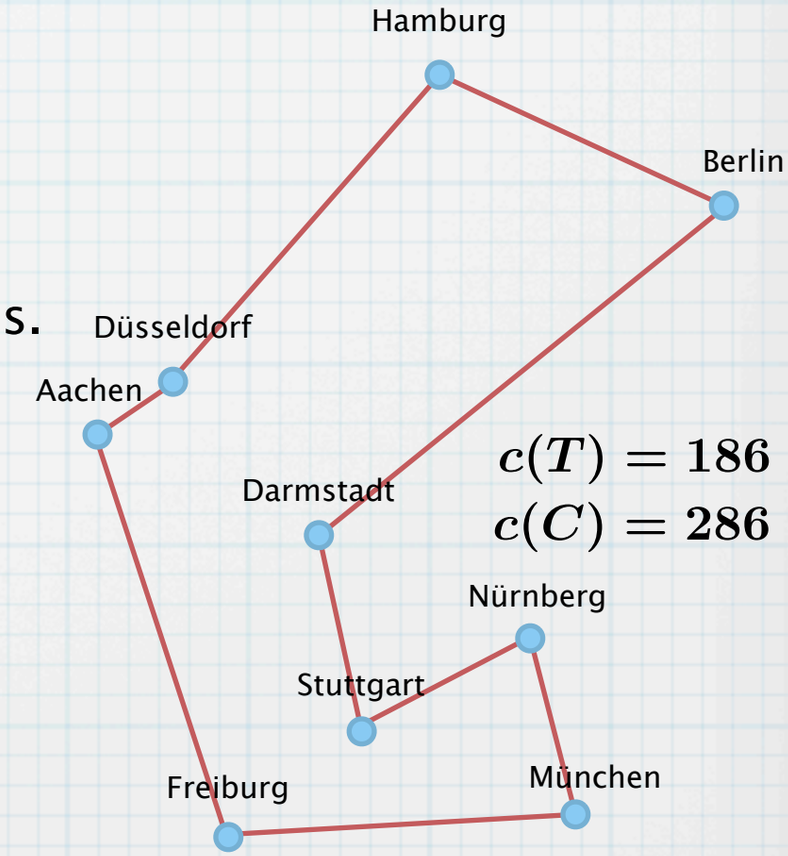
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.
  $O(n)$ für (3), die Erzeugung von G.
  $O(n)$ für (4), die Bestimmung einer Eulertour.
  $O(n)$ für (5), die Wahl einer TSP-Tour gemäß Lemma 30.
  Für diese Tour gilt $c(C) \leq 2c(T)$.



$c(T) = 186$
$c(C) = 286$

K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

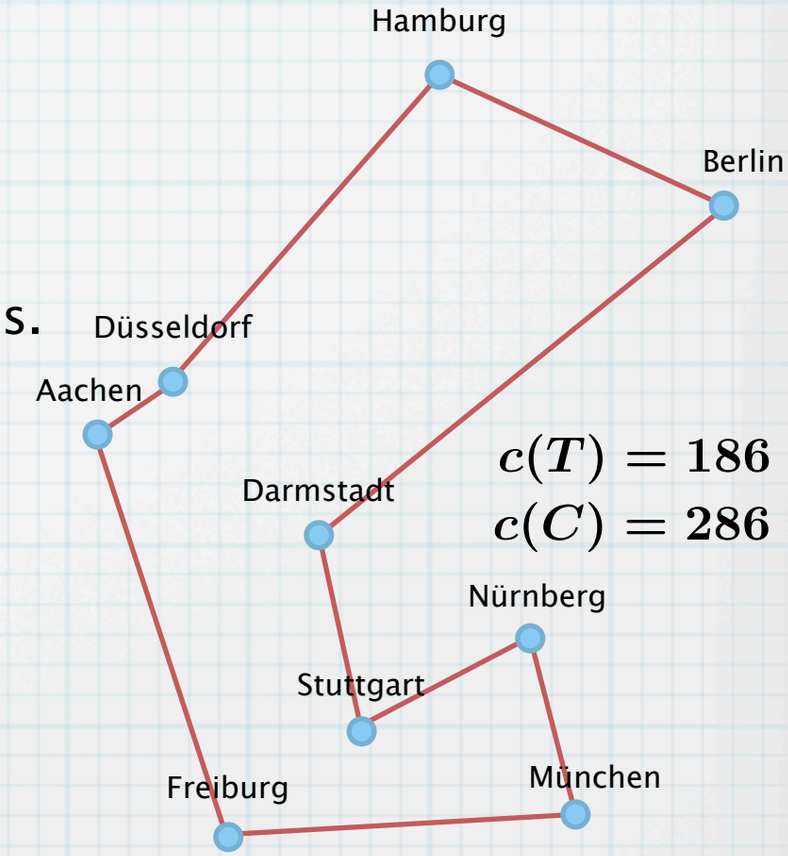
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.
  $O(n)$ für (3), die Erzeugung von G.
  $O(n)$ für (4), die Bestimmung einer Eulertour.
  $O(n)$ für (5), die Wahl einer TSP-Tour gemäß Lemma 30.
  Für diese Tour gilt $c(C) \leq 2c(T)$.
  Eine Tour hat mindestens die Länge $c(T)$.



K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

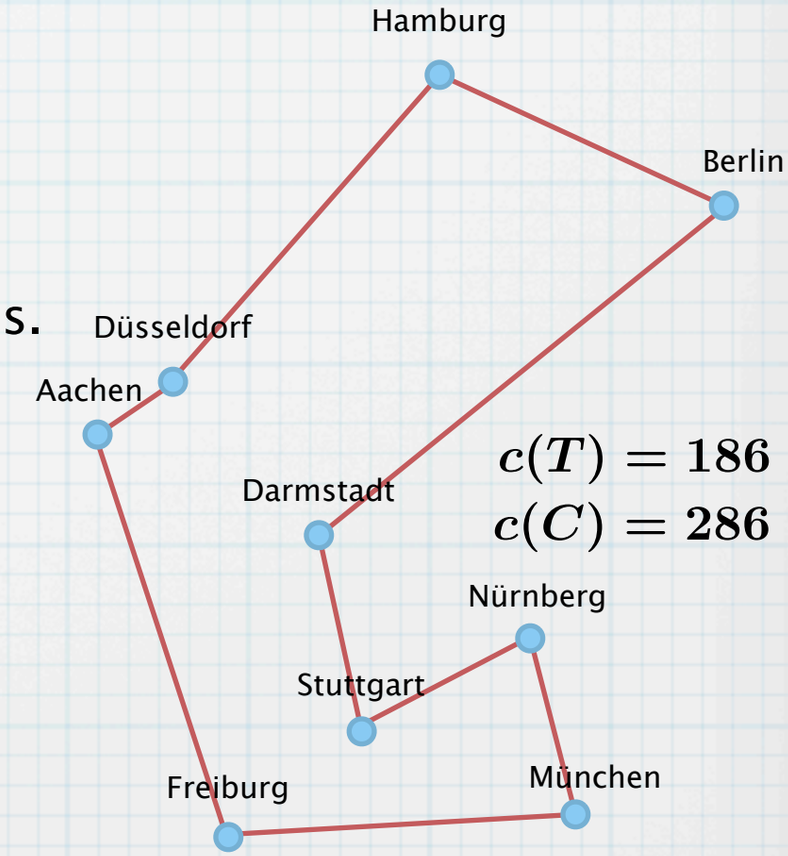
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.
  $O(n)$ für (3), die Erzeugung von G.
  $O(n)$ für (4), die Bestimmung einer Eulertour.
  $O(n)$ für (5), die Wahl einer TSP-Tour gemäß Lemma 30.
  Für diese Tour gilt $c(C) \leq 2c(T)$.
  Eine Tour hat mindestens die Länge $c(T)$.
  Also $c(C) \geq c(T)$.



$c(T) = 186$
$c(C) = 286$

K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

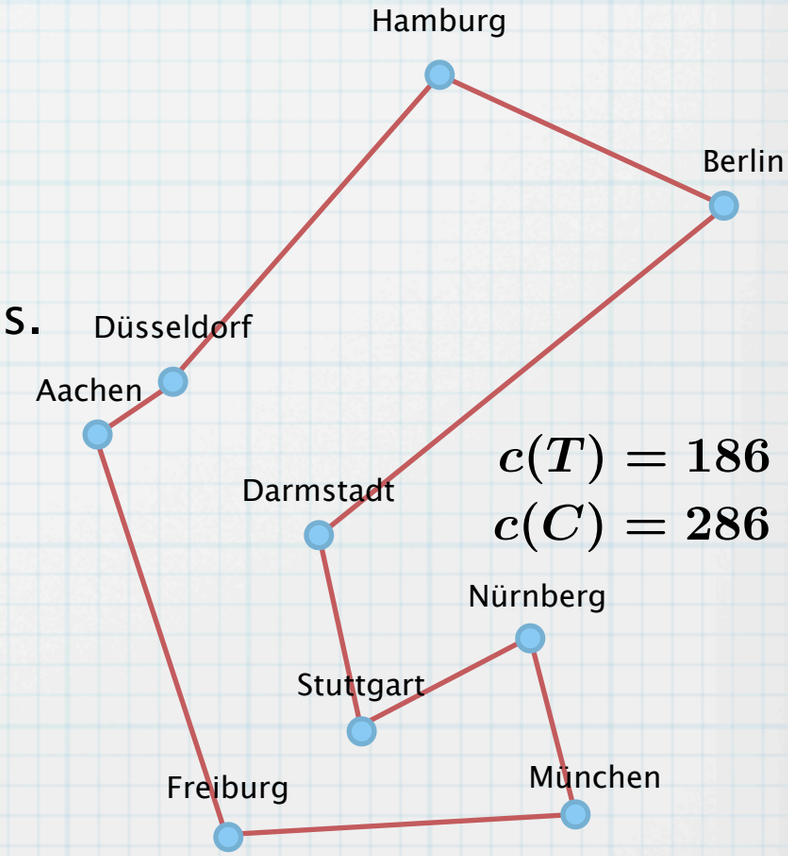
C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Ein 1-approximativer Algorithmus für mTSP

* Eingabe: Instanz (K_n, c) für mTSP
* Ausgabe: 1-approximative Tour C

(1) **algorithm** treeTour
(2) bestimme minimalen Spannbaum T für (K_n, c)
(3) sei G der Eulersche Multigraph, der durch Verdopplung aller Kanten aus T entsteht
(4) bestimme eine Eulertour K für G
(5) wähle eine in K enthaltene Tour C
(6) **end algorithm**

* **Satz 31**:
  Algorithmus treeTour ist 1-approximativ und in $O(n^2)$.
* Beweis:
  $O(n^2)$ für (2), die Bestimmung eines minimalen Spannbaums.
  $O(n)$ für (3), die Erzeugung von G.
  $O(n)$ für (4), die Bestimmung einer Eulertour.
  $O(n)$ für (5), die Wahl einer TSP-Tour gemäß Lemma 30.
  Für diese Tour gilt $c(C) \leq 2c(T)$.
  Eine Tour hat mindestens die Länge $c(T)$.
  Also $c(C) \geq c(T)$.
  Zusammen ist $c(C)$ höchstens doppelt so groß wie das Gewicht einer optimalen Tour.



$c(T) = 186$
$c(C) = 286$

K = (Da,Dü,Ha,Be,Ha,Dü,Aa,Dü,Da,St,Fr,St,Nü,Mü,Nü,St,Da)

C = (Da,Be,Ha,Dü,Aa,Fr,Mü,Nü,St,Da)

Lineare Modelle und Schnittebenenverfahren

Lineare Modelle und Schnittebenenverfahren

* Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.

Lineare Modelle und Schnittebenenverfahren

* Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
* Ihre bahnbrechend neuen Ideen dabei:

Lineare Modelle und Schnittebenenverfahren

* Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
* Ihre bahnbrechend neuen Ideen dabei:
  * Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.

Lineare Modelle und Schnittebenenverfahren

* Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
* Ihre bahnbrechend neuen Ideen dabei:
  * Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  * Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.

Lineare Modelle und Schnittebenenverfahren

* Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
* Ihre bahnbrechend neuen Ideen dabei:
  * Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  * Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  * Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i, j\} \in E$ wird eine Variable $x_{i,j} \in [0, 1]$ eingeführt.

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.

Lineare Modelle und Schnittebenenverfahren

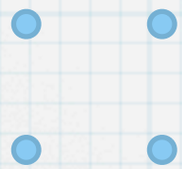
- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:

Lineare Modelle und Schnittebenenverfahren

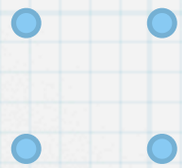
- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:



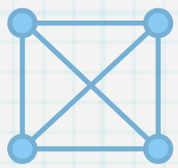
$x = (0,0,0,0,0,0)$

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:



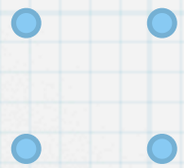
$x = (0,0,0,0,0,0)$



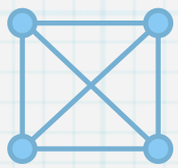
$x = (1,1,1,1,1,1)$

Lineare Modelle und Schnittebenenverfahren

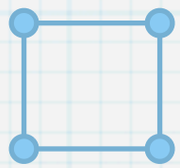
- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:



$x = (0,0,0,0,0,0)$



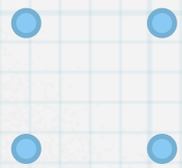
$x = (1,1,1,1,1,1)$



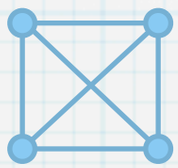
$x = (1,0,1,1,0,1)$

Lineare Modelle und Schnittebenenverfahren

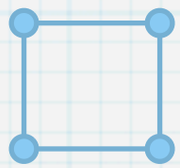
- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:



$x = (0,0,0,0,0,0)$



$x = (1,1,1,1,1,1)$



$x = (1,0,1,1,0,1)$

- Nebenbedingungen:

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:

$x = (0,0,0,0,0,0)$

$x = (1,1,1,1,1,1)$

$x = (1,0,1,1,0,1)$

- Nebenbedingungen:
  - Triviale Ungleichungen: $0 \leq x_{i,j} \leq 1$ für alle $\{i,j\} \in E$.

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:

$x = (0,0,0,0,0,0)$

$x = (1,1,1,1,1,1)$

$x = (1,0,1,1,0,1)$

- Nebenbedingungen:
  - Triviale Ungleichungen: $0 \leq x_{i,j} \leq 1$ für alle $\{i,j\} \in E$.
  - Gradbedingung: $\sum_{j:\{i,j\}\in E} x_{i,j} = 2$ für alle $i \in V$.

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:

$x = (0,0,0,0,0,0)$

$x = (1,1,1,1,1,1)$

$x = (1,0,1,1,0,1)$

- Nebenbedingungen:
  - Triviale Ungleichungen: $0 \leq x_{i,j} \leq 1$ für alle $\{i,j\} \in E$.
  - Gradbedingung: $\sum_{j:\{i,j\}\in E} x_{i,j} = 2$ für alle $i \in V$.
- Zielfunktion: $z = \min \sum_{\{i,j\}\in E} c_{i,j} \cdot x_{i,j}$.

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:

$x = (0,0,0,0,0,0)$

$x = (1,1,1,1,1,1)$

$x = (1,0,1,1,0,1)$

- Nebenbedingungen:
  - Triviale Ungleichungen: $0 \leq x_{i,j} \leq 1$ für alle $\{i,j\} \in E$.
  - Gradbedingung: $\sum_{j:\{i,j\}\in E} x_{i,j} = 2$ für alle $i \in V$.
- Zielfunktion: $z = \min \sum_{\{i,j\}\in E} c_{i,j} \cdot x_{i,j}$.
- Bemerke: Dieses Modell ist ein lineares Programm. Daher kann es mittels Dantzigs Simplex-Algorithmus gelöst werden.

Lineare Modelle und Schnittebenenverfahren

- Der Durchbruch von 1954: Dantzig, Fulkerson, Johnson lösen ein 42-Städte-Problem.
- Ihre bahnbrechend neuen Ideen dabei:
  - Modellierung des Problems durch lineare Nebenbedingungen und Zielfunktion.
  - Bestimmung einer unteren Schranke mittels Dantzigs Simplex-Algorithmus.
  - Verbesserung der Schranke durch weitere Nebenbedingungen (Schnittebenen).
- Für jede Kanten $\{i,j\} \in E$ wird eine Variable $x_{i,j} \in [0,1]$ eingeführt.
  - Ist $x_{i,j} = 1$, so wird die Kante $\{i,j\}$ in der Lösung benutzt.
  - Ist $x_{i,j} = 0$, so wird die Kante $\{i,j\}$ nicht benutzt.
- Beispiele:

$x = (0,0,0,0,0,0)$

$x = (1,1,1,1,1,1)$

$x = (1,0,1,1,0,1)$

- Nebenbedingungen:
  - Triviale Ungleichungen: $0 \leq x_{i,j} \leq 1$ für alle $\{i,j\} \in E$.
  - Gradbedingung: $\sum_{j:\{i,j\}\in E} x_{i,j} = 2$ für alle $i \in V$.
- Zielfunktion: $z = \min \sum_{\{i,j\}\in E} c_{i,j} \cdot x_{i,j}$.
- Bemerke: Dieses Modell ist ein lineares Programm. Daher kann es mittels Dantzigs Simplex-Algorithmus gelöst werden.
- Allerdings ist das Modell noch nicht vollständig. Es fehlen noch die Kurzzyklus-Bedingungen.

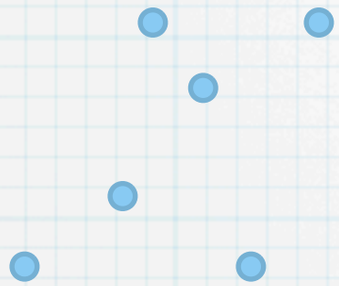
Die Kurzzyklus-Bedingung

Die Kurzzyklus-Bedingung

* Beispiel:

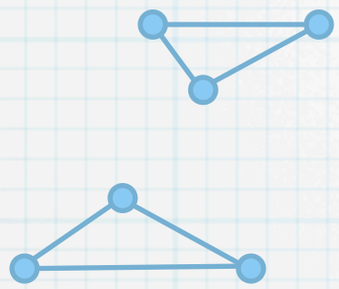
Die Kurzzyklus-Bedingung

* Beispiel:



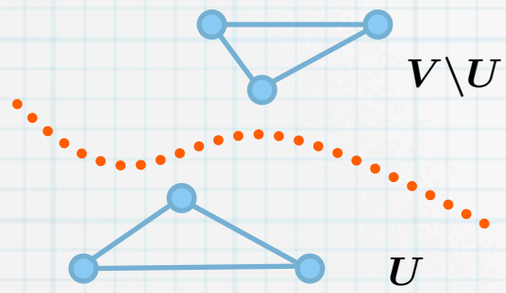
Die Kurzzyklus-Bedingung

* Beispiel:



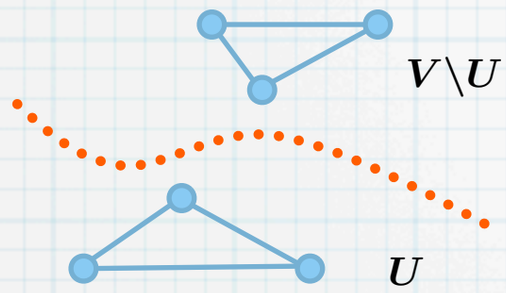
Die Kurzzyklus-Bedingung

* Beispiel:



Die Kurzzyklus-Bedingung

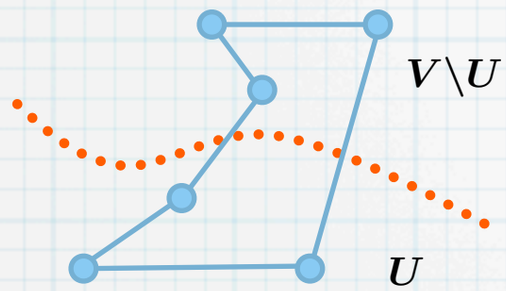
* Beispiel:



* Jede Rundreise erfüllt die sog. Kurzzyklus-Bedingung: $\forall\, U \subset V, U \neq \emptyset :\quad \sum_{\substack{\{i,j\} \in E \\ i \in U, j \notin U}} x_{i,j} \geq 2.$

Die Kurzzyklus-Bedingung

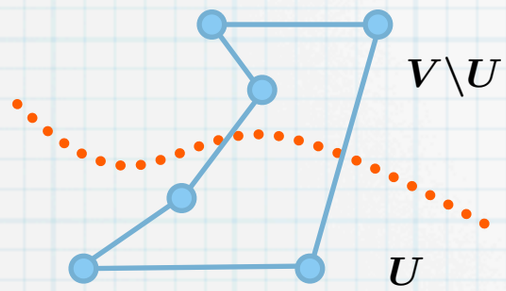
* Beispiel:



* Jede Rundreise erfüllt die sog. Kurzzyklus-Bedingung: $\forall\, U \subset V, U \neq \emptyset :\quad \sum_{\substack{\{i,j\} \in E \\ i \in U, j \notin U}} x_{i,j} \geq 2.$

Die Kurzzyklus-Bedingung

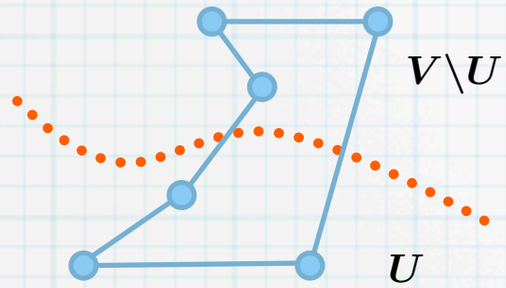
* Beispiel:



* Jede Rundreise erfüllt die sog. Kurzzyklus-Bedingung: $\forall\, U \subset V, U \neq \emptyset :\quad \sum_{\substack{\{i,j\} \in E \\ i \in U, j \notin U}} x_{i,j} \geq 2.$
* Da es exponentiell viele solche Ungleichungen gibt (in der Anzahl der Knoten), können nicht alle gleichzeitig zur Formulierung hinzugefügt werden.

Die Kurzzyklus-Bedingung

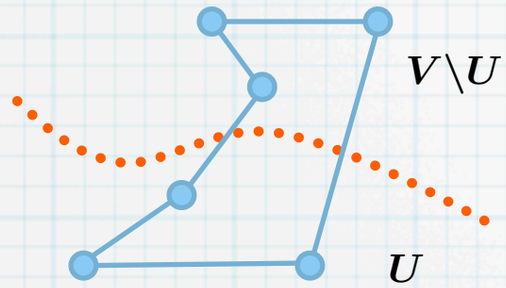
* Beispiel:



* Jede Rundreise erfüllt die sog. Kurzzyklus-Bedingung: $\forall\, U \subset V, U \neq \emptyset :\quad \sum_{\substack{\{i,j\} \in E \\ i \in U, j \notin U}} x_{i,j} \geq 2.$
* Da es exponentiell viele solche Ungleichungen gibt (in der Anzahl der Knoten), können nicht alle gleichzeitig zur Formulierung hinzugefügt werden.
* Daher löst man das Problem iterativ, in dem man immer nur die jeweils verletzte Ungleichung erkennt und in die Formulierung aufnimmt.

Die Kurzzyklus-Bedingung

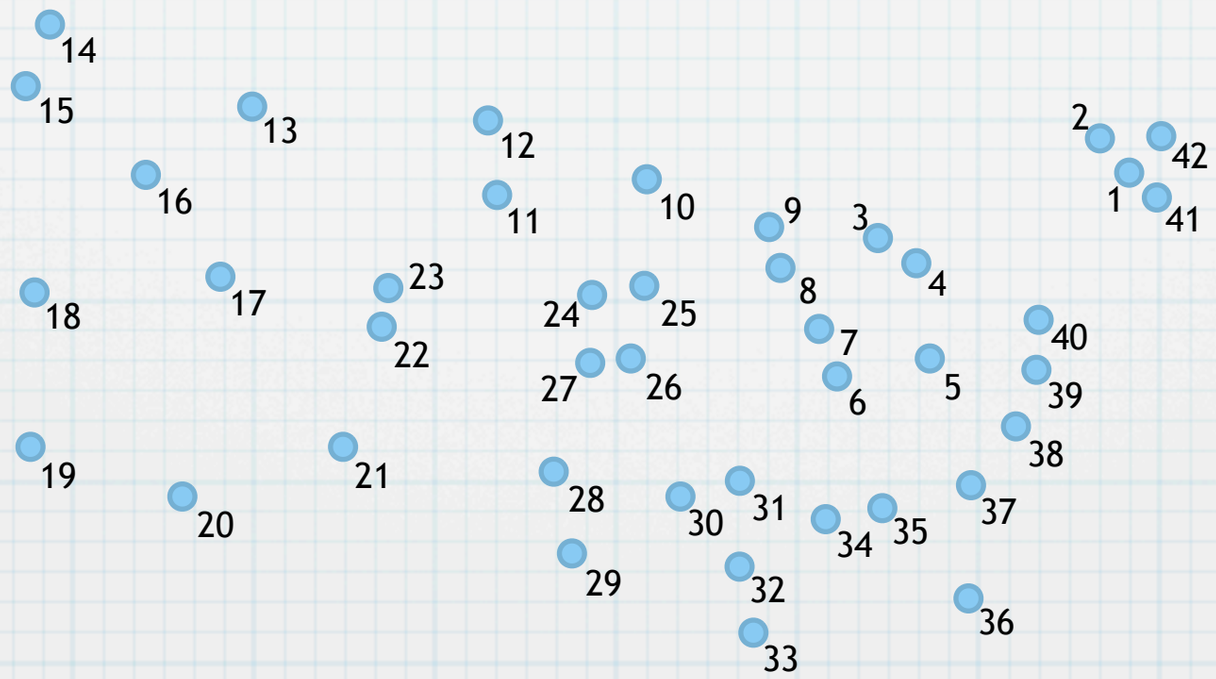
* Beispiel:



* Jede Rundreise erfüllt die sog. Kurzzyklus-Bedingung: $\forall\, U \subset V, U \neq \emptyset :\quad \sum_{\substack{\{i,j\} \in E \\ i \in U, j \notin U}} x_{i,j} \geq 2.$
* Da es exponentiell viele solche Ungleichungen gibt (in der Anzahl der Knoten), können nicht alle gleichzeitig zur Formulierung hinzugefügt werden.
* Daher löst man das Problem iterativ, in dem man immer nur die jeweils verletzte Ungleichung erkennt und in die Formulierung aufnimmt.
* **Definition 32**:
  Gegeben sei eine Lösung x^* eines linearen Programms und eine Menge von Ungleichungen $\mathcal{C}$. Das **Separierungproblem** besteht darin, zu entscheiden, ob x^* alle Ungleichungen von $\mathcal{C}$ erfüllt, oder, falls es in $\mathcal{C}$ eine verletzte Ungleichung gibt, diese zu bestimmen.

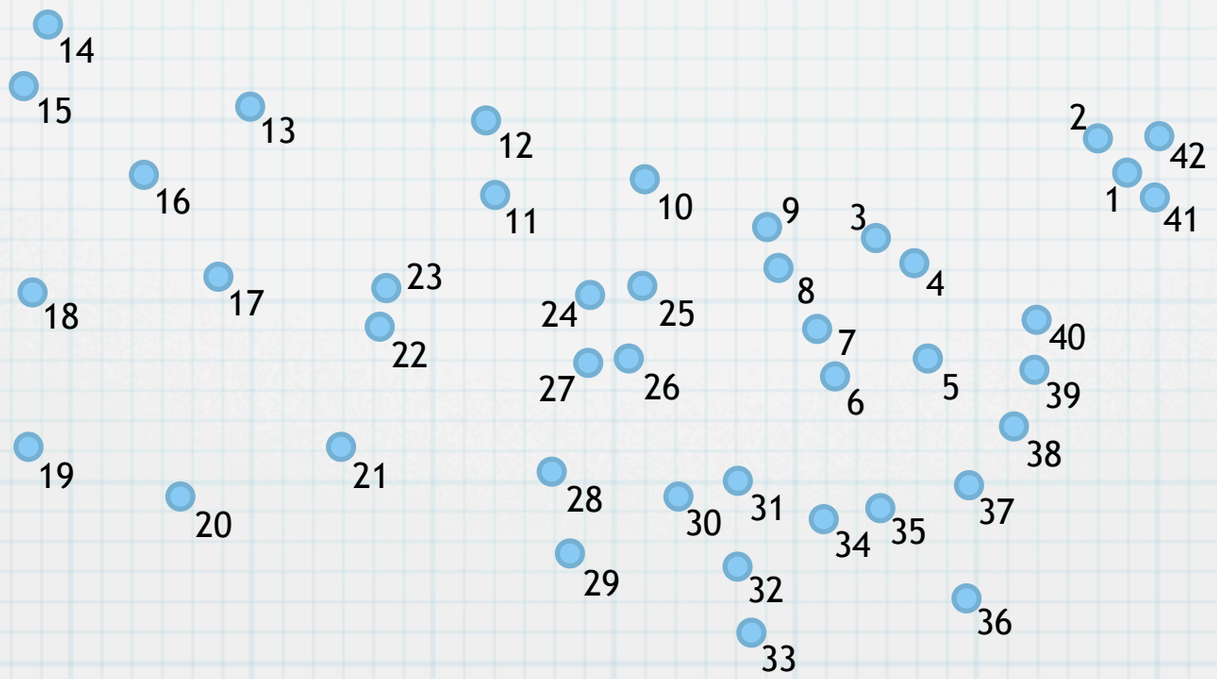
Die Lösung des 42-Städte-Problems

Die Lösung des 42-Städte-Problems



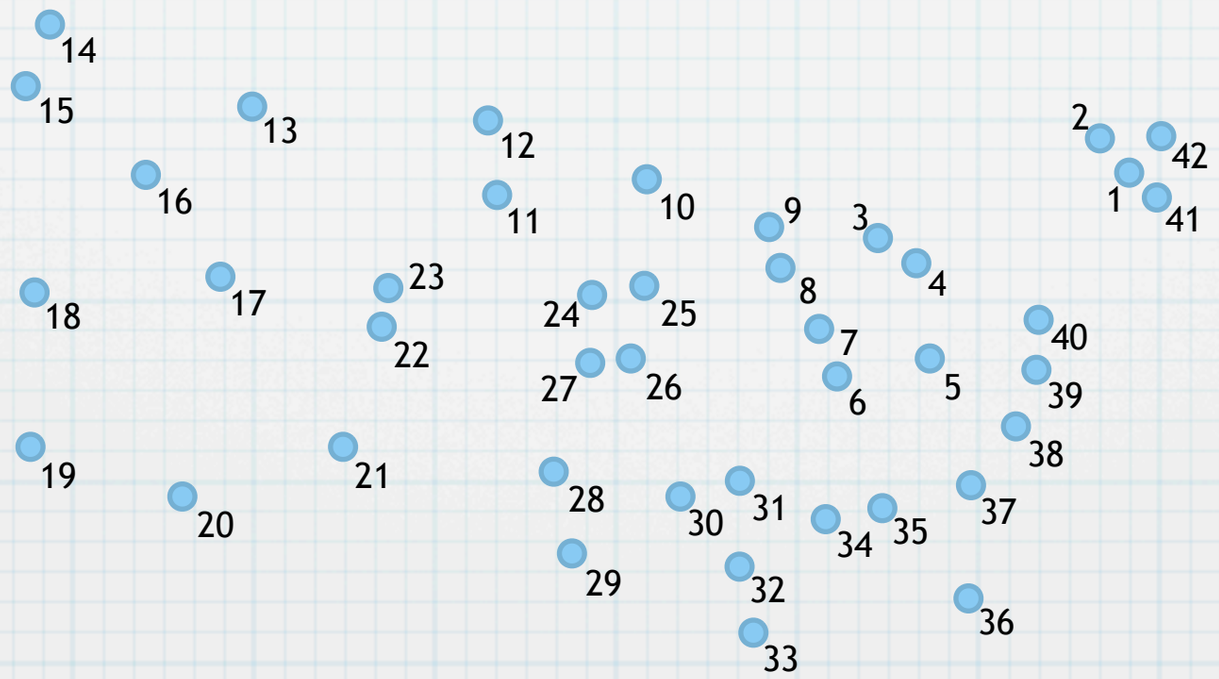
Die Lösung des 42-Städte-Problems

* Zielfunktionswert (=untere Schranke):



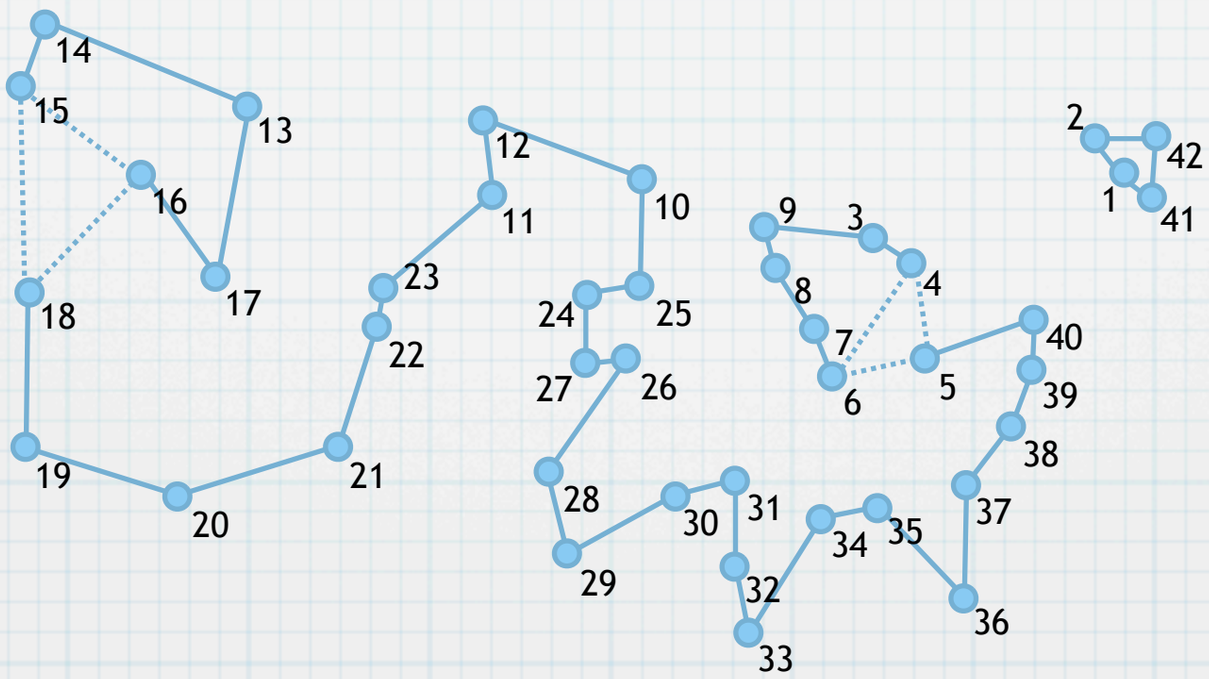
Die Lösung des 42-Städte-Problems

* Zielfunktionswert (=untere Schranke):
  * Keine Kurzzyklen: $z = 641$



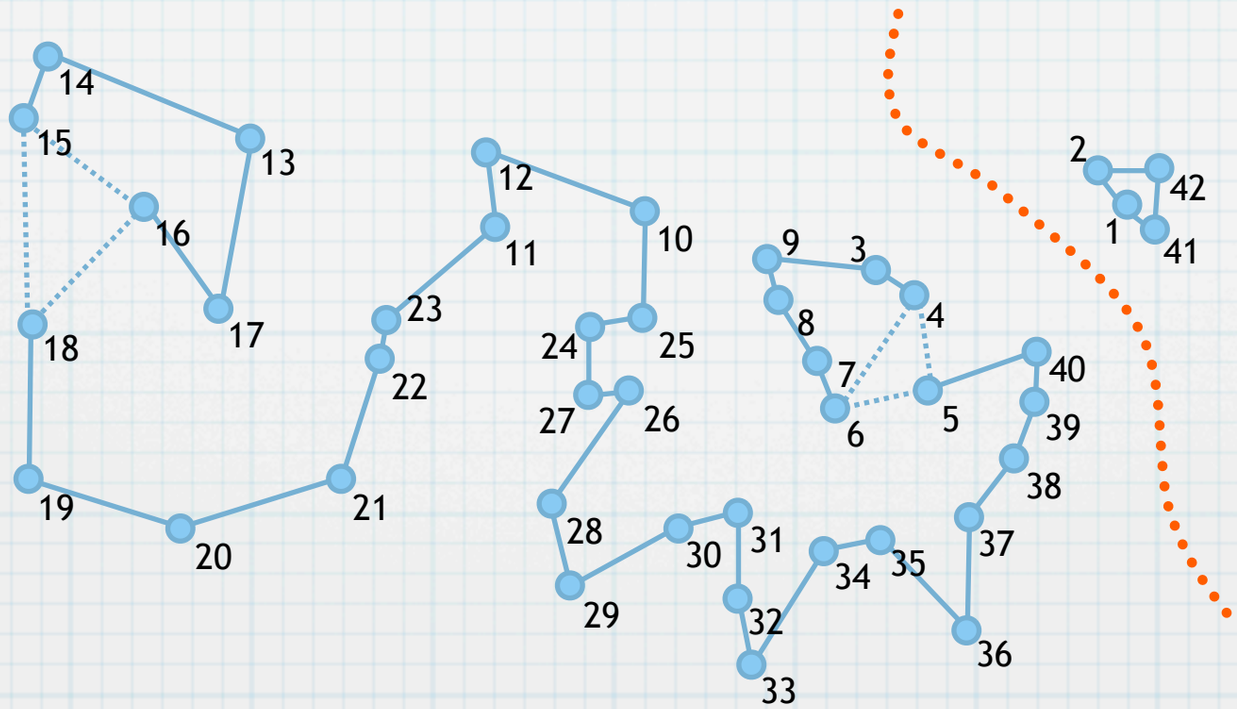
Die Lösung des 42-Städte-Problems

* Zielfunktionswert (=untere Schranke):
  * Keine Kurzzyklen: $z = 641$



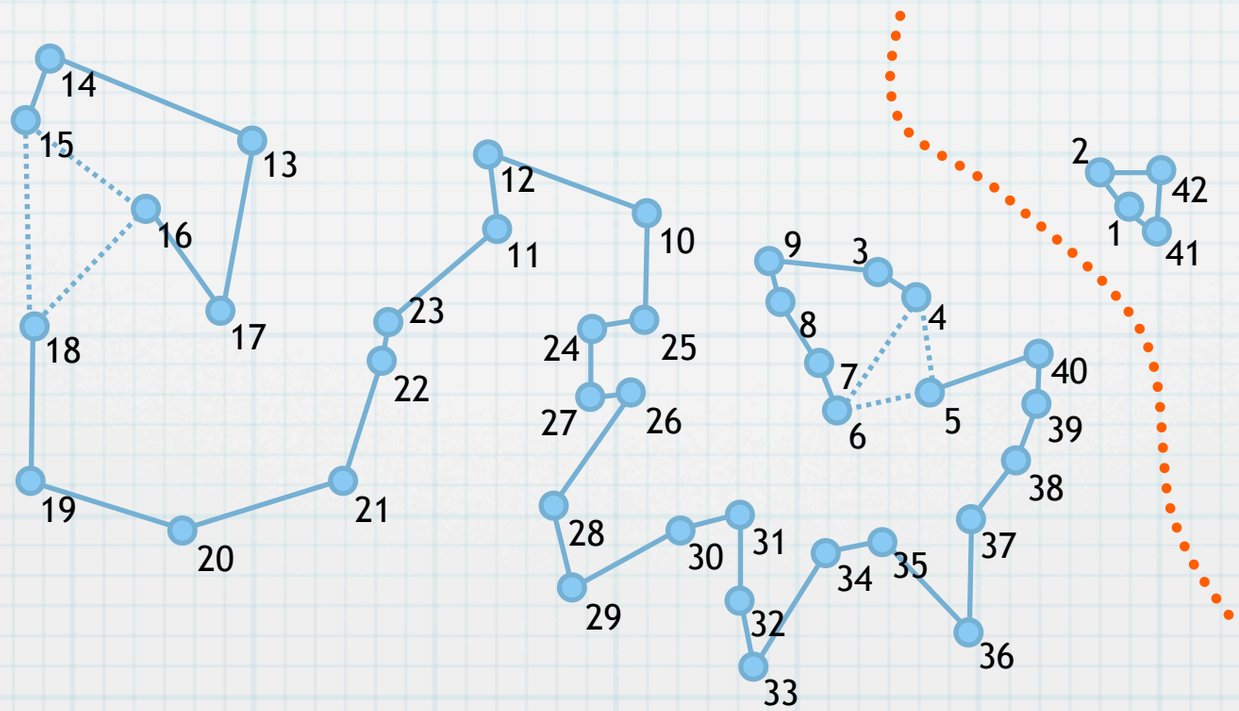
Die Lösung des 42-Städte-Problems

* Zielfunktionswert (=untere Schranke):
  * Keine Kurzzyklen: $z = 641$



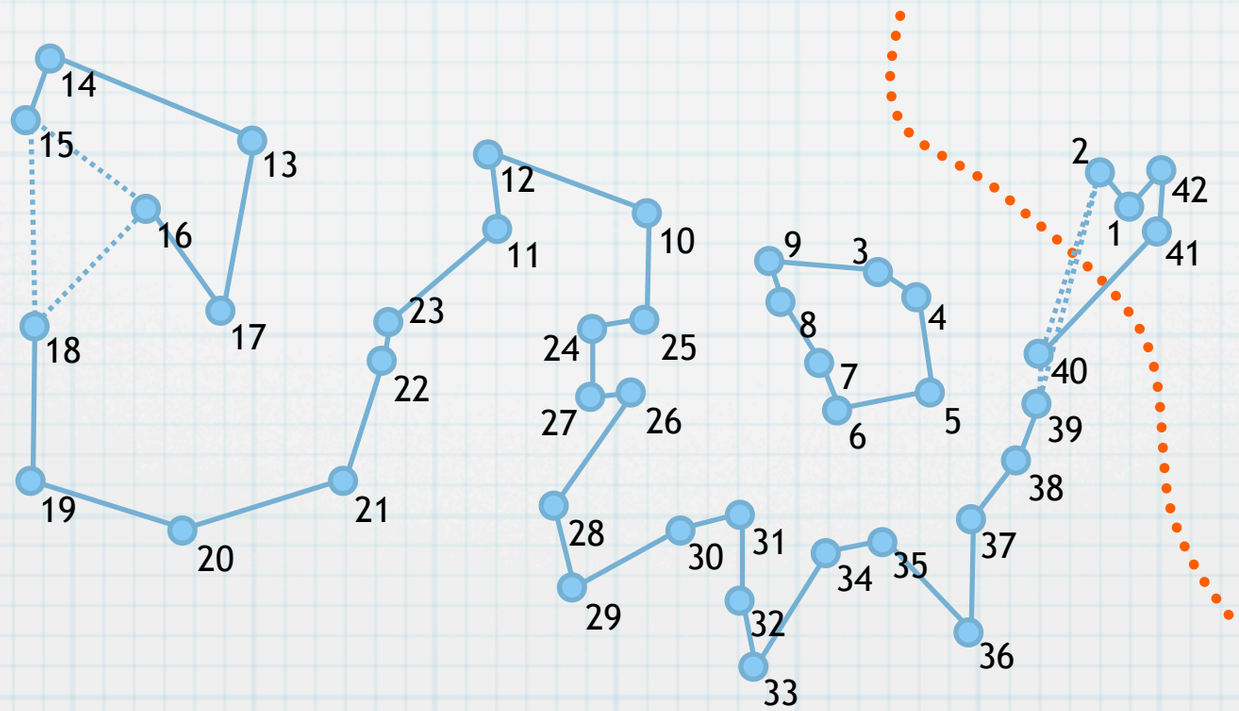
Die Lösung des 42-Städte-Problems

* Zielfunktionswert (=untere Schranke):
  * Keine Kurzzyklen: $z = 641$
  * 1 Kurzzyklus: $z = 676$



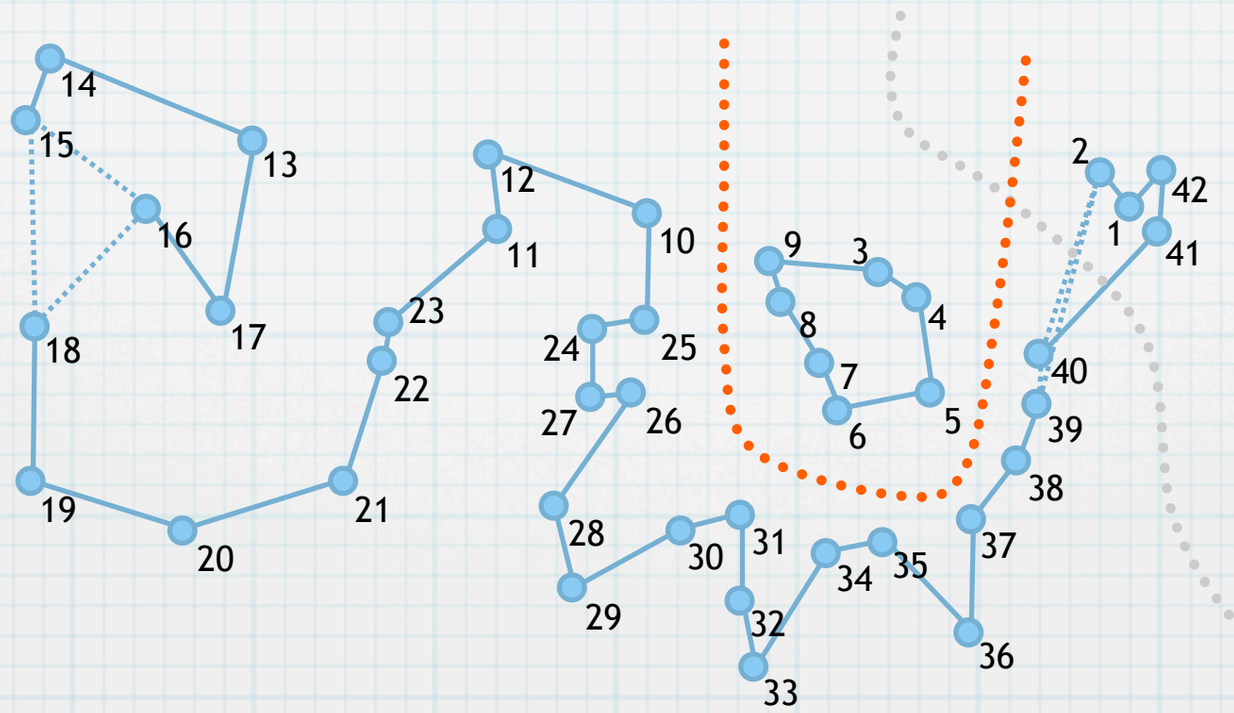
Die Lösung des 42-Städte-Problems

* Zielfunktionswert (=untere Schranke):
  * Keine Kurzzyklen: $z = 641$
  * 1 Kurzzyklus: $z = 676$



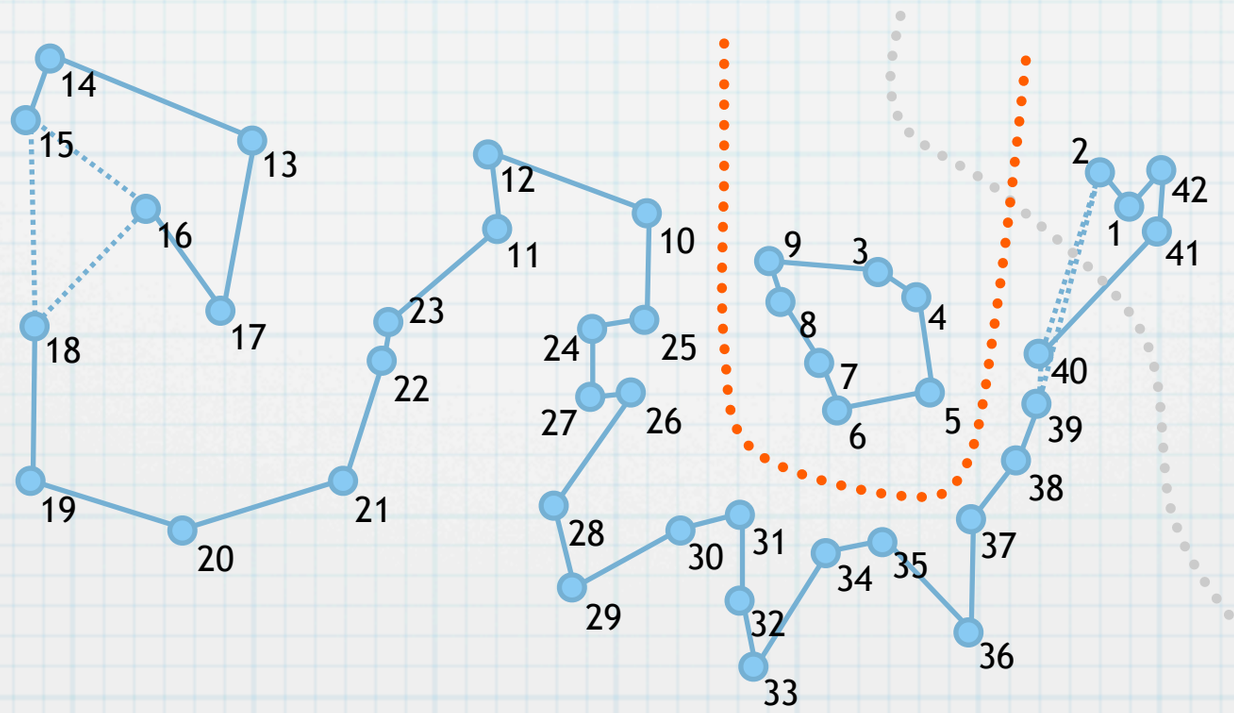
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$



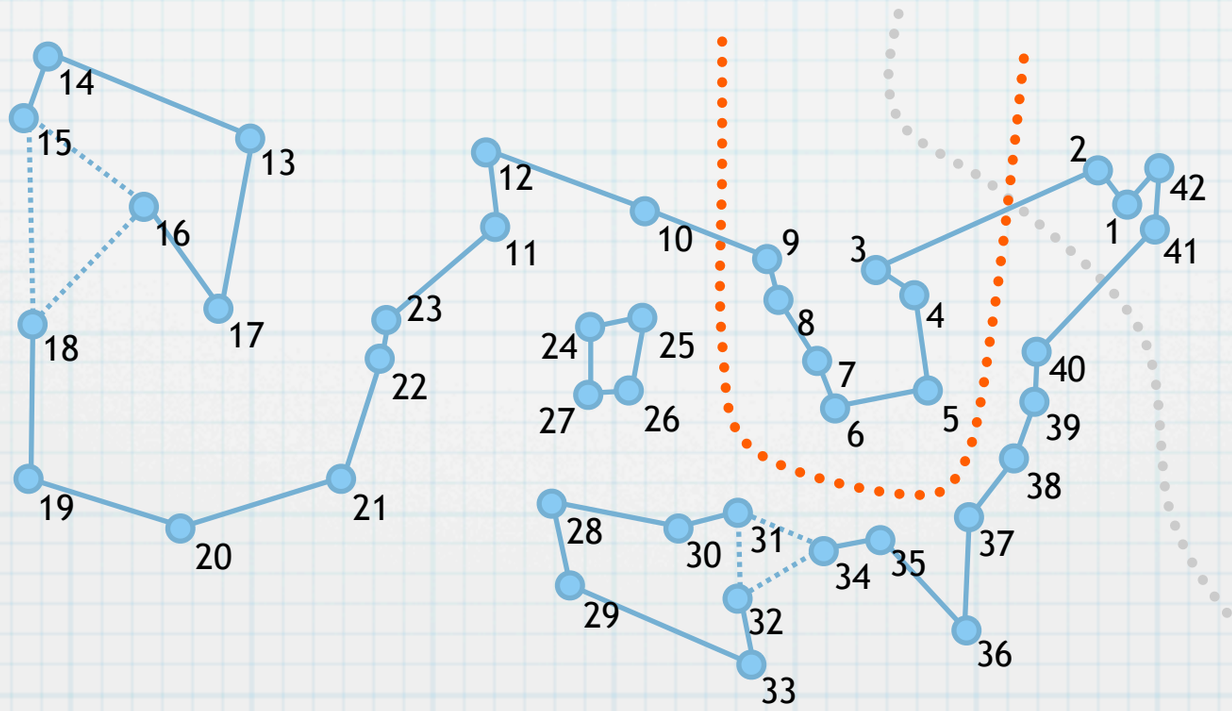
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$



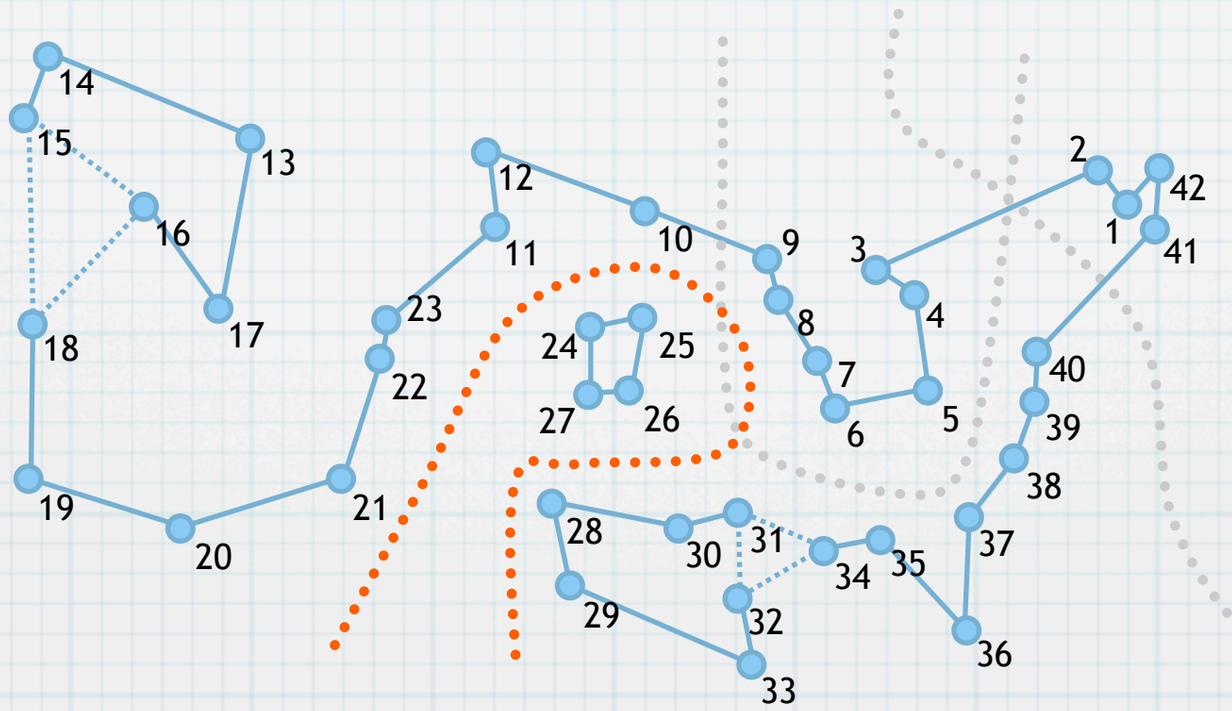
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$



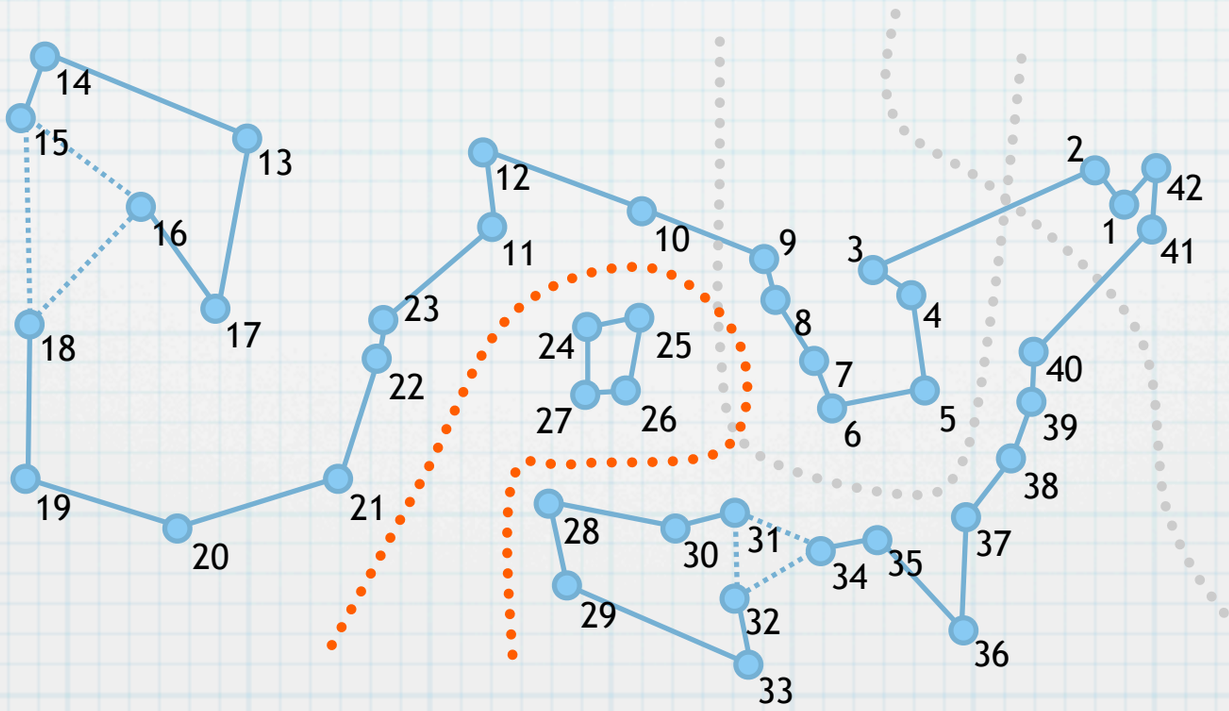
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$



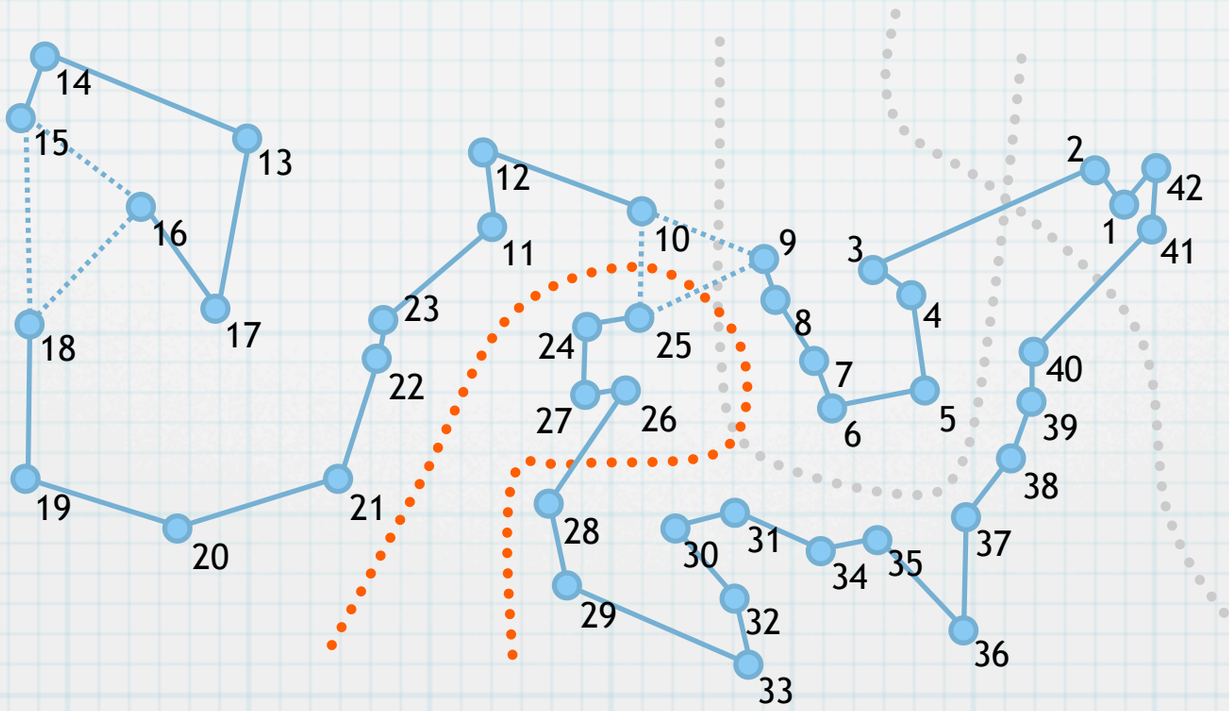
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$



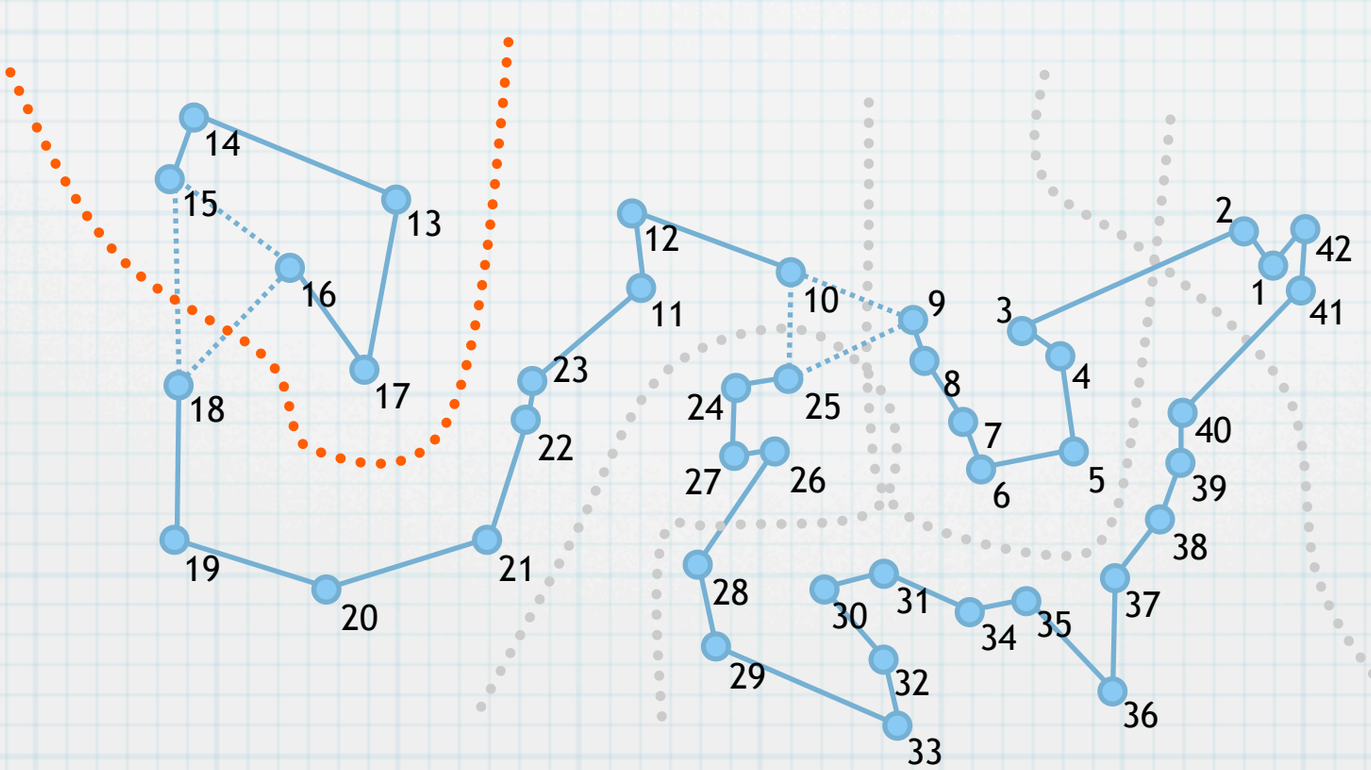
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$



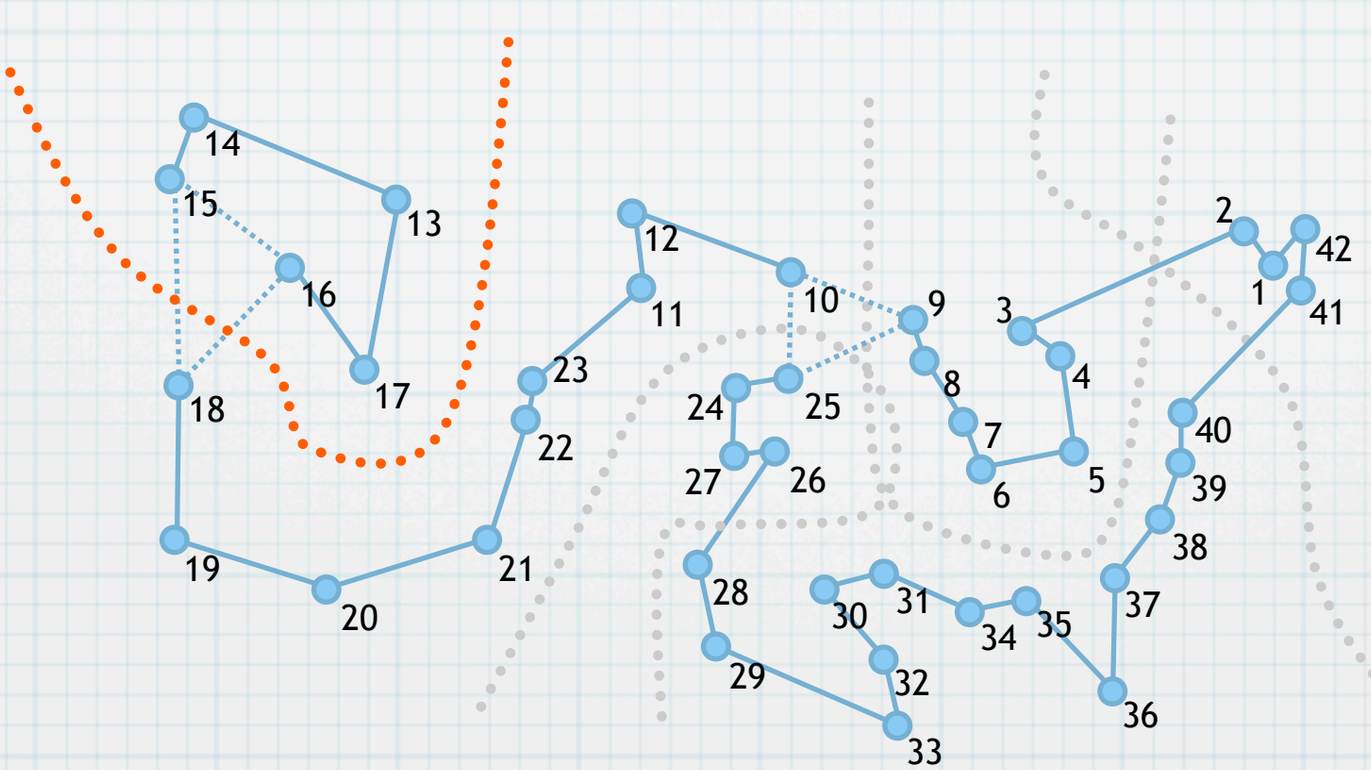
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$



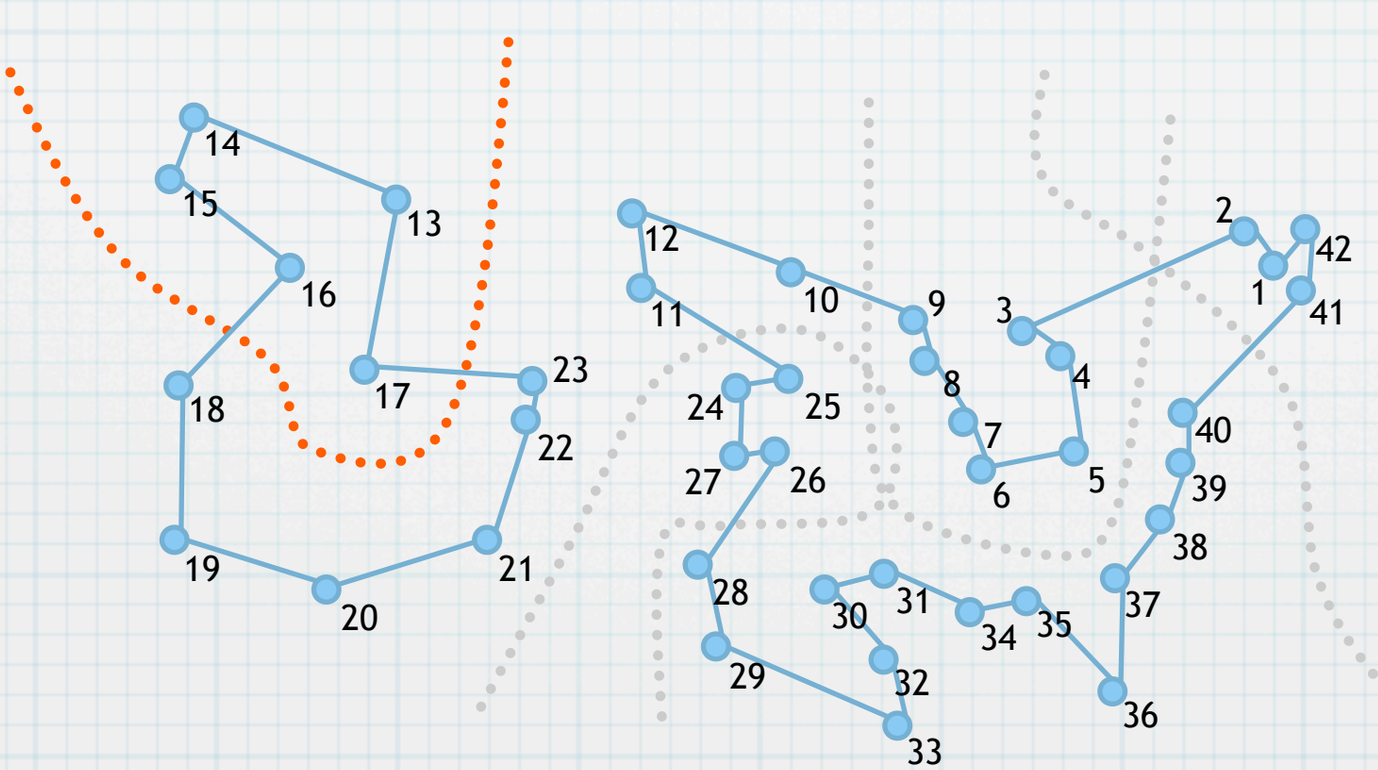
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$



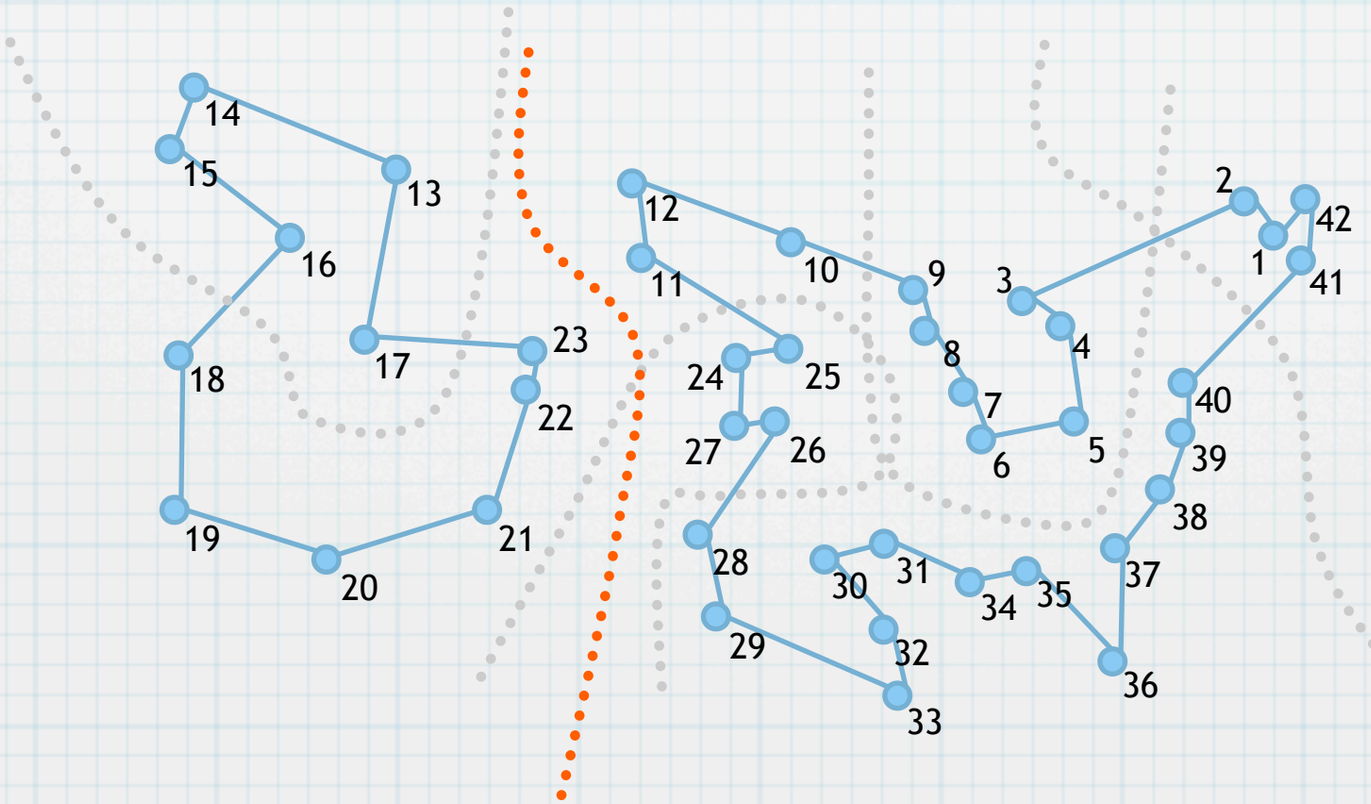
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$



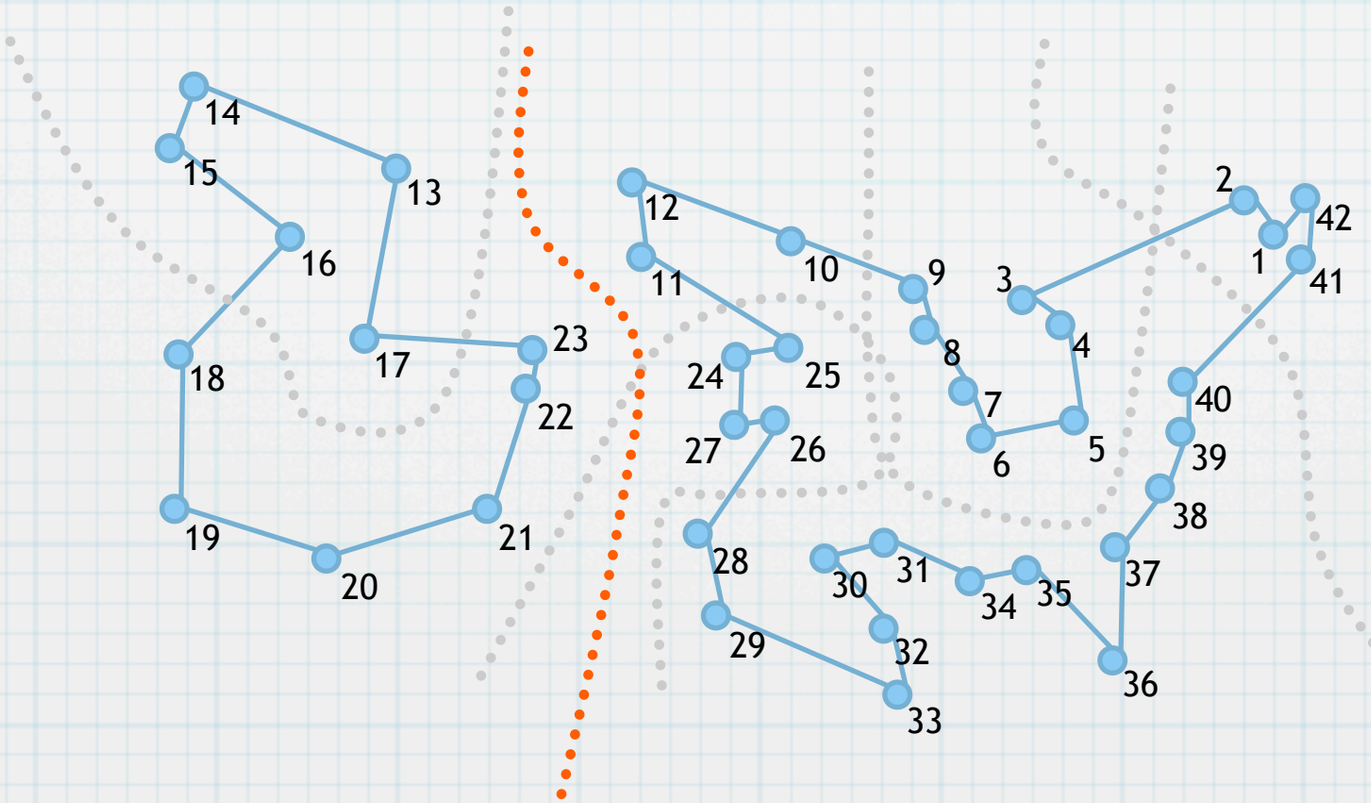
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$



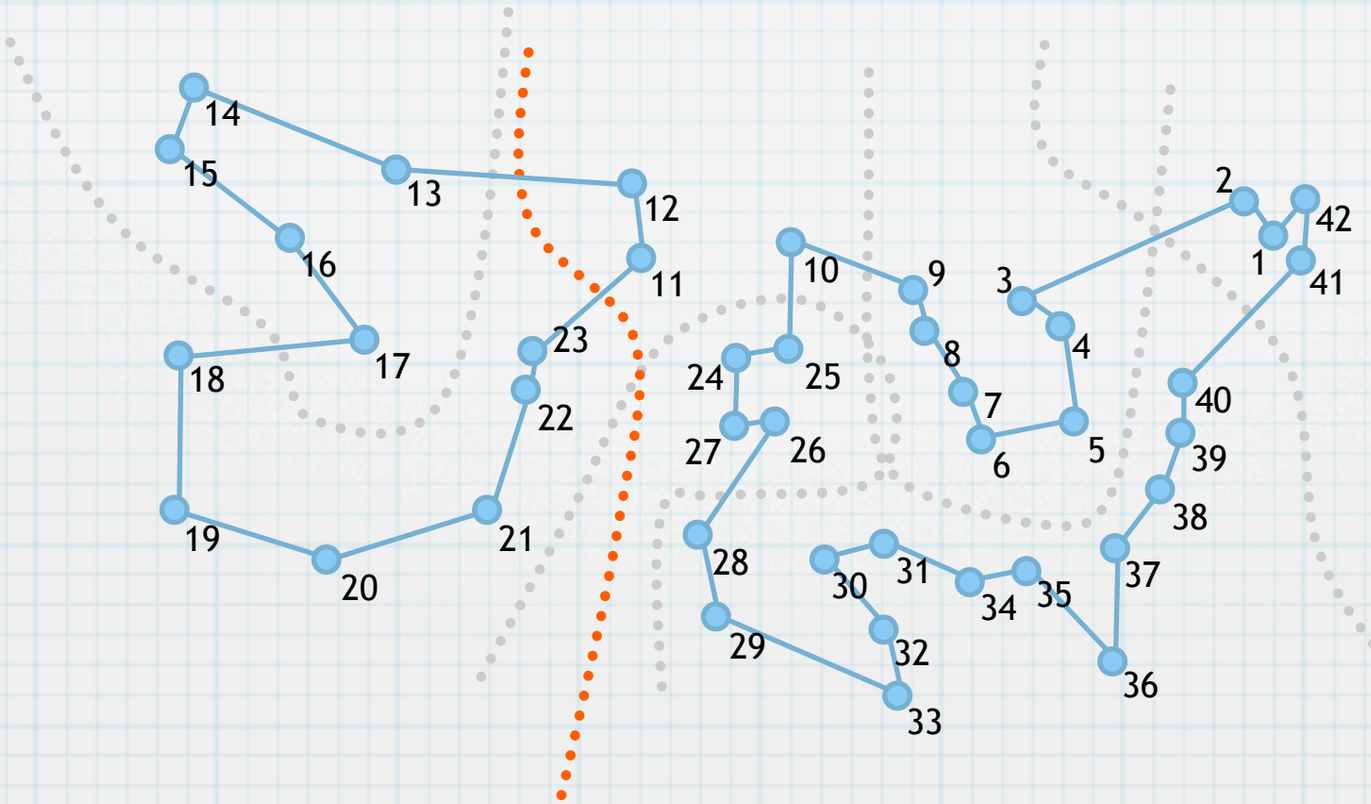
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$
  - 5 Kurzzyklen: $z = 688$



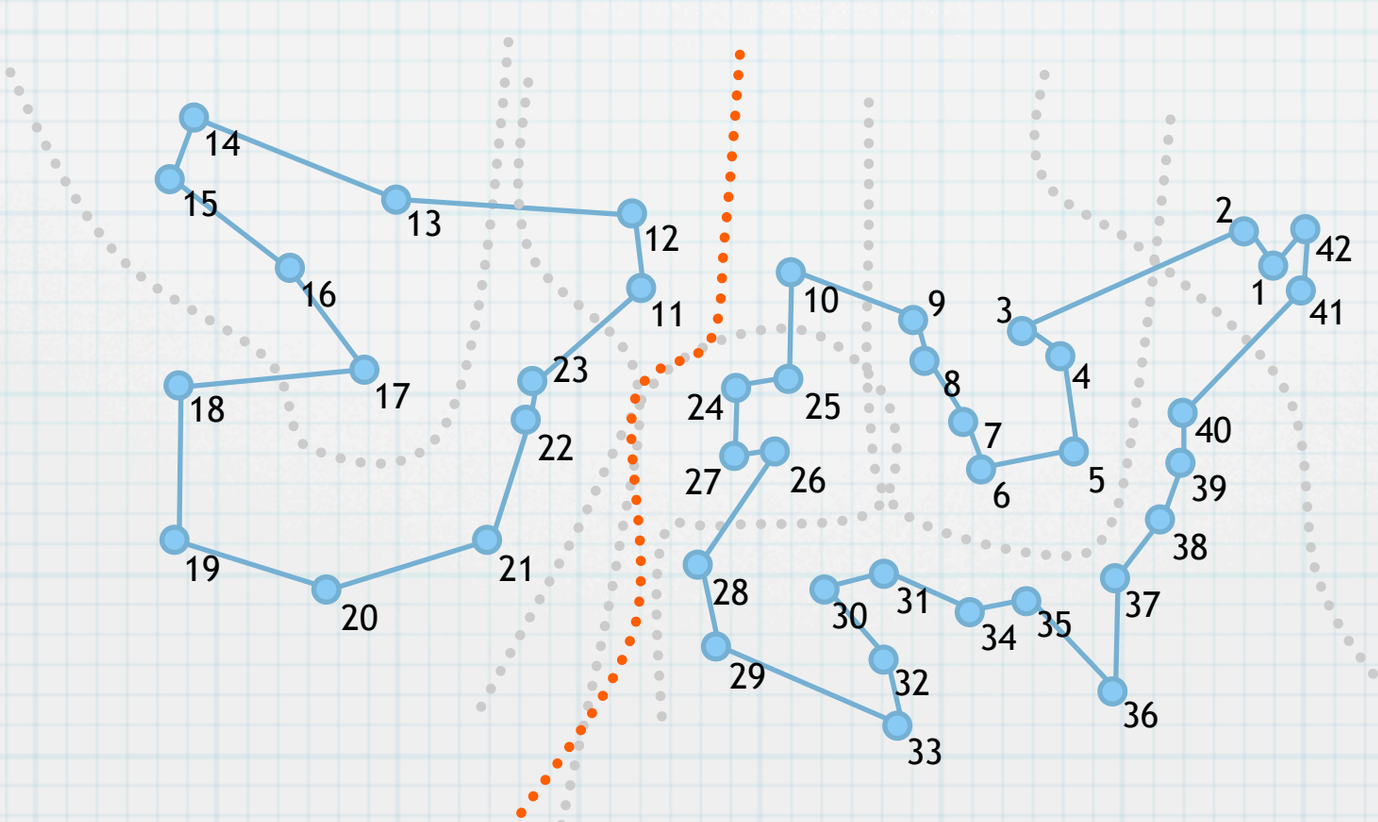
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$
  - 5 Kurzzyklen: $z = 688$



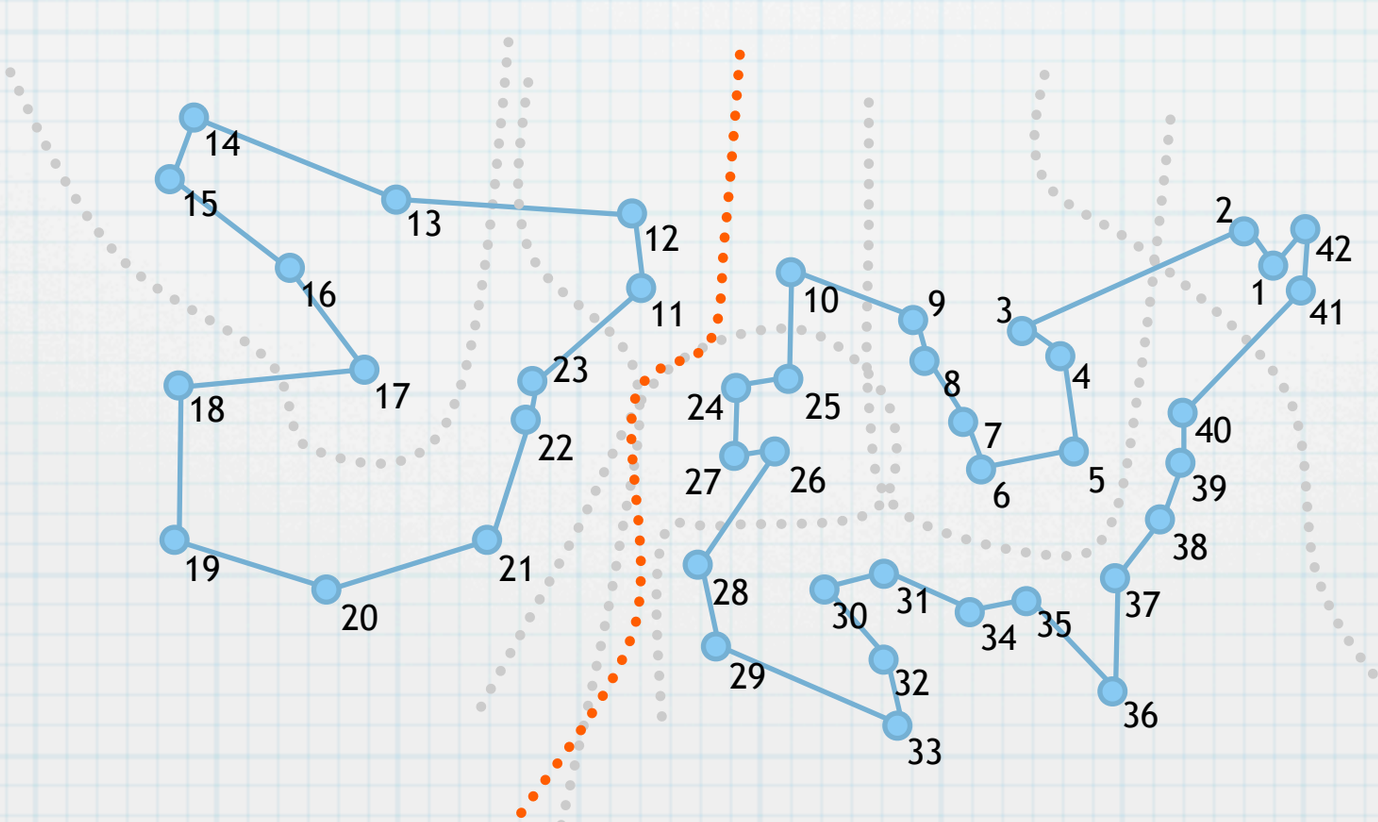
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$
  - 5 Kurzzyklen: $z = 688$



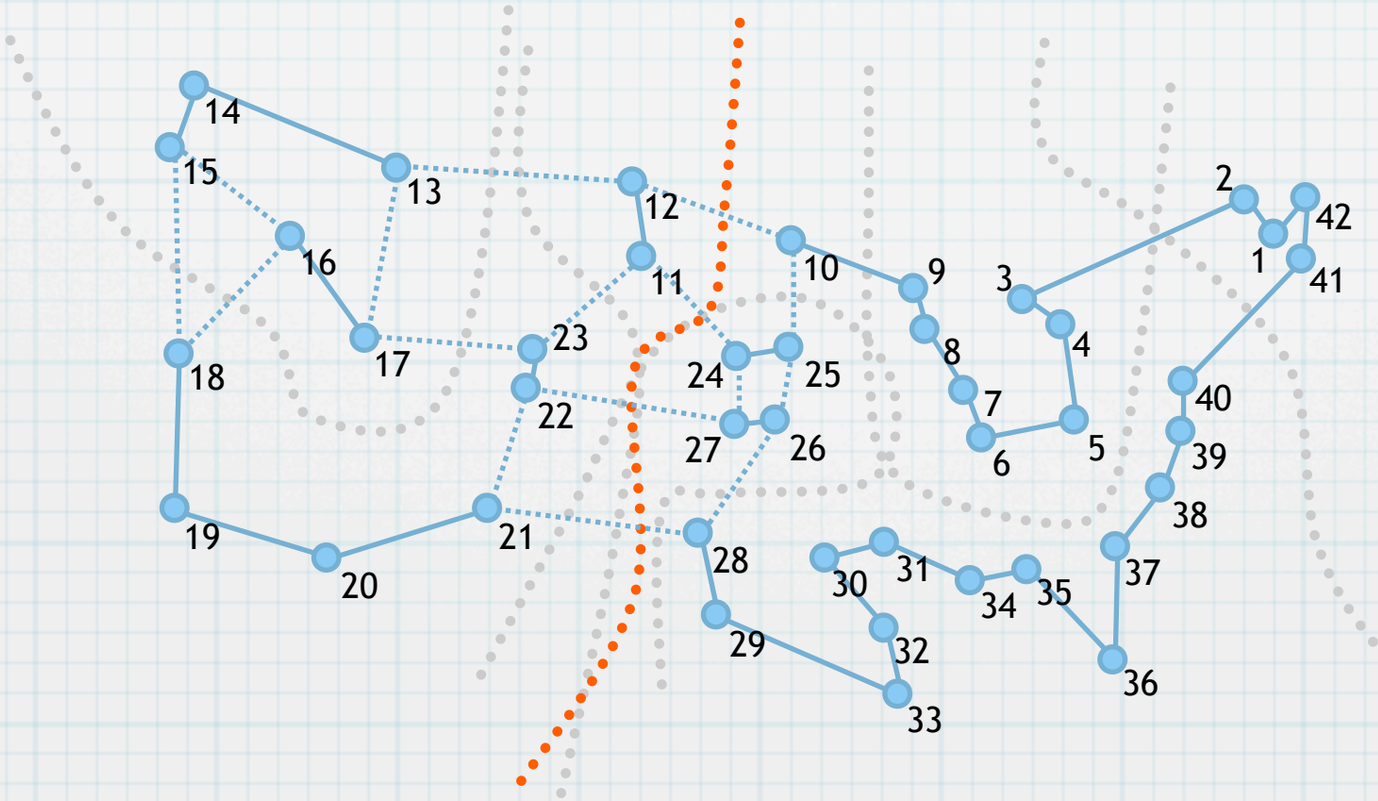
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$
  - 5 Kurzzyklen: $z = 688$
  - 6 Kurzzyklen: $z = 697$



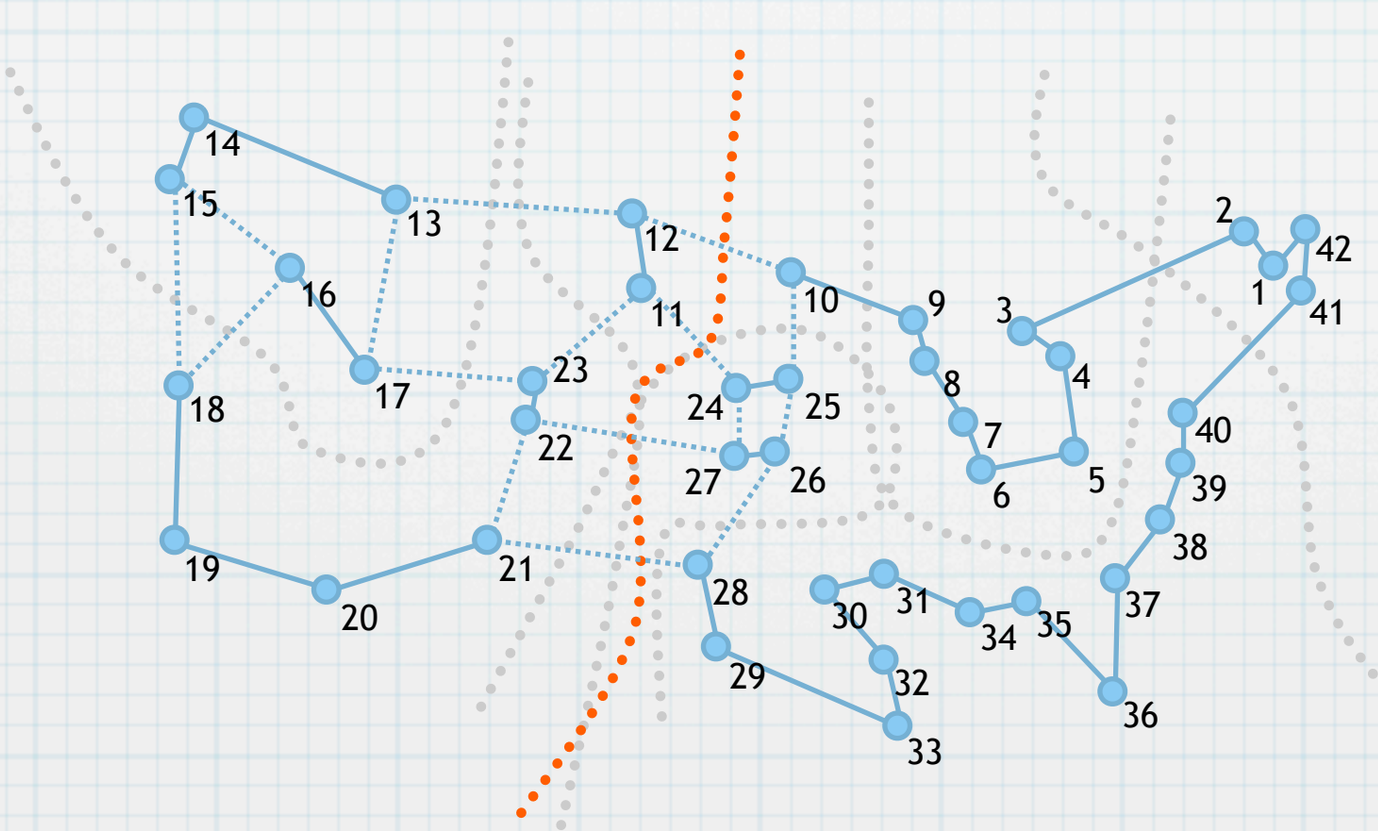
Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$
  - 5 Kurzzyklen: $z = 688$
  - 6 Kurzzyklen: $z = 697$



Die Lösung des 42-Städte-Problems

- Zielfunktionswert (=untere Schranke):
  - Keine Kurzzyklen: $z = 641$
  - 1 Kurzzyklus: $z = 676$
  - 2 Kurzzyklen: $z = 681$
  - 3 Kurzzyklen: $z = 682,5$
  - 4 Kurzzyklen: $z = 686$
  - 5 Kurzzyklen: $z = 688$
  - 6 Kurzzyklen: $z = 697$
- Keine weiteren verletzten Kurzzyklus-Ungleichungen mehr.



Zwei weitere Ungleichungen...

Zwei weitere Ungleichungen...

* „... due to Glicksberg (RAND Corp.)...“

Zwei weitere Ungleichungen...

* „... due to Glicksberg (RAND Corp.)...“

$$\sum_{\substack{i,j \in \{15,16,18,19\} \\ i<j}} x_{i,j} + 2x_{14,15} + x_{16,17} + x_{19,20} \leq 6$$

Zwei weitere Ungleichungen...

* „... due to Glicksberg (RAND Corp.)...“

$$\sum_{\substack{i,j\in\{15,16,18,19\}\\ i<j}} x_{i,j} + 2x_{14,15} + x_{16,17} + x_{19,20} \leq 6$$

$$\sum_{1\leq i,j\leq 42} a_{i,j}\cdot x_{i,j} \leq 42, \qquad a_{i,j} = \begin{cases} 2, & \{i,j\} = \{22,23\}, \\ 0, & \{i,j\} = \{25,26\}, \\ 0, & \text{entweder } i \text{ oder } j \in \{10,11,\ldots,28\}, x^*_{i,j} = 0, \\ 0, & \text{mind. } i \text{ oder } j \in \{10,21,25,\ldots,28\}, x^*_{i,j} = 0, \\ 1, & \text{sonst.} \end{cases}$$

Zwei weitere Ungleichungen...

* „... due to Glicksberg (RAND Corp.)...“

$$\sum_{\substack{i,j\in\{15,16,18,19\}\\ i<j}} x_{i,j} + 2x_{14,15} + x_{16,17} + x_{19,20} \leq 6$$

$$\sum_{1\leq i,j\leq 42} a_{i,j}\cdot x_{i,j} \leq 42, \qquad a_{i,j} = \begin{cases} 2, & \{i,j\} = \{22,23\}, \\ 0, & \{i,j\} = \{25,26\}, \\ 0, & \text{entweder } i \text{ oder } j \in \{10,11,\ldots,28\}, x^*_{i,j} = 0, \\ 0, & \text{mind. } i \text{ oder } j \in \{10,21,25,\ldots,28\}, x^*_{i,j} = 0, \\ 1, & \text{sonst.} \end{cases}$$

* ... liefern dann eine Optimallösung mit Zielfunktionswert $z = 699$.

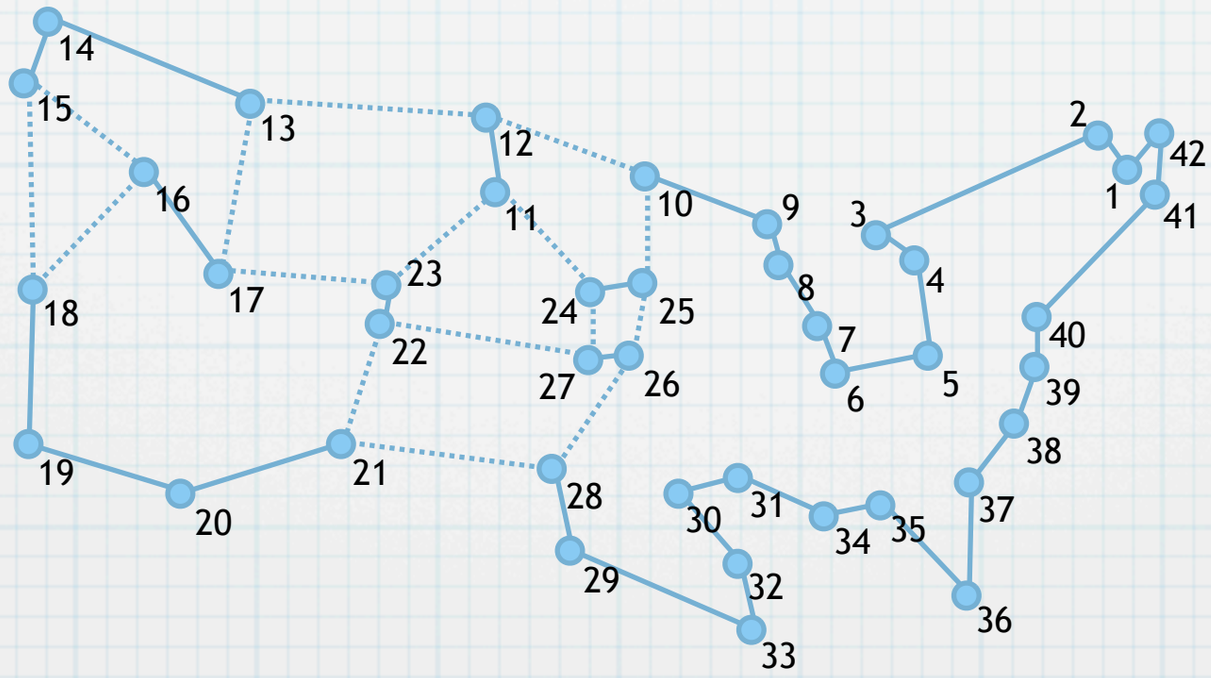
Zwei weitere Ungleichungen...

* „... due to Glicksberg (RAND Corp.)...“

$$\sum_{\substack{i,j\in\{15,16,18,19\}\\ i<j}} x_{i,j} + 2x_{14,15} + x_{16,17} + x_{19,20} \leq 6$$

$$\sum_{1\leq i,j\leq 42} a_{i,j} \cdot x_{i,j} \leq 42, \qquad a_{i,j} = \begin{cases} 2, & \{i,j\} = \{22,23\}, \\ 0, & \{i,j\} = \{25,26\}, \\ 0, & \text{entweder } i \text{ oder } j \in \{10,11,\ldots,28\}, x^*_{i,j} = 0, \\ 0, & \text{mind. } i \text{ oder } j \in \{10,21,25,\ldots,28\}, x^*_{i,j} = 0, \\ 1, & \text{sonst.} \end{cases}$$

* ... liefern dann eine Optimallösung mit Zielfunktionswert $z = 699$.



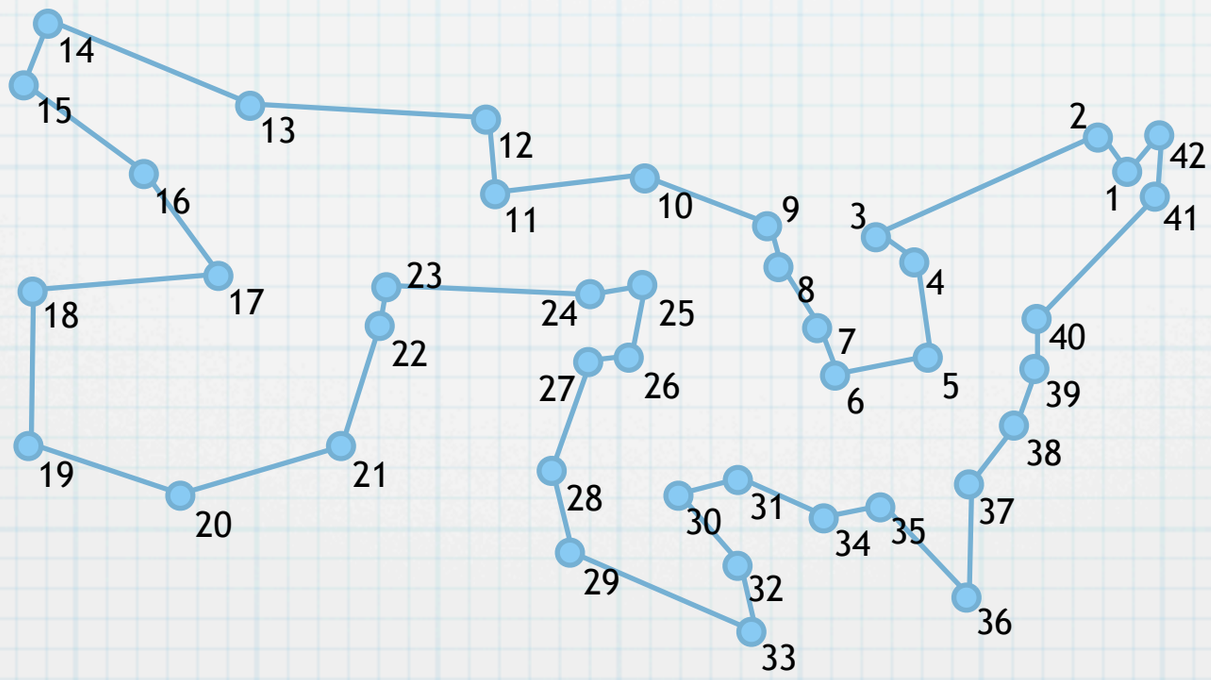
Zwei weitere Ungleichungen...

* „... due to Glicksberg (RAND Corp.)...“

$$\sum_{\substack{i,j\in\{15,16,18,19\}\\ i<j}} x_{i,j} + 2x_{14,15} + x_{16,17} + x_{19,20} \leq 6$$

$$\sum_{1\leq i,j\leq 42} a_{i,j} \cdot x_{i,j} \leq 42, \qquad a_{i,j} = \begin{cases} 2, & \{i,j\} = \{22,23\}, \\ 0, & \{i,j\} = \{25,26\}, \\ 0, & \text{entweder } i \text{ oder } j \in \{10,11,\ldots,28\}, x^*_{i,j} = 0, \\ 0, & \text{mind. } i \text{ oder } j \in \{10,21,25,\ldots,28\}, x^*_{i,j} = 0, \\ 1, & \text{sonst.} \end{cases}$$

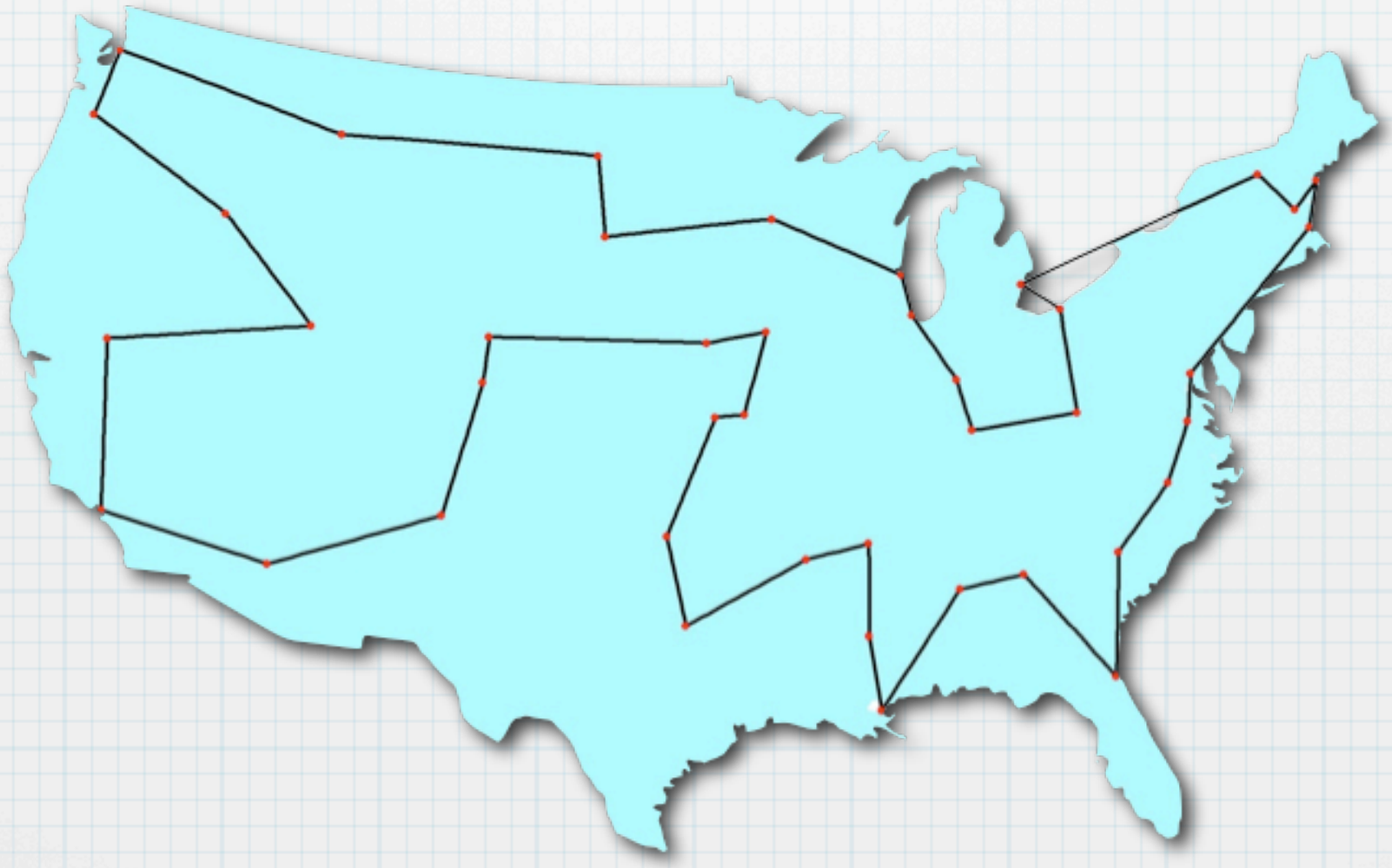
* ... liefern dann eine Optimallösung mit Zielfunktionswert $z = 699$.



Die TSP-Olympiade: Höher, Schneller, Weiter

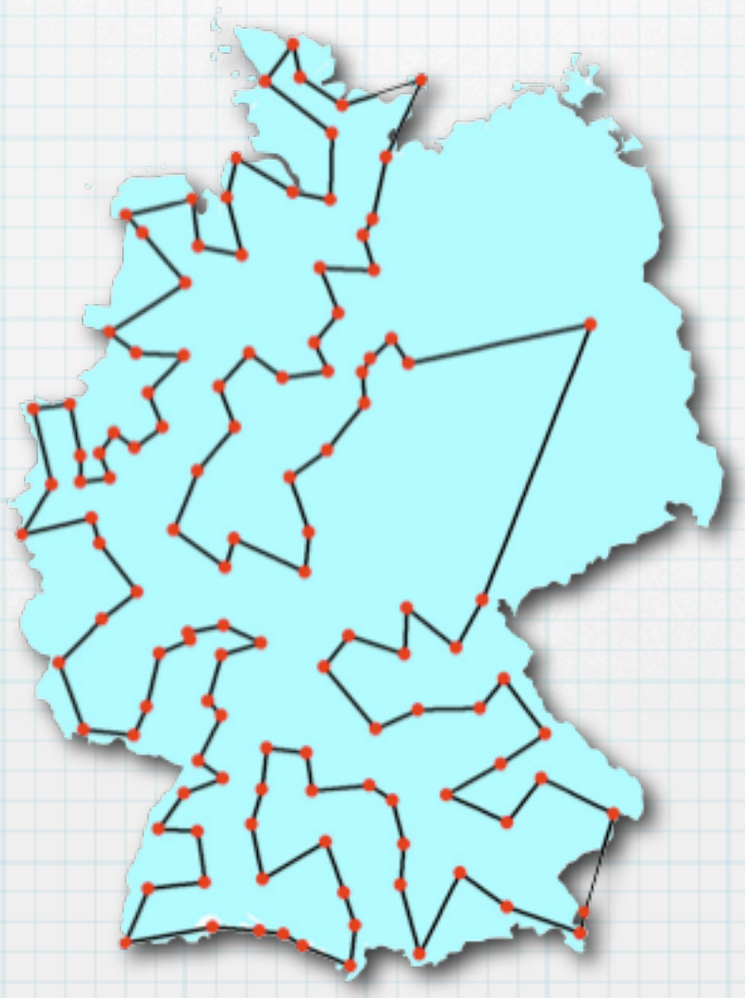
Die TSP-Olympiade: Höher, Schneller, Weiter

* Dantzig, Fulkerson, Johnson (1954): 42 Städte in den USA



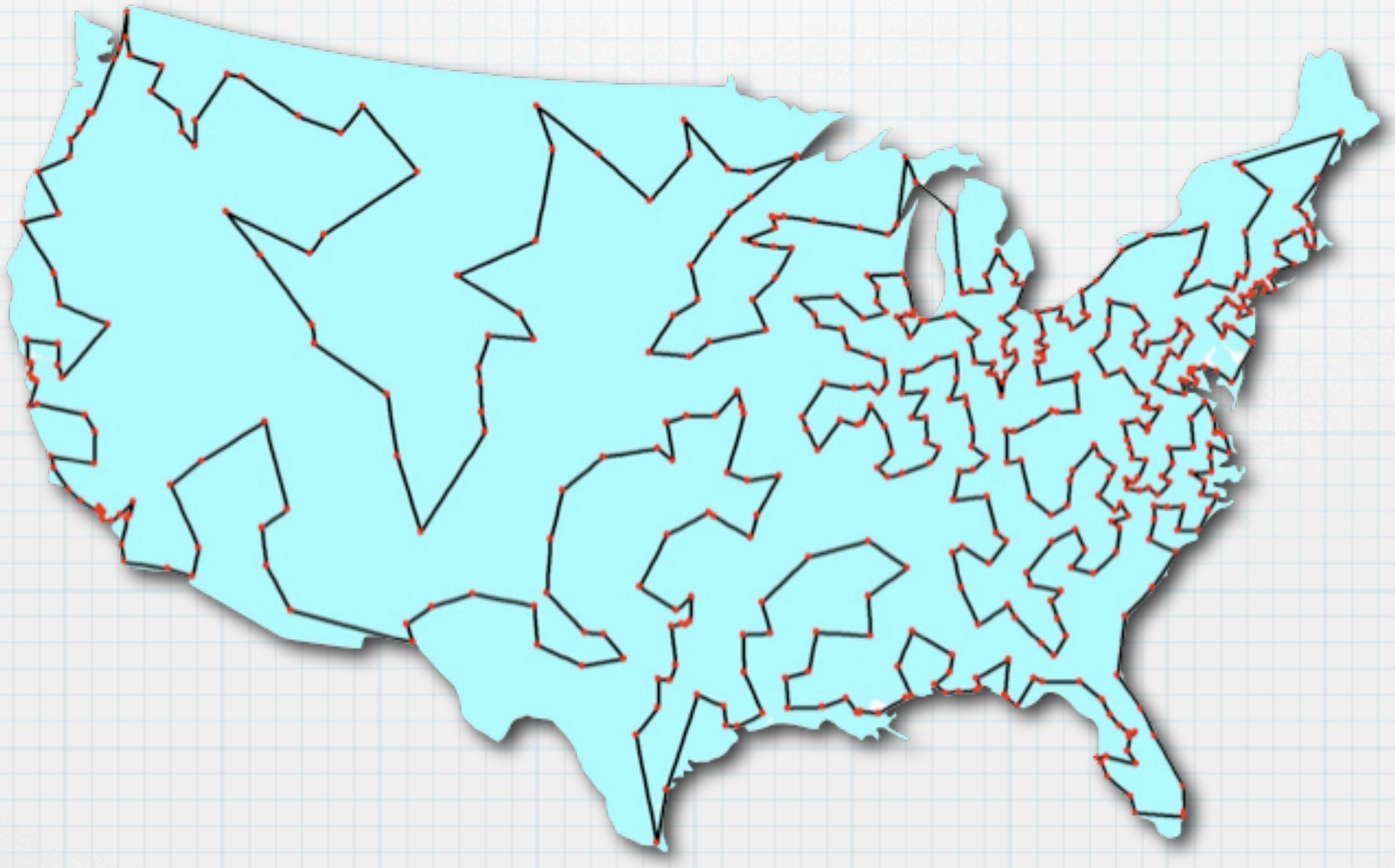
Die TSP-Olympiade: Höher, Schneller, Weiter

* Dantzig, Fulkerson, Johnson (1954): 42 Städte in den USA
* Grötschel (1977): 120 Städte in (West-) Deutschland



Die TSP-Olympiade: Höher, Schneller, Weiter

* Dantzig, Fulkerson, Johnson (1954): 42 Städte in den USA
* Grötschel (1977): 120 Städte in (West-) Deutschland
* Padberg, Rinaldi (1987): 532 AT&T Schaltstellen in den USA

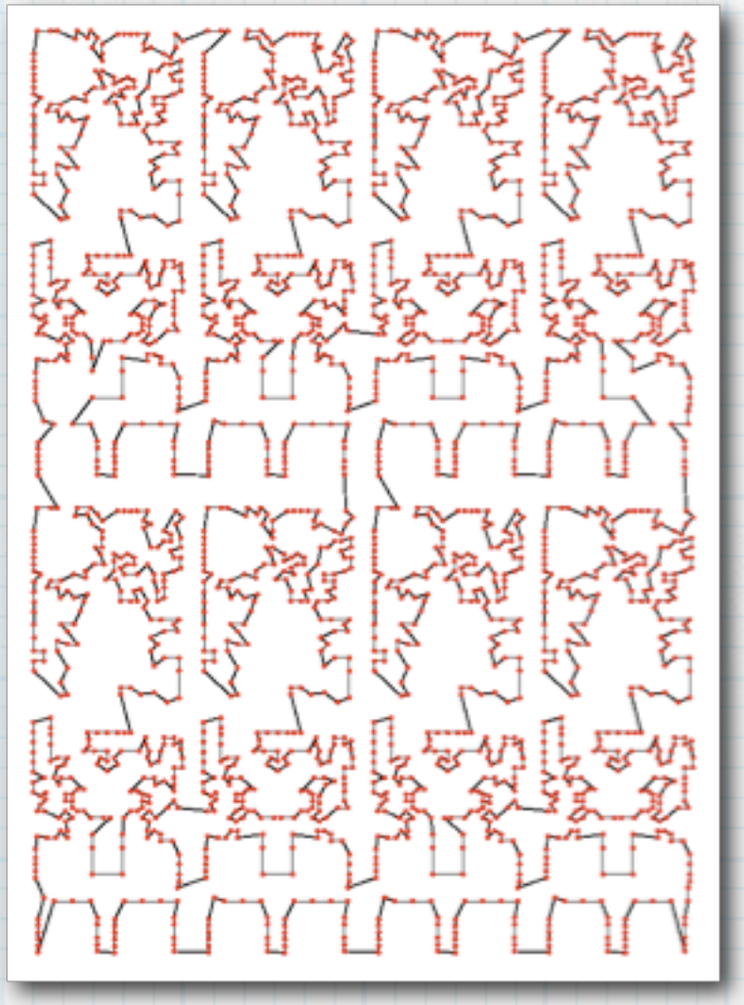


Die TSP-Olympiade: Höher, Schneller, Weiter

* Dantzig, Fulkerson, Johnson (1954): 42 Städte in den USA
* Grötschel (1977): 120 Städte in (West-) Deutschland
* Padberg, Rinaldi (1987): 532 AT&T Schaltstellen in den USA
* Grötschel, Holland (1987): 666 Sehenswürdigkeiten in der Welt

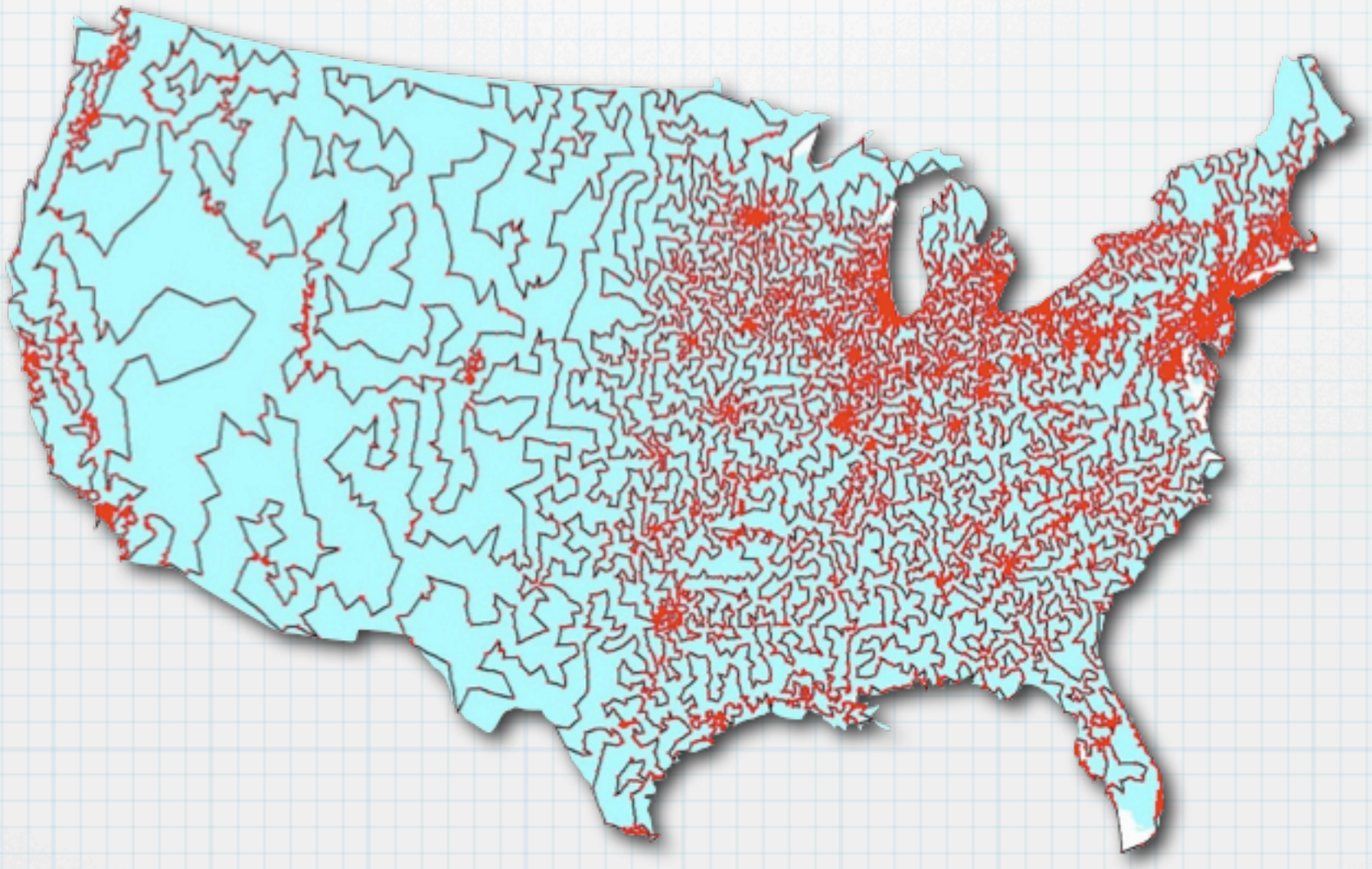
Die TSP-Olympiade: Höher, Schneller, Weiter

* Dantzig, Fulkerson, Johnson (1954): 42 Städte in den USA
* Grötschel (1977): 120 Städte in (West-) Deutschland
* Padberg, Rinaldi (1987): 532 AT&T Schaltstellen in den USA
* Grötschel, Holland (1987): 666 Sehenswürdigkeiten in der Welt
* Padberg, Rinaldi (1987): 2.392 Bohrlöcher einer Platine



Die TSP-Olympiade: Höher, Schneller, Weiter

* Dantzig, Fulkerson, Johnson (1954): 42 Städte in den USA
* Grötschel (1977): 120 Städte in (West-) Deutschland
* Padberg, Rinaldi (1987): 532 AT&T Schaltstellen in den USA
* Grötschel, Holland (1987): 666 Sehenswürdigkeiten in der Welt
* Padberg, Rinaldi (1987): 2.392 Bohrlöcher einer Platine
* Applegate, Bixby, Chvátal, Cook (1998): 13.509 Orte in den USA mit >500 Einwohnern



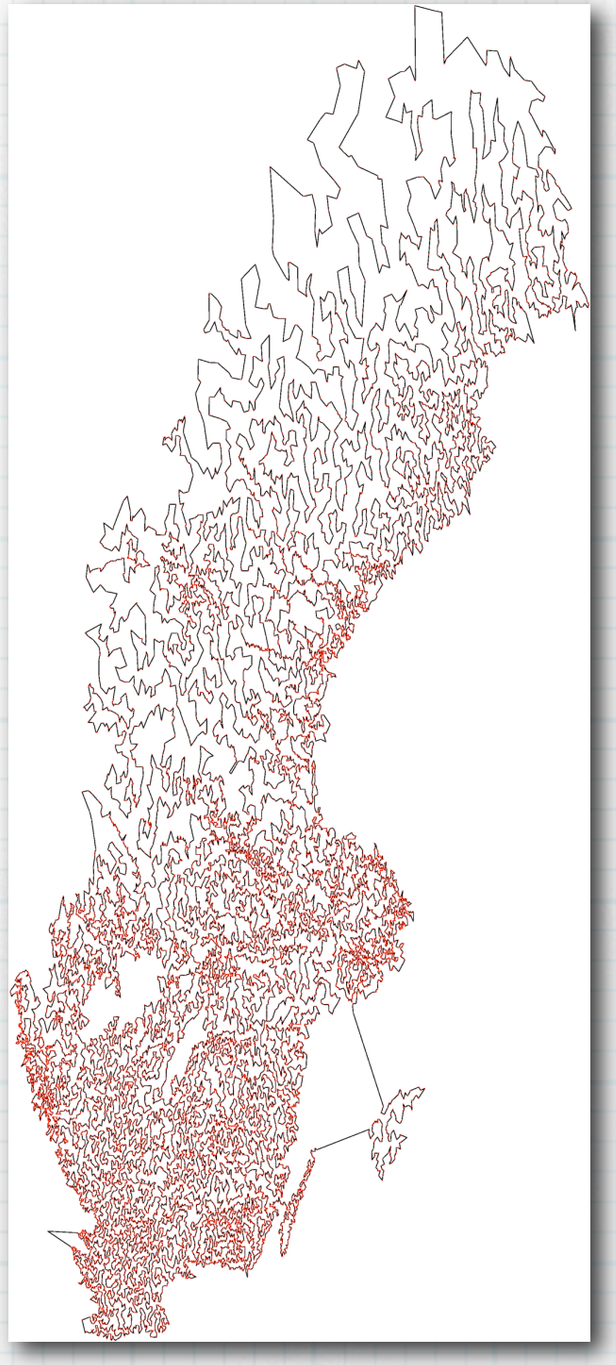
Die TSP-Olympiade: Höher, Schneller, Weiter

* Dantzig, Fulkerson, Johnson (1954): 42 Städte in den USA
* Grötschel (1977): 120 Städte in (West-) Deutschland
* Padberg, Rinaldi (1987): 532 AT&T Schaltstellen in den USA
* Grötschel, Holland (1987): 666 Sehenswürdigkeiten in der Welt
* Padberg, Rinaldi (1987): 2.392 Bohrlöcher einer Platine
* Applegate, Bixby, Chvátal, Cook (1998): 13.509 Orte in den USA mit >500 Einwohnern
* Applegate, Bixby, Chvátal, Cook (2001): 15.112 Orte in Deutschland

Der aktuelle Rekord

Der aktuelle Rekord

* Applegate, Bixby, Chvátal, Cook, Helsgaun (2004): 24.978 bewohnte Orte in Schweden



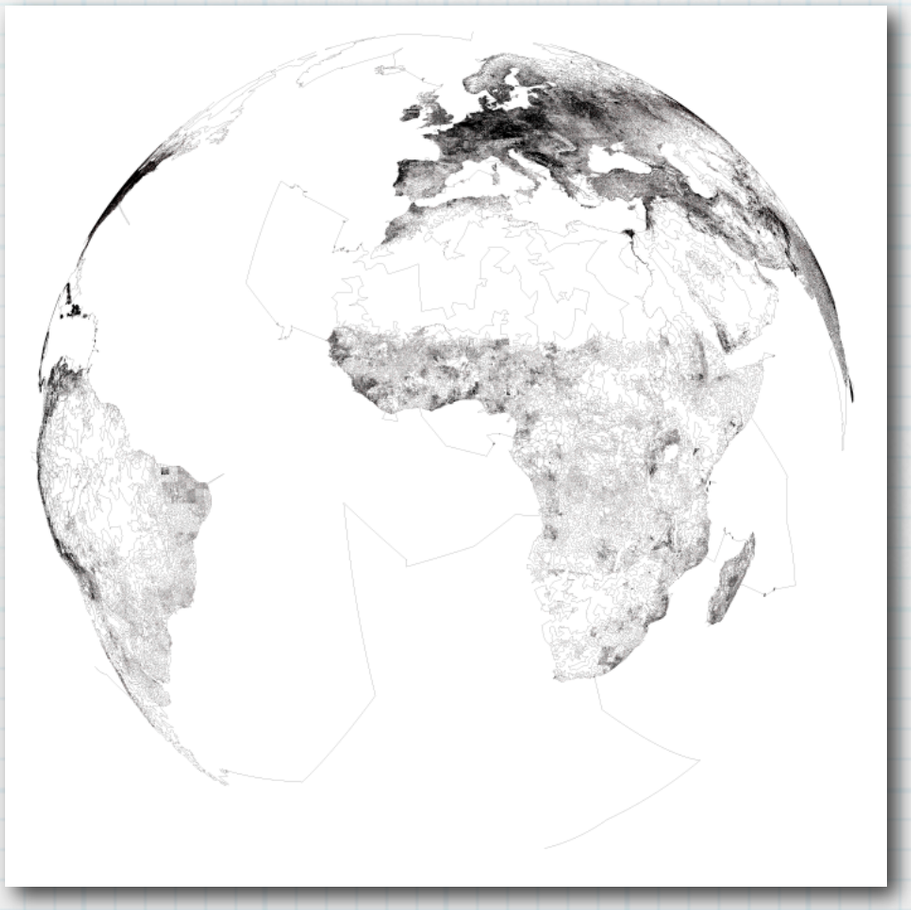
Die gegenwärtige Herausforderung

Die gegenwärtige Herausforderung

* Noch offen: 1.904.711 bewohnte Orte auf der Erde

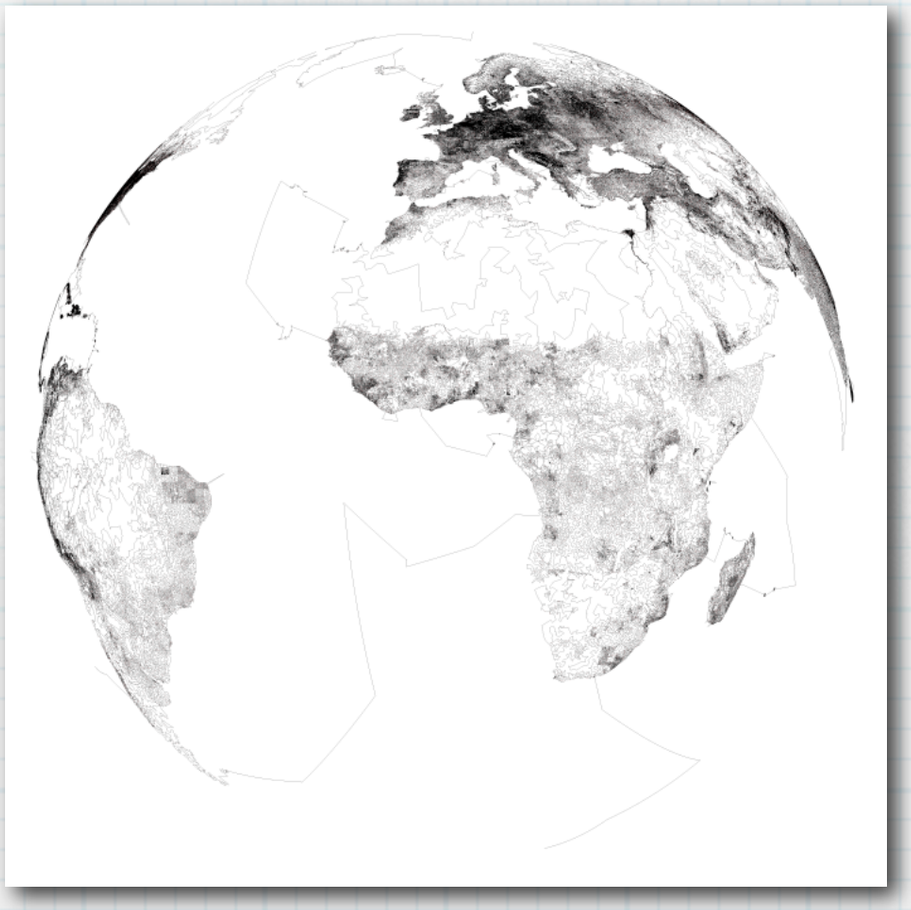
Die gegenwärtige Herausforderung

* Noch offen: 1.904.711 bewohnte Orte auf der Erde



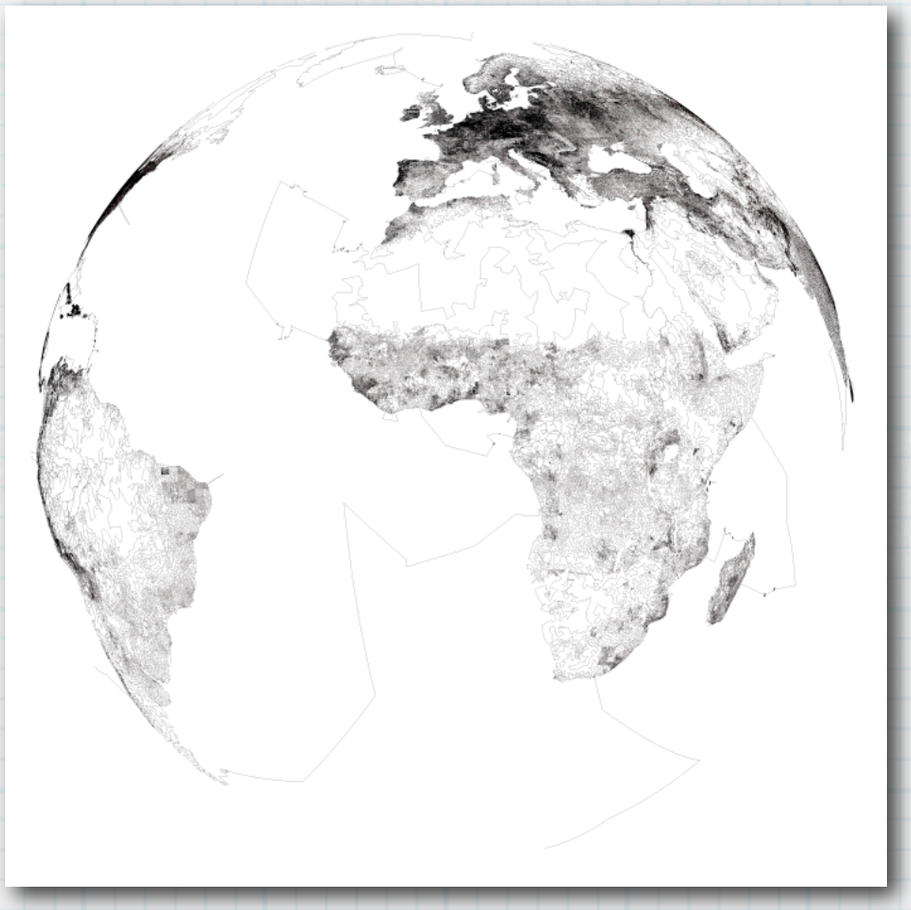
Die gegenwärtige Herausforderung

* Noch offen: 1.904.711 bewohnte Orte auf der Erde
* Beste bekannte Lösungen:



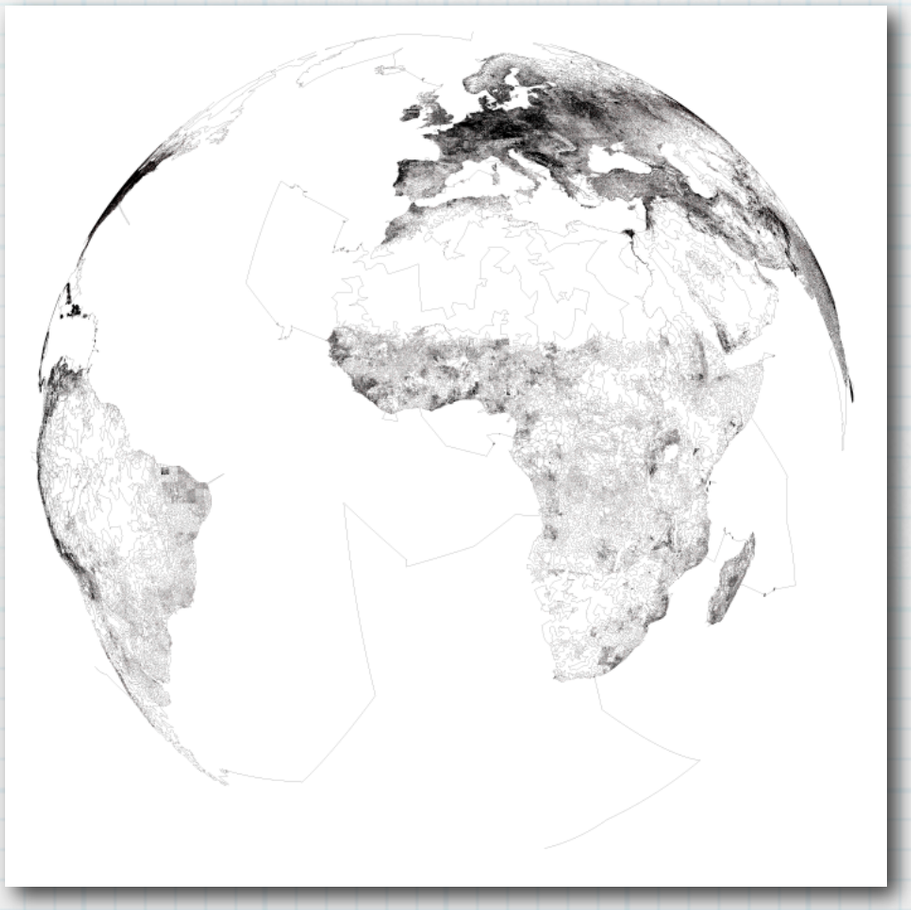
Die gegenwärtige Herausforderung

* Noch offen: 1.904.711 bewohnte Orte auf der Erde
* Beste bekannte Lösungen:
  * Nguyen, Yoshihara, Yamamori, Yasunaga (2003): 7.518.425.642



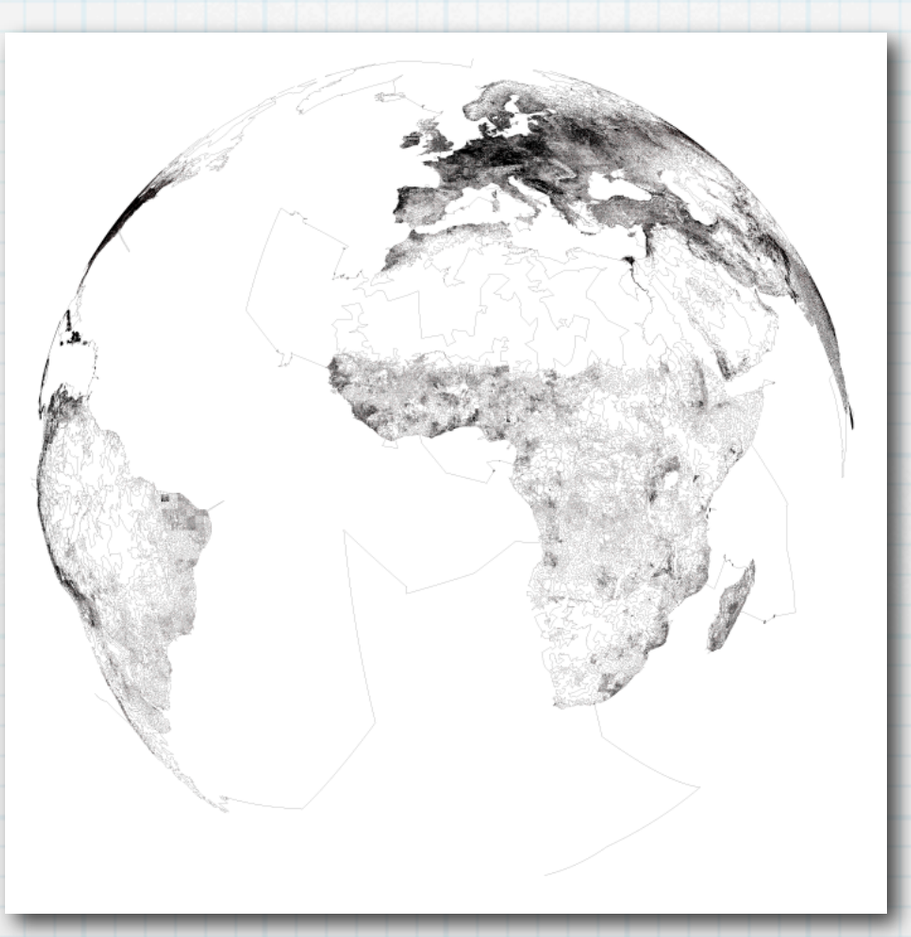
Die gegenwärtige Herausforderung

* Noch offen: 1.904.711 bewohnte Orte auf der Erde
* Beste bekannte Lösungen:
  * Nguyen, Yoshihara, Yamamori, Yasunaga (2003): 7.518.425.642
  * Helsgaun (2003): 7,517,285,610 (siehe Abbildung)



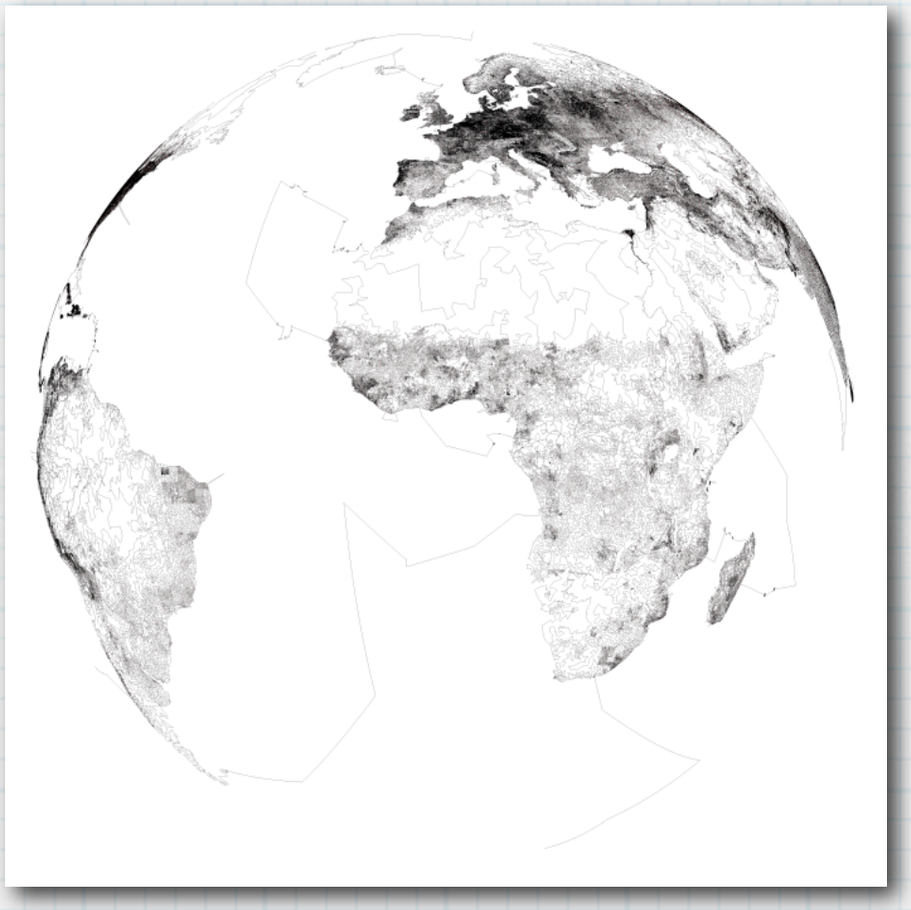
Die gegenwärtige Herausforderung

* Noch offen: 1.904.711 bewohnte Orte auf der Erde
* Beste bekannte Lösungen:
  * Nguyen, Yoshihara, Yamamori, Yasunaga (2003): 7.518.425.642
  * Helsgaun (2003): 7,517,285,610 (siehe Abbildung)
  * Helsgaun (2007): 7,515,964,553



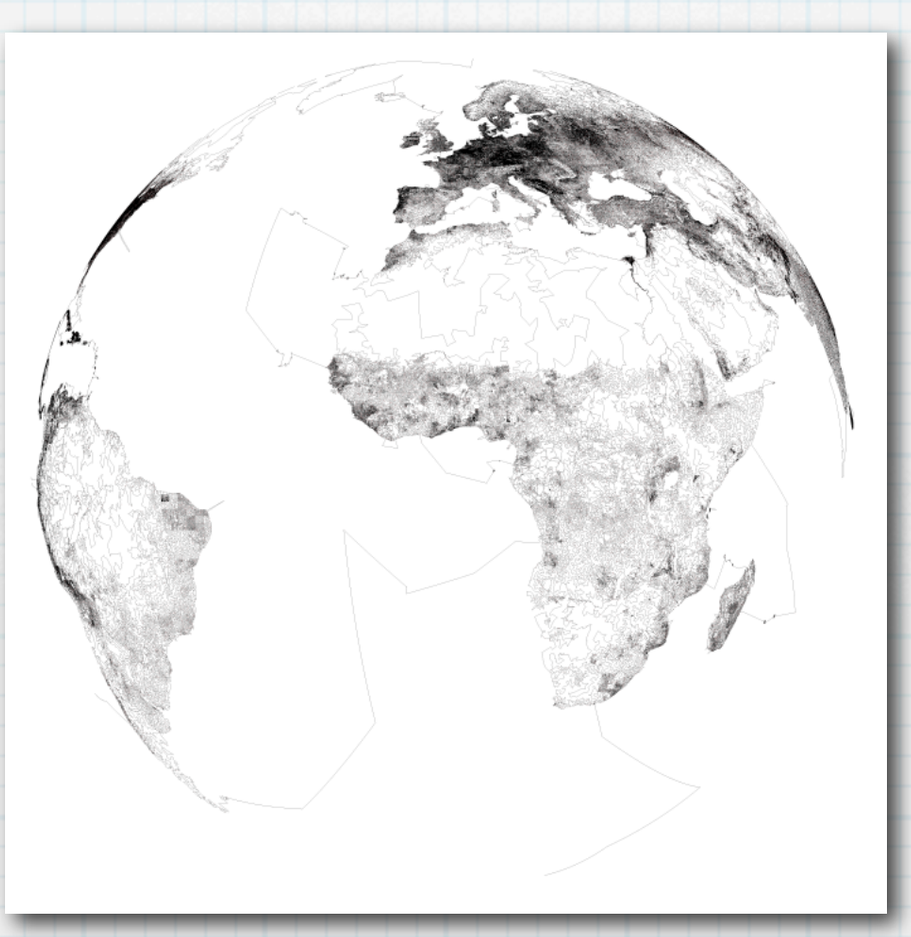
Die gegenwärtige Herausforderung

- Noch offen: 1.904.711 bewohnte Orte auf der Erde
- Beste bekannte Lösungen:
  - Nguyen, Yoshihara, Yamamori, Yasunaga (2003): 7.518.425.642
  - Helsgaun (2003): 7,517,285,610 (siehe Abbildung)
  - Helsgaun (2007): 7,515,964,553
- Beste bekannte untere Schranke:



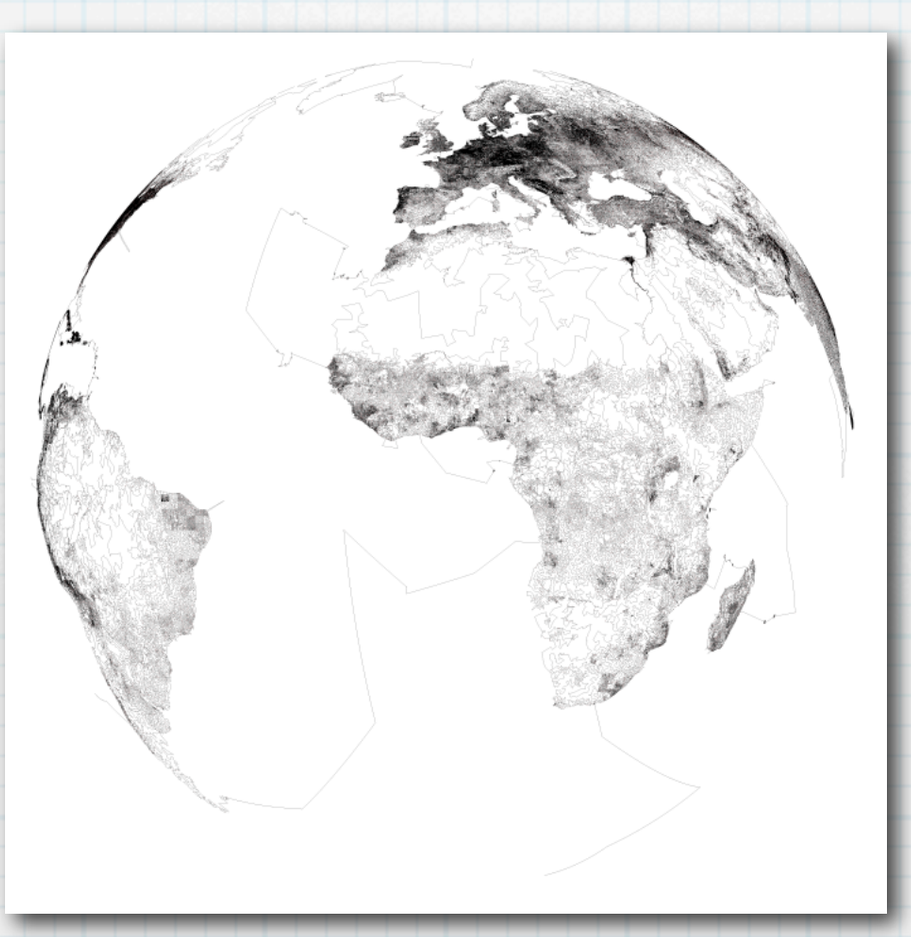
Die gegenwärtige Herausforderung

- Noch offen: 1.904.711 bewohnte Orte auf der Erde
- Beste bekannte Lösungen:
  - Nguyen, Yoshihara, Yamamori, Yasunaga (2003): 7.518.425.642
  - Helsgaun (2003): 7,517,285,610 (siehe Abbildung)
  - Helsgaun (2007): 7,515,964,553
- Beste bekannte untere Schranke:
  - Applegate, Bixby, Chvátal, Cook, Helsgaun (2007): 7,512,218,268



Die gegenwärtige Herausforderung

- Noch offen: 1.904.711 bewohnte Orte auf der Erde
- Beste bekannte Lösungen:
  - Nguyen, Yoshihara, Yamamori, Yasunaga (2003): 7.518.425.642
  - Helsgaun (2003): 7,517,285,610 (siehe Abbildung)
  - Helsgaun (2007): 7,515,964,553
- Beste bekannte untere Schranke:
  - Applegate, Bixby, Chvátal, Cook, Helsgaun (2007): 7,512,218,268
- Die beste Lösung (Helsgaun, 2007) ist höchstens 0,04987% schlechter als eine optimale Tour.



Literaturquellen

Literaturquellen

* J. Clark, D.A. Holton: **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 3, Seite 93–132)

Literaturquellen

* J. Clark, D.A. Holton: **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 3, Seite 93–132)
* W. Cook: **Taveling Salesman Problem Homepage.** http://www.tsp.gatech.edu

Literaturquellen

* J. Clark, D.A. Holton: **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 3, Seite 93–132)
* W. Cook: **Taveling Salesman Problem Homepage.** http://www.tsp.gatech.edu
* A. Fügenschuh: **The Integrated Optimization of School Starting Times and Public Transport**, Logos Verlag, Berlin, 2005. (Kapitel 5, Seite 53–74)

Literaturquellen

* J. Clark, D.A. Holton: **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 3, Seite 93–132)
* W. Cook: **Taveling Salesman Problem Homepage.** http://www.tsp.gatech.edu
* A. Fügenschuh: **The Integrated Optimization of School Starting Times and Public Transport**, Logos Verlag, Berlin, 2005. (Kapitel 5, Seite 53–74)
* D. Jungnickel: **Graphen, Netzwerke und Algorithmen**, BI Wissenschaftsverlag, Mannheim, 1994. (Kapitel 14, Seite 515–570)

Literaturquellen

* J. Clark, D.A. Holton: **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 3, Seite 93–132)
* W. Cook: **Taveling Salesman Problem Homepage.** http://www.tsp.gatech.edu
* A. Fügenschuh: **The Integrated Optimization of School Starting Times and Public Transport**, Logos Verlag, Berlin, 2005. (Kapitel 5, Seite 53–74)
* D. Jungnickel: **Graphen, Netzwerke und Algorithmen**, BI Wissenschaftsverlag, Mannheim, 1994. (Kapitel 14, Seite 515–570)
* A. Schrijver: **On the History of Combinatorial Optimization (till 1960).** http://www.csi.nl/~lex