Vorlesung: Algorithmische Diskrete Mathematik

TECHNISCHE UNIVERSITÄT DARMSTADT

Raymond Hemmecke

Vertretungsprofessor für Algorithmische Diskrete Mathematik

Inhalt der Vorlesung

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Komplexitätstheorie**
  - Datenstrukturen und Kodierungsschemata
  - asymptotische Notation, untere und obere Schranken
  - Klassen $\mathcal{P}, \mathcal{NP}, \mathcal{NP}$-vollständig
- **Algorithmen auf Graphen**
  - DFS-Algorithmus (aufspannende Bäume)
  - Greedy-Algorithmus (minimale aufspannende Bäume)
  - Dijkstra, Moore-Bellman, Yen-Variante (kürzeste Wege in Graphen)
  - Ford-Fulkerson (maximale Flüsse in Netzwerken, Matching in bipartiten Graphen)
- **Sortieren in Arrays**
  - Mergesort, Quicksort, Heapsort
  - Divide-and-Conquer
  - untere Komplexitätsschranken für das Sortieren

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Grundlegende Begriffe der Graphentheorie

Was ist ein Graph?

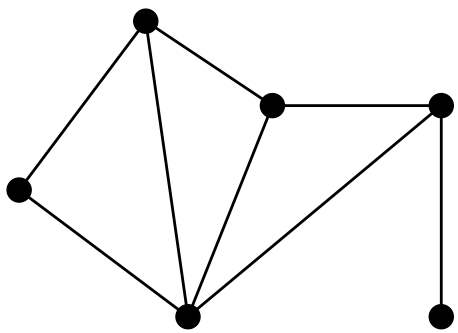
TECHNISCHE UNIVERSITÄT DARMSTADT

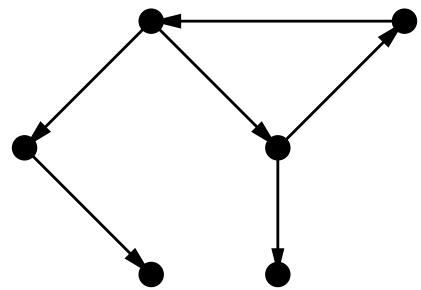
Graph

- Ein ungerichter **Graph** ist ein Paar $G = (V, E)$ disjunkter Mengen, wobei die Elemente von E ungeordnete Paare von Elementen aus V sind.
- Die Elemente aus V heißen **Knoten**, die Elemente aus E **Kanten**.

Digraph

- Ein **gerichter Graph** ist ein Paar $G = (V, A)$ disjunkter Mengen, wobei die Elemente von A geordnete Paare von Elementen aus V sind.
- Die Elemente aus A heißen **Bögen** oder **gerichtete Kanten**.





Nachbarschaftsbeziehungen

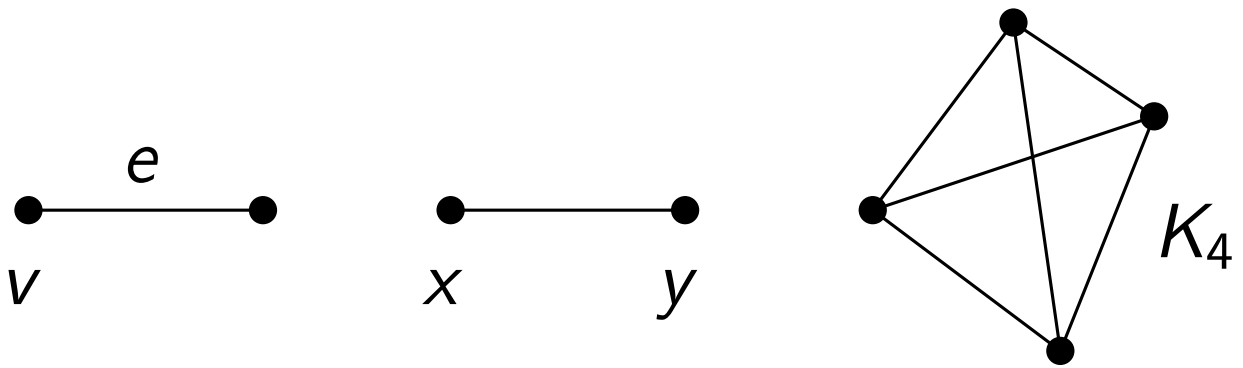
TECHNISCHE UNIVERSITÄT DARMSTADT

Inzidenz, Grad/Valenz

- Ein Knoten v heißt mit einer Kante e **inzident**, wenn $v \in e$ gilt.
- Der **Grad**/die **Valenz** eines Knoten v ist die Anzahl der mit v inzidenten Kanten.

Adjazenz, vollständiger Graph

- Zwei Knoten x und y heißen **adjazent** oder **benachbart** in G, wenn $xy \in E(G)$ ist.
- Sind je zwei Knoten von G benachbart, so heißt G **vollständig**.

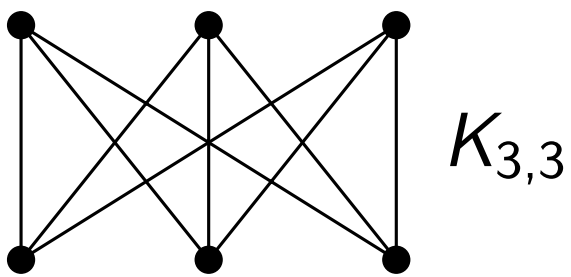


Bipartite Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

- Ein Graph heißt **bipartit**, wenn sich V disjunkt in zwei Teile V_1 und V_2 zerteilen lässt, so dass jede Kante in E einen Endknoten in V_1 und einen Endknoten in V_2 besitzt.



Fakt → Beweis als Übungsaufgabe

- Ein Graph ist genau dann bipartit, wenn er nur Kreise gerader Länge enthält.

Was sind Kreise?

TECHNISCHE UNIVERSITÄT DARMSTADT

Pfad, Weg, Kreis

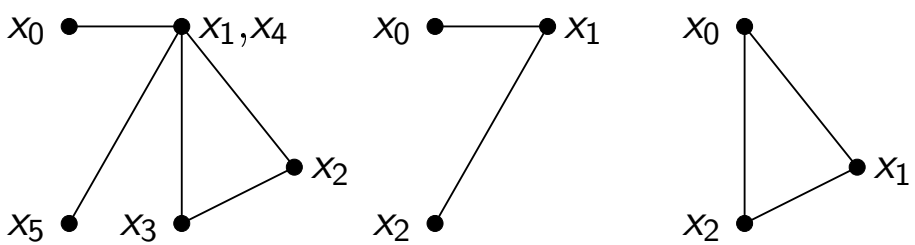
- Ein **Pfad** ist ein nichtleerer Graph $P = (V, E)$ der Form

$$V = \{x_0, x_1, \dots, x_k\}, E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}.$$

- Ein **Weg** ist ein Pfad, in dem alle x_i paarweise verschieden sind.
- Ein **Kreis** ist ein nichtleerer Graph $P = (V, E)$ der Form

$$V = \{x_0, x_1, \dots, x_k\}, E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k, x_kx_0\},$$

wobei die x_i paarweise verschieden sind.



Zusammenhangskomponenten, Bäume und Wälder

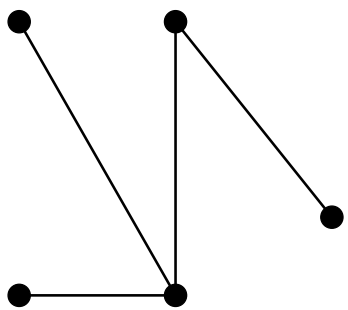
TECHNISCHE UNIVERSITÄT DARMSTADT

Zusammenhängende Graphen

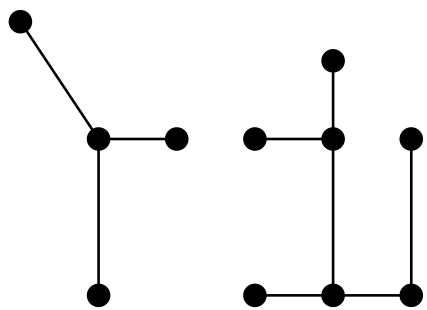
- Ein Graph G heißt **zusammenhängend**, falls es zu jedem Paar $v, w \in V$ einen Weg von v nach w in G gibt.
- Die zusammenhängenden Teile von G heißen **Zusammenhangskomponenten**.

Bäume und Wälder

- Ein **Baum** ist ein zusammenhängender Graph, der keinen Kreis besitzt.
- Ein **Wald** ist ein Graph, der keinen Kreis besitzt.



Baum



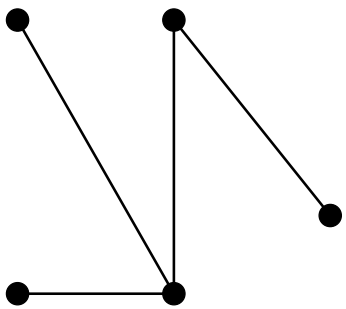
Wald, 2 Komponenten

Komplementärgraph

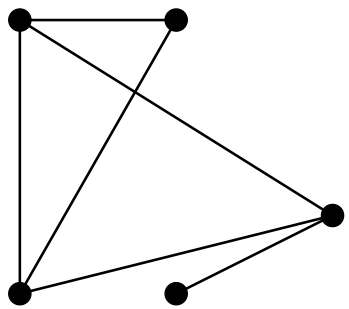
TECHNISCHE UNIVERSITÄT DARMSTADT

Komplementärgraph

- Der zu G **komplementäre Graph** $\bar{G}$ ist der Graph $\bar{G} = (V, \bar{E})$, mit $ij \in \bar{E} \Leftrightarrow ij \notin E$.



Graph



Komplement

Fakt $\longrightarrow$ Beweis als Übungsaufgabe

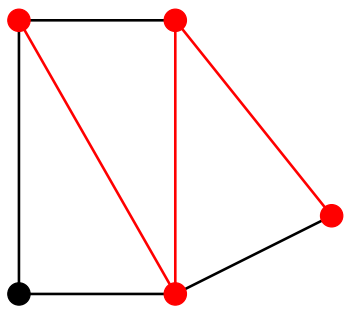
- Mindestens einer der Graphen G oder $\bar{G}$ ist zusammenhängend.

Untergraphen, aufspannende Untergraphen

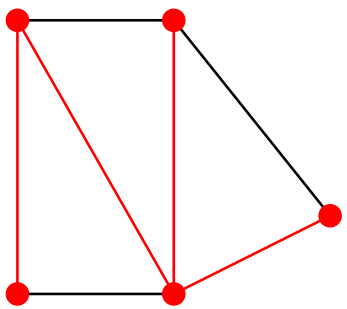
TECHNISCHE UNIVERSITÄT DARMSTADT

Untergraph, aufspannende Untergraphen

- $G' = (V', E')$ heißt **Untergraph** von $G = (V, E)$, falls $V' \subseteq V$ und $E' \subseteq E$ gilt.
- $G' \subseteq G$ heißt **aufspannender Untergraph** von G, falls $V' = V$ gilt.



Graph mit Untergraph



aufspannender Baum

Finden eines aufspannenden Waldes → DFS-Algorithmus

- Für gegebenes $G = (V, E)$ finde man einen aufspannenden Wald.

Probleme auf Graphen (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

Finden eines Kreises/Hamiltonischen Kreises

- Entscheide für einen gegebenen Graphen $G = (V, E)$, ob er einen Kreis enthält. Falls ja, gib einen solchen an.
- Entscheide für einen gegebenen Graphen $G = (V, E)$, ob er einen hamiltonischen Kreis enthält. Falls ja, gib einen solchen an. (Ein **hamiltonischer Kreis** ist ein Kreis, der jeden Knoten genau einmal durchläuft.)

Finden eines aufspannenden Waldes → DFS-Algorithmus

- Für gegebenes $G = (V, E)$ finde man einen aufspannenden Wald.

Maximales Matching

- Finde für gegebenen Graphen $G = (V, E)$ die maximale Anzahl von Kanten aus E, so dass je zwei Kanten nicht inzident sind.

Probleme auf Graphen (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Maximale stabile Mengen/Cliquen

- Finde für gegebenen Graphen $G = (V, E)$ die maximale Anzahl von Knoten aus V, so dass je zwei Knoten nicht benachbart sind. $\rightarrow$ stabile Menge
- Finde für gegebenen Graphen $G = (V, E)$ die maximale Anzahl von Knoten aus V, so dass je zwei Knoten benachbart sind. $\rightarrow$ Clique

Kürzeste Wege

- Für gegebenes $G = (V, E)$ und gegebenen Kantengewichten c_{ij} finde man einen kürzesten Weg zwischen zwei gegebenen Knoten v und w.

Färbungsproblem

- Finde für gegebenen Graphen $G = (V, E)$ die minimale Anzahl k von Farben, so dass sich die Knoten von G so mit k Farben färben lassen, dass benachbarte Knoten unterschiedlich gefärbt sind.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Asymptotische Notation

Asymptotische Notation

TECHNISCHE UNIVERSITÄT DARMSTADT

Obere Schranken: O-Notation

- Sei $g : \mathbb{N} \to \mathbb{R}$. Dann bezeichnet

$$O(g) := \{f : \mathbb{N} \to \mathbb{R} : \exists c > 0, n_0 \in \mathbb{N} \text{ mit } |f(n)| \leq c|g(n)| \; \forall n \geq n_0\}$$

  die Menge aller Funktionen $f : \mathbb{N} \to \mathbb{R}$, für die zwei positive Konstanten $c \in \mathbb{R}$ und $n_0 \in \mathbb{N}$ existieren, so dass für alle $n \geq n_0$ gilt $|f(n)| \leq c|g(n)|$.

Bemerkungen

- Die asymptotische Notation vernachlässigt Konstanten und Terme niedrigerer Ordnung (wie z.B. die Terme $a_k n^k$ mit $k < m$ im Polynom).

Satz

- Für ein Polynom $f(n) = a_m n^m + ... + a_1 n + a_0$ vom Grade m gilt $f \in O(n^m)$.

Vergleich

TECHNISCHE UNIVERSITÄT DARMSTADT

Die nachfolgende Tabelle zeigt, wie sechs typischen Funktionen anwachsen, wobei die Konstante gleich 1 gesetzt wurde. Wie man feststellt, zeigen die Zeiten vom Typ $O(n)$ und $O(n \log n)$ ein wesentlich schwächeres Wachstum also die anderen.

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

Untere Schranken

TECHNISCHE UNIVERSITÄT DARMSTADT

Ω- und Θ-Notation

- Seien $f, g : \mathbb{N} \to \mathbb{R}$. Dann ist $f \in \Omega(g)$, wenn zwei positive Konstanten $c \in \mathbb{R}$ und $n_0 \in \mathbb{N}$ existieren, so dass für alle $n \geq n_0$ gilt:

$$|f(n)| \geq c|g(n)|.$$

- $f \in \Theta(g)$, wenn es positive Konstanten $c_1, c_2 \in \mathbb{R}$ und $n_0 \in \mathbb{N}$ gibt, so dass für alle $n \geq n_0$ gilt $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$.
- Falls $f \in \Theta(g)$, dann ist g sowohl eine obere als auch eine untere Schranke für f.

Beispiel: Sequentielle Suche

- Sei $f(n)$ die Anzahl von Vergleichen bei der sequentiellen Suche nach einem bestimmten Wert in einem unsortierten Array mit n Komponenten. Dann ist $f \in O(n)$, da man ja mit n Vergleichen auskommt.
- Andererseits muss man auch jede Komponente überprüfen, denn ihr Wert könnte ja der gesuchte Wert sein. Also $f \in \Omega(n)$ und damit $f \in \Theta(n)$.

Ein weiteres Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Matrixmultiplikation

- Bei der Matrixmultiplikation von $n \times n$ Matrizen ergibt sich die Berechnung eines Eintrags c_{ij} von $C = A \cdot B$ gemäß $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$. Sie erfordert also n Multiplikationen und $n-1$ Additionen.
- Insgesamt sind für ganz C also n^2 Einträge c_{ij} zu berechnen, und somit $n^2(n+n-1) = 2n^3 - n^2 = O(n^3)$ arithmetische Operationen insgesamt auszuführen.
- Da jeder Algorithmus für die Matrixmultiplikation n^2 Einträge berechnen muss, folgt andererseits, dass jeder Algorithmus zur Matrixmultiplikation von zwei $n \times n$ Matrizen $\Omega(n^2)$ Operationen benötigt.
- Es klafft zwischen $\Omega(n^2)$ und $O(n^3)$ also noch eine "Komplexitätslücke". Die schnellsten bekannten Algorithmen zur Matrixmultiplikation kommen mit $O(n^{2,376})$ Operationen aus.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Komplexitätstheorie

Was ist ein Problem?

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

- Ein **Problem** ist eine allgemeine Fragestellung, bei der mehrere Parameter offen gelassen sind und für die eine Lösung oder Antwort gesucht wird.
- Ein Problem ist dadurch definiert, dass alle seine Parameter beschrieben werden und dass genau angegeben wird, welche Eigenschaften eine Antwort (Lösung) haben soll.
- Sind alle Parameter eines Problems mit expliziten Daten belegt, dann sprechen wir von einem Problembeispiel (**Instanz**).

Beispiel

- Finde einen kürzesten Hamiltonischen Kreis.
- Offene Parameter: Anzahl Städte, Entfernungen

Algorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

Was ist ein Algorithmus?

- Ein **Algorithmus** ist eine Anleitung zur schrittweisen Lösung eines Problems. Wir sagen, ein Algorithmus A löst ein Problem Π, falls A für alle Problembeispiele $I \in \Pi$, eine Lösung in einer endlichen Anzahl an Schritten findet.
- Ein Schritt ist eine elementare Operation: Addieren, Subtrahieren, Vergleichen, Multiplikation, Division.
- Die Laufzeit eines Algorithmus ist die Anzahl der Schritte, die notwendig sind zur Lösung des Problembeispiels.

Forschungsschwerpunkte zu Algorithmen

- Entwurf von Algorithmen
- Berechenbarkeit: Was kann durch einen Algorithmus berechnet werden?
- Komplexität von Algorithmen
- Korrektheit von Algorithmen

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Komplexität von Algorithmen

Was ist Effizienz?

TECHNISCHE UNIVERSITÄT DARMSTADT

Was ist Effizienz?

- Komplexitätstheorie
- Speicher- und Laufzeitkomplexität

Trivial

- Laufzeit eines Algorithmus hängt ab von der "Größe" der Eingabedaten
- Laufzeitanalyse erfordert eine Beschreibung, wie Problembeispiele dargestellt werden (Kodierungsschema)
- Notwendigkeit exakter Definitionen
  - geeignetes Rechnermodell $\rightarrow$ **Turing-Maschine**

Kodierungsschema

TECHNISCHE UNIVERSITÄT DARMSTADT

Ganze Zahlen

- Ganze Zahlen werden binär dargestellt, d.h. wir schreiben

$$n = \pm \sum_{i=0}^{k} x_i \cdot 2^i,$$

mit $x_i \in \{0, 1\}$ und $k = \lfloor \log_2(|n|) \rfloor$.

- D.h. die Kodierungslänge $\langle n \rangle$ einer ganzen Zahl n ist gegeben durch die Formel

$$\langle n \rangle = \lceil \log_2(|n| + 1) \rceil + 1 = \lfloor \log_2(|n|) \rfloor + 2,$$

wobei $+1$ wegen des Vorzeichens $+$ oder $-$.

Rationale Zahlen

- Sei $r \in \mathbb{Q}$. Dann existieren $p \in \mathbb{Z}$ und $q \in \mathbb{Z}$, teilerfremd, mit $r = \frac{p}{q}$.

$$\langle r \rangle = \langle p \rangle + \langle q \rangle$$

Kodierungsschema (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Vektoren

- Für $x = (x_1, \ldots, x_n)^\intercal \in \mathbb{Q}^n$ ist

$$\langle x \rangle = \sum_{i=1}^{n} \langle x_i \rangle .$$

Matrizen

- Für $A \in \mathbb{Q}^{m \times n}$ ist

$$\langle A \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle a_{ij} \rangle .$$

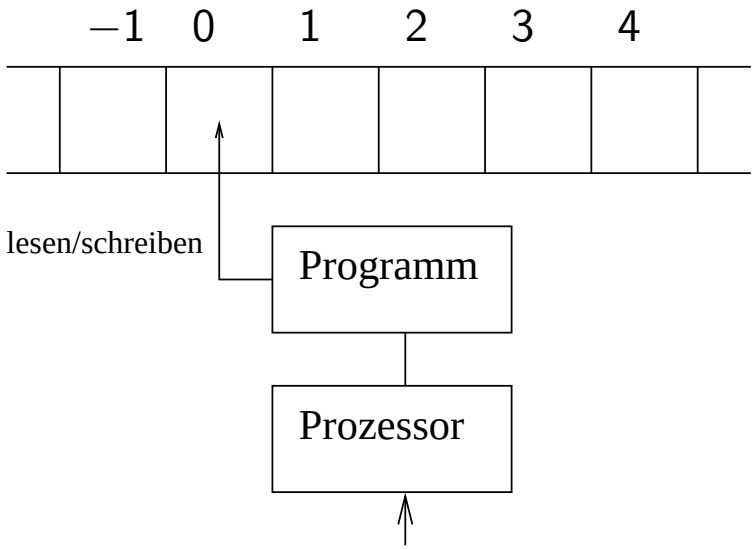
Rechnermodell

TECHNISCHE UNIVERSITÄT DARMSTADT

Nach Festlegung der Kodierungsvorschrift muss ein Rechnermodell entworfen werden, auf dem unsere Speicher- und Laufzeitberechnungen durchgeführt werden.

In der Komplexitätstheorie: **Turing-Maschine** (ein ganz normaler Computer)

Unendliches Band, auf dem Lese-/Schreiboperationen durchgeführt werden



stellt Operationen zur Verfügung:
+, −, *, /, vergleichen, löschen, schreiben, lesen

Ablauf eines Algorithmus auf der Turing-Maschine

TECHNISCHE UNIVERSITÄT DARMSTADT

Algorithmus A soll Problembeispiel I des Problems Π lösen. Alle Problembeispiele liegen in kodierter Form vor.

Inputlänge

- Die Anzahl der Speicherplätze, die nötig sind, um I vollständig anzugeben, heißt **Inputlänge**, $\langle I \rangle$.

Der Algorithmus liest die Daten und beginnt dann, Operationen auszuführen, d.h. Zahlen zu berechnen, zu speichern, zu löschen.

Speicherbedarf

- Die Anzahl der Speicherplätze, die während der Ausführung des Algorithmus mindestens einmal benutzt werden, nennen wir **Speicherbedarf** von A zur Lösung von I.

Laufzeit

TECHNISCHE UNIVERSITÄT DARMSTADT

Informell

- Die **Laufzeit** von A zur Lösung von I ist die Anzahl elementarer Operationen. Dazu zählen $+$, $-$, $*$, $/$, Vergleich, Löschen, Schreiben, Lesen von rationalen Zahlen.

Dies ist jedoch zu unpräzise!

- Zur Darstellung der entsprechenden Zahlen werden mehrere Bits benötigt.

$\rightarrow$ Für jede Operation müssen wir mehrere Bits zählen.

Korrekte Definition

- Die **Laufzeit** von A zur Lösung von I ist definiert durch die Anzahl elementarer Operationen, die A ausgeführt hat, um I zu lösen, multipliziert mit der größten Kodierungslänge der während der Berechnung aufgetretenen ganzen oder rationalen Zahl.

Laufzeitfunktion, Speicherplatzfunktion (worst-case)

TECHNISCHE UNIVERSITÄT DARMSTADT

Laufzeitfunktion, polynomielle Laufzeit

- Sei A ein Algorithmus zur Lösung eines Problems Π. Die Funktion $f_A : \mathbb{N} \to \mathbb{N}$,

$$f_A(n) = \max\{\text{Laufzeit von A zur Lösung von } I : I \in \Pi, \langle I \rangle \leq n\}$$

  heißt **Laufzeitfunktion** von A.
- Der Algorithmus A hat eine **polynomiale Laufzeit** (kurz: A ist ein polynomialer Algorithmus), wenn es ein Polynom $p : \mathbb{N} \to \mathbb{N}$ gibt mit $f_A(n) \leq p(n)$ für alle $n \in \mathbb{N}$.

Speicherplatzfunktion, polynomieller Speicherbedarf

- Die **Speicherplatzfunktion** $s_A : \mathbb{N} \to \mathbb{N}$ von A ist definiert durch

$$s_A(n) = \max\{\text{Speicherbedarf von } A \text{ zur Lösung von } I : I \in \Pi, \langle I \rangle \leq n\}.$$

- Der Algorithmus A hat polynomialen Speicherbedarf, falls es ein Polynom $q : \mathbb{N} \to \mathbb{N}$ gibt mit $s_A(n) \leq q(n)$ für alle $n \in \mathbb{N}$.

Weitere Bemerkungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Bemerkungen

- Nehmen wir an, wir ermitteln die Rechenzeit $f(n)$ für einen bestimmten Algorithmus. Die Variable n kann z.B. die Anzahl der Ein- und Ausgabewerte sein, ihre Summe, oder auch die Größe eines dieser Werte. Da $f(n)$ maschinenabhängig ist, genügt eine a priori Analyse nicht. Jedoch kann man mit Hilfe einer a priori Analyse ein g bestimmen, so dass $f \in O(g)$.
- Wenn wir sagen, dass ein Algorithmus eine Rechenzeit $O(g)$ hat, dann meinen wir damit folgendes: Wenn der Algorithmus auf unterschiedlichen Computern mit den gleichen Datensätzen läuft, und diese Größe n haben, dann werden die resultierenden Laufzeiten immer kleiner sein als eine Konstante mal $|g(n)|$. Bei der Suche nach der Größenordnung von f werden wir darum bemüht sein, das kleinste g zu finden, so dass $f \in O(g)$ gilt.

Die Klassen $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-vollständig

Die Klasse $\mathcal{P}$

TECHNISCHE UNIVERSITÄT DARMSTADT

Was ist ein Entscheidungsproblem?

- Problem, das nur zwei mögliche Antworten besitzt, "ja" oder "nein".
- Beispiele: "Ist n eine Primzahl?" oder "Enthält G einen Kreis?"

Die Klasse $\mathcal{P}$: informelle Definition

- Klasse der Entscheidungsprobleme, für die ein polynomialer Lösungsalgorithmus existiert

Die Klasse $\mathcal{P}$: formale Definition

- Gegeben ein Kodierungsschema E und ein Rechnermodell M.
- Π sei ein Entscheidungsproblem, wobei jede Instanz aus Π durch Kodierungsschema E kodierbar sei.
- Π gehört zur Klasse $\mathcal{P}$ (bzgl. E und M), wenn es einen auf M implementierbaren Algorithmus zur Lösung aller Problembeispiele aus Π gibt, dessen Laufzeitfunktion auf M polynomial ist.

Die Klasse $\mathcal{NP}$

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation

- Enthält G einen Kreis? $\rightarrow$ "einfach" $\rightarrow \mathcal{P}$
- Enthält G einen hamiltonischen Kreis? $\rightarrow$ "schwieriger" $\rightarrow \mathcal{NP}$

Die Klasse $\mathcal{NP}$: informelle Definition

- Entscheidungsproblem Π gehört zur Klasse $\mathcal{NP}$, wenn es folgende Eigenschaft hat:
  - Ist die Antwort "ja" für $I \in \Pi$, dann kann Korrektheit dieser Aussage mit Hilfe eines geeigneten Zusatzobjekts in polynomialer Laufzeit überprüft werden.

Beispiel: "Enthält G einen hamiltonischen Kreis?"

- Geeignetes Zusatzobjekt wäre ein hamiltonischer Kreis.
- Es muss nun lediglich geprüft werden, ob der angegebene Kreis tatsächlich ein hamiltonischer Kreis von G ist.

Die Klasse $\mathcal{NP}$ (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Formale Definition

- Entscheidungsproblem Π gehört zu $\mathcal{NP}$, wenn es die folgende Eigenschaften hat:
  - (a) Für jedes Problembeispiel $I \in \Pi$, für das die Antwort "ja" lautet, gibt es mindestens ein Objekt Q, mit dessen Hilfe die Korrektheit der "ja"-Antwort überprüft werden kann.
  - (b) Es gibt einen Algorithmus, der Problembeispiele $I \in \Pi$ und Zusatzobjekte Q als Input akzeptiert und der in einer Laufzeit, die polynomial in $\langle I \rangle$ ist, überprüft, ob Q ein Objekt ist, aufgrund dessen Existenz eine "ja"-Antwort für I gegeben werden muss.

Bemerkungen

- Es wird keine Aussage über die Berechenbarkeit eines geeigneten Q gemacht. Q kann auch geraten werden.
- Da der Überprüfungsalgorithmus Q lesen muss, muss $\langle Q \rangle$ polynomial in $\langle I \rangle$ sein!
- Es wird keine Aussage zu möglicher "nein"-Antwort gemacht. $\longrightarrow$ coNP

Charakterisierung besonders schwieriger Probleme in $\mathcal{NP}$

TECHNISCHE UNIVERSITÄT DARMSTADT

Polynomiale Transformation von Problemen

- Gegeben seien zwei Entscheidungsprobleme Π und Π'.
- Eine polynomiale Transformation von Π in Π' ist ein polynomialer Algorithmus, der, gegeben ein (kodiertes) Problembeispiel $I \in \Pi$, ein (kodiertes) Problembeispiel $I' \in \Pi'$ produziert, so dass folgendes gilt:
  - Die Antwort auf I ist genau dann "ja", wenn die Antwort auf I' "ja" ist.

Die Klasse $\mathcal{NP}$-vollständig

- Ein Entscheidungsproblem Π heißt $\mathcal{NP}$-**vollständig**, falls $\Pi \in \mathcal{NP}$ und falls jedes andere Problem aus $\mathcal{NP}$ polynomial in Π transformiert werden kann.

Bemerkung und Fragen

TECHNISCHE UNIVERSITÄT DARMSTADT

Bemerkung

- Falls ein $\mathcal{NP}$-vollständiges Entscheidungsproblem Π in polynomialer Zeit gelöst werden kann, so gilt das auch für jedes andere Problem aus $\mathcal{NP}$:

$$\Pi \quad \mathcal{NP} - \text{vollständig und } \Pi \in \mathcal{P} \Rightarrow \mathcal{P} = \mathcal{NP}.$$

- Diese Eigenschaft zeigt, dass (bzgl. polynomialer Lösbarkeit) kein Problem in $\mathcal{NP}$ schwieriger ist als ein $\mathcal{NP}$-vollständiges.

Existieren $\mathcal{NP}$-vollständige Probleme? Ja.

- 3-SAT: $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (...) \wedge ... = \text{TRUE}$

Offene Fragen

$$\begin{aligned} \mathcal{P} &= \mathcal{NP} \cap \text{co}\mathcal{NP} \\ \mathcal{NP} &= \text{co}\mathcal{NP} \\ \mathcal{P} &\neq \mathcal{NP} \end{aligned}$$

Komplexität von Optimierungsproblemen

Die Klasse $\mathcal{NP}$-schwer

TECHNISCHE UNIVERSITÄT DARMSTADT

Transformation in Entscheidungsproblem

- Sei O ein Maximierungsproblem (Minimierungsproblem). Man legt zusätzlich zur Instanz I noch eine Schranke B fest und stellt die Frage:
  - Gibt es für I eine Lösung, deren Wert nicht kleiner (größer) als B ist?
- Damit ist ein Optimierungsproblem polynomial in ein Entscheidungsproblem transformierbar.

Die Klasse $\mathcal{NP}$-schwer: formale Definition

- Ein Optimierungsproblem O heißt $\mathcal{NP}$-**schwer**, falls es ein $\mathcal{NP}$-vollständiges Entscheidungsproblem Π gibt, so dass Π in polynomialer Zeit gelöst werden kann, wenn O in polynomialer Zeit gelöst werden kann.

Beispiele

- Rucksackproblem, Maschinenbelegung, stabile Menge maximaler Kardinalität, Clique maximaler Kardinalität, minimale Knotenüberdeckung

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Traveling Salesman Problem

- Input: Instanz I ist charakterisiert durch eine Distanzmatrix $c_{ij} \in \mathbb{Z}\ \forall i, j \in V$, $|V| = n$.
- Problem: "Finde einen bzgl. c minimalen Hamiltonischen Kreis in K_n."
- TSP-Entscheidungsproblem:
  - Input wie beim TSP, $B \in \mathbb{Z}$.
  - Problem: "Existiert eine TSP-Tour der Länge $\leq B$?"

Lemma

- Polynomialer Algorithmus zur Lösung des TSP-Entscheidungsproblems würde polynomialen Algorithmus zur Lösung des TSP implizieren.

Lösung: Binäre Suche

TECHNISCHE UNIVERSITÄT DARMSTADT

Beweis

- Länge eines hamiltonischen Kreises in K_n ist ein ganzzahliger Wert im Intervall $[ns, nt]$, wobei $s = \min\{c_{ij} : i, j \in V, i \neq j\}$ und $t = \max\{c_{ij} : i, j \in V, i \neq j\}$.
- Rufe TSP-Entscheidungsproblem mit $B = \lfloor \frac{n(s+t)}{2} \rfloor$ auf, passe obere Schranke nt oder untere Schranke ns je nach Ergebnis an und iteriere.
- Insgesamt reichen demnach $\log(nt - ns)$ Iterationen aus. □

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Datenstrukturen zur Kodierung von Graphen

Kanten- und Bogenlisten

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

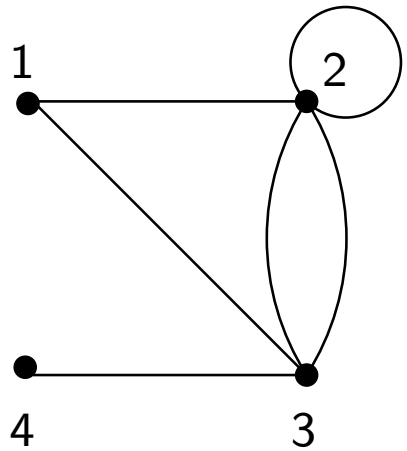
- Ist $G = (V, E)$ ein Graph mit $n = |V|$ Knoten und $m = |E|$ Kanten, so sieht die Kantenliste wie folgt aus:

$$n, m, a_1, e_1, a_2, e_2, \ldots, a_m, e_m,$$

  wobei a_i, e_i die beiden Endknoten der Kante i sind.
- Die Reihenfolge des Aufführens der Endknoten von i bzw. den Kanten ist beliebig.
- Bei Schleifen wird der Endknoten zweimal aufgelistet.
- Bei der Bogenliste eines Digraphen verfahren wir genauso, müssen jedoch darauf achten, dass ein Bogen immer zunächst durch seinen Anfangs- und dann durch seinen Endknoten repräsentiert wird.

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



Kantenliste

- Dieser Graph hat die folgende Kantenliste:

$$4, 6, 1, 2, 2, 3, 4, 3, 3, 2, 2, 2, 1, 3.$$

- Man beachte, dass hier die Reihenfolge der Knoten beliebig ist, da es sich um ungerichtete Kanten handelt.

Gewichte

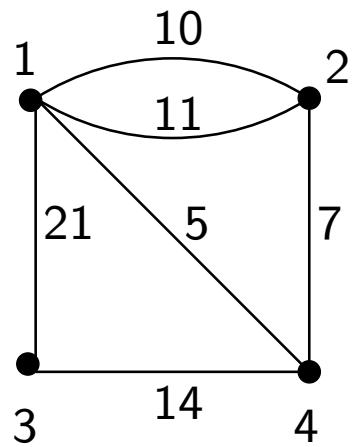
TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

- Haben die Kanten oder Bögen Gewichte, so repräsentiert man eine Kante (einen Bogen) entweder durch Anfangsknoten, Endknoten, Gewicht oder macht eine Kanten- bzw. Bogenliste wie oben und hängt an diese noch eine Liste mit den m Gewichten der Kanten $1, 2, \ldots, m$ an.
- Der Speicheraufwand einer Kanten- bzw. Bogenliste beträgt $2(m+1)$ Speicherzellen, eine Liste mit Gewichten erfordert $3m+2$ Zellen.

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



Kantenliste mit Gewichten

- Dieser gewichtete Graph ist in den beiden folgenden Kantenlisten mit Gewichten gespeichert:
  - 4,6,1,2,10,1,2,11,2,4,7,4,3,14,3,1,21,1,4,5
  - 4,6,1,2,1,2,2,4,3,4,1,4,1,3,11,10,7,14,5,21

Adjazenzmatrix eines ungerichteten Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

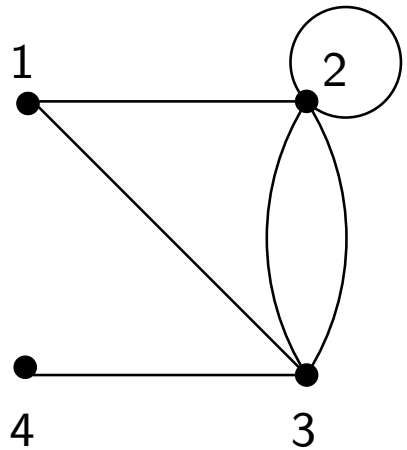
Definition

- Sei $G = (V, E)$ ein ungerichteter Graph mit $V = \{1, \dots, n\}$, so ist die symmetrische Matrix $A \in \mathbb{R}^{n \times n}$ mit
  - $a_{ij} = a_{ji} =$ Anzahl der Kanten, die i mit j, $i \neq j$, verbinden,
  - $a_{ii} = 2*$ (Anzahl der Schleifen, die i enthalten), $i = 1, \dots, n$,

  die **Adjazenzmatrix** von G.
- Wegen der Symmetrie genügt es, die obere Dreiecksmatrix von A zu speichern.
- Hat G Kantengewichte und ist G einfach, so setzt man
  - $a_{ij} =$ Gewicht der Kante, wenn $ij \in E$,
  - $a_{ij} = 0, -\infty, +\infty$, anderenfalls.
- Der Speicheraufwand für Adjazenzmatrizen beträgt n^2 Zellen (bzw. $\frac{n(n-1)}{2}$ für obere Dreiecksmatrix).

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT



Adjazenzmatrix

- Die Adjazenzmatrix dieses Graphen ist:

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Adjazenzmatrix eines gerichteten Graphen

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

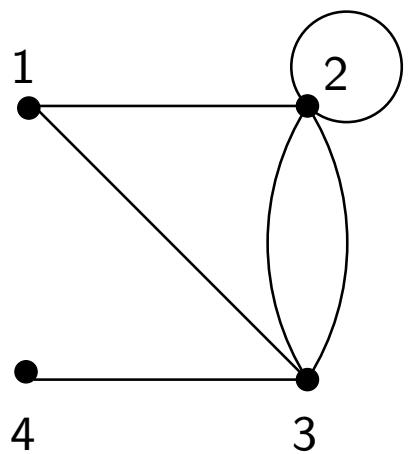
- Die Adjazenzmatrix eines Digraphen $D = (V, A)$ mit $V = \{1, \ldots, n\}$ ohne Schleifen ist definiert durch
  - $a_{ii} = 0$, $i = 1, \ldots, n$,
  - $a_{ij} =$ Anzahl der Bögen in A mit Anfangsknoten i und Endknoten j.
- Bogengewichte einfacher Digraphen werden wie im ungerichteten Fall behandelt.
- Der Speicheraufwand für Adjazenzmatrizen beträgt n^2 Zellen (bzw. $\frac{n(n-1)}{2}$ für obere Dreiecksmatrix).

Adjazenzlisten

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

- Speichert man für einen Graphen $G = (V, E)$ die Anzahl von Knoten und für jeden Knoten seinen Grad und die Namen der Nachbarknoten, so nennt sich eine solche Datenstruktur **Adjazenzliste** von G.



Adjazenzliste

	Knotennummer	Grad	Nachbarknoten
4	1	2	2, 3
	2	4	1, 3, 3, 2
	3	4	1, 2, 2, 4
	4	1	3

- Die Adjazenzliste dieses Graphen ist:

$$4, 1, 2, 2, 3, 2, 4, 1, 3, 3, 2, 3, 4, 1, 2, 2, 4, 4, 1, 3.$$

Bemerkungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Adjazenzlisten für Digraphen

- Bei Digraphen geht man analog vor: nur speichert man lediglich die Nachfolger eines Knotens (jeder Bogen kommt also nur einmal vor).
- Die Speicherung der Adjazenzliste eines Graphen erfordert $2n + 1 + 2m$ Speicherplätze, ihre Speicherung bei einem Digraphen erfordert $2n + 1 + m$ Speicherplätze.

Vor- und Nachteil von Adjazenzlisten

- kompakt, "gut für dünn besetzte Graphen",
- aber schlecht: Suchen, ob Kante ij in G enthalten ist.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Finden von aufspannenden Wäldern

Depth-First Search Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Input/Output

- **Input:** Einfacher Graph $G = (V, E)$ in Form einer Adjazenzliste, d.h. für jeden Knoten v ist eine Nachbarliste $N(v)$ gegeben.
- **Output:** Kantenmenge T (= aufspannender Wald) und Kantenmenge B (mit $B \cup T = E$ und $B \cap T = \emptyset$)

Depth-First Search Algorithmus (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Idee und Ziel des DFS-Algorithmus

- Berechnet für einen Graphen G:
  - die Anzahl der Komponenten,
  - entscheidet, ob G Kreis enthält und
  - findet für jede Komponente einen aufspannenden Baum.
- Tiefensuche: "so schnell wie möglich einen Knoten verlassen"
- rekursives Hinzufügen von Kanten zum Wald T oder zu kreisschließenden Kanten B
- DFS-Algorithmus teilt die Kanten des Graphen in zwei Teilmengen:
  - Kante xy heißt Vorwärtskante, wenn wir im Verfahren von einem markierten Knoten x entlang xy zum Knoten y gehen und y markieren.
    $\rightarrow$ Kante des aufspannenden Waldes
  - Anderfalls heißt xy Rückwärtskante.
    $\rightarrow$ Kante würde Kreis schließen

DFS-Algorithmus im Detail

TECHNISCHE UNIVERSITÄT DARMSTADT

Algorithmus

- Alle Knoten seien unmarkiert.
- Setze $T := \emptyset$, $B := \emptyset$.
- Für alle $v \in V$ führe aus
  - Ist v unmarkiert, dann **CALL SEARCH**(v).
  - (**Bemerkung:** Es existieren genau soviele Zusammenhangskomponenten in G wie hier **SEARCH**(v) aufgerufen wird.)
- Gib T und B aus.

Funktion **SEARCH**(v)

- Markiere v.
- Für alle Knoten $w \in N(v)$ führe aus:
  - Ist w markiert und $vw \notin T$, setze $B := B \cup \{vw\}$.
  - Ist w unmarkiert, setze $T := T \cup \{vw\}$ und **CALL SEARCH**(w).

Eigenschaften des DFS-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Laufzeit

- Laufzeit des Verfahrens: $O(|V| + |E|)$, d.h. polynomiell

Korrektheit

- Die Menge aller Vorwärtskanten ist ein Wald von G, der in jeder Komponente einen aufspannenden Baum bildet.

  $\rightarrow$ Beweis als Übungsaufgabe

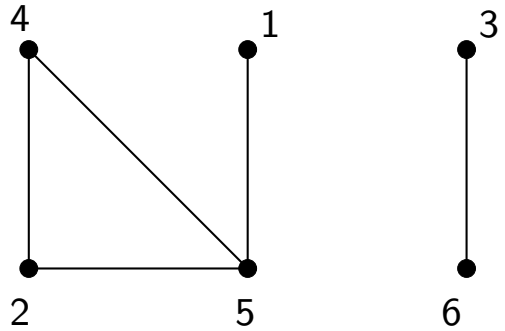
- G enthält einen Kreis $\Leftrightarrow B \neq \emptyset$.

Datenstruktur

- DFS-Algorithmus ist konzipiert für die Datenstruktur einer Adjazenzliste!

Beispiel für DFS-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT



Adjazenzliste (in Tabellenform)

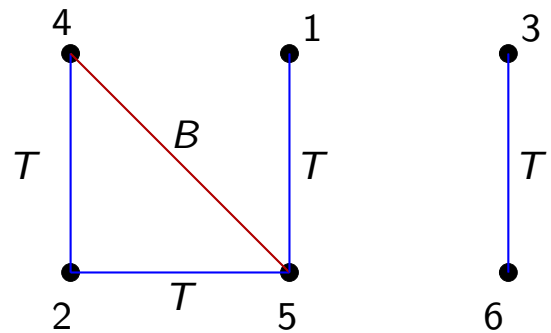
Dieser Graph hat die folgende Adjazenzliste:

	Knotennummer	Grad	Nachbarknoten
6	1	1	5
	2	2	4, 5
	3	1	6
	4	2	2, 5
	5	3	1, 2, 4
	6	1	3

Beispiel für DFS-Algorithmus (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- $1 \rightarrow 5 \rightarrow 2 \rightarrow 4$ $T = \{15, 52, 24\}$
- $4 \rightarrow 2$ $(24 \in T)$
- $4 \rightarrow 5$ $B = \{45\}$
- Knoten 4 erledigt, zurück zu Knoten 2
- $2 \rightarrow 5$ $(52 \in T)$
- Knoten 2 erledigt, zurück zu Knoten 5
- $5 \rightarrow 1$ $(15 \in T)$
- Knoten 5 erledigt, zurück zu Knoten 1, Knoten 1 erledigt, Komponente erledigt
- $3 \rightarrow 6$ $T = \{15, 52, 24, 36\}$
- Knoten 6 erledigt, zurück zu Knoten 3, Knoten 3 erledigt, Komponente erledigt



Finden von gewichts-maximalen Wäldern

Greedy–Max Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Input/Output

- **Input:** Graph $G = (V, E)$ mit Kantengewichten c_e, für alle $e \in E$.
- **Output:** Ein Wald $W \subseteq E$ mit maximalen Gewicht $c(W)$.

Algorithmus

1. (Sortieren)
   $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_K) > 0 \geq c(e_{K+1}) \geq \ldots \geq c(e_{|E|})$.
2. (Initialisierung)
   Setze $W := \emptyset$.
3. ***Für*** $i = 1, \ldots, K$ führe durch:
   Falls $W \cup \{e_i\}$ keinen Kreis enthält,
   setze $W := W \cup \{e_i\}$.
4. (Ausgabe)
   Gib W aus.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Finden von kürzesten Wegen in Graphen

Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

Problemformulierung

- Gegeben sei ein Digraph $D = (V, A)$ mit Bogengewichten $c(a)$ für alle $a \in A$. Weiterhin seien $s, t \in V$ gegeben.
- Gesucht ist ein s-t-Weg minimalen Gesamtgewichts (= Summe der Kantengewichte).

Schwierigkeiten

- negative Gewichte
- Hamiltonische-Wege Problem (jeder Knoten genau einmal besucht)

Unterscheidung der Algorithmen

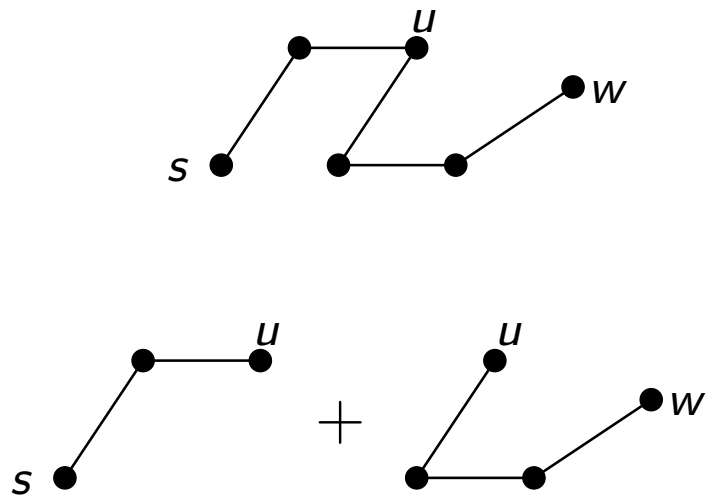
- nur nichtnegative Gewichte
- auch negative Gewichte möglich

Ein Startknoten, nichtnegative Gewichte

TECHNISCHE UNIVERSITÄT DARMSTADT

Idee

- Dekomposition eine kürzesten (s, w)-Weges in einen kürzesten (s, u)-Weg plus kürzesten (u, w)-Weg.



Dijkstra-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Input/Output

- **Input:** Digraph $D = (V, A)$, Gewichte $c(a) \geq 0$ für alle $a \in A$, ein Knoten $s \in V$ (und ein Knoten $t \in V \setminus \{s\}$).
- **Output:** Kürzeste gerichtete Wege von s nach v für alle $v \in V$ und ihre Länge (bzw. einfach nur ein kürzester (s, t)-Weg).

Datenstrukturen

$$\begin{array}{lcl} \text{DIST}(v) & = & \text{Länge des kürzesten } (s, v)\text{-Weges} \\ \text{VOR}(v) & = & \text{Vorgänger von } v \text{ im kürzesten } (s, v)\text{-Weg} \end{array}$$

Initialisierung

$$\begin{array}{lcl} \text{DIST}(s) & = & 0 \\ \text{DIST}(v) & = & +\infty \text{ für alle } v \in V \setminus \{s\} \\ \text{VOR}(v) & = & s \text{ für alle } v \in V \end{array}$$

Dijkstra-Algorithmus im Detail

TECHNISCHE UNIVERSITÄT DARMSTADT

Algorithmus

1. Alle Knoten seien unmarkiert.
2. Sei u unmarkierter Knoten mit $\text{DIST}(u) = \min\{\text{DIST}(v) : v \text{ unmarkiert}\}$. Markiere u. (Falls $u = t$, breche ab.)
3. Für alle unmarkierten Knoten v mit $uv \in A$ führe aus:

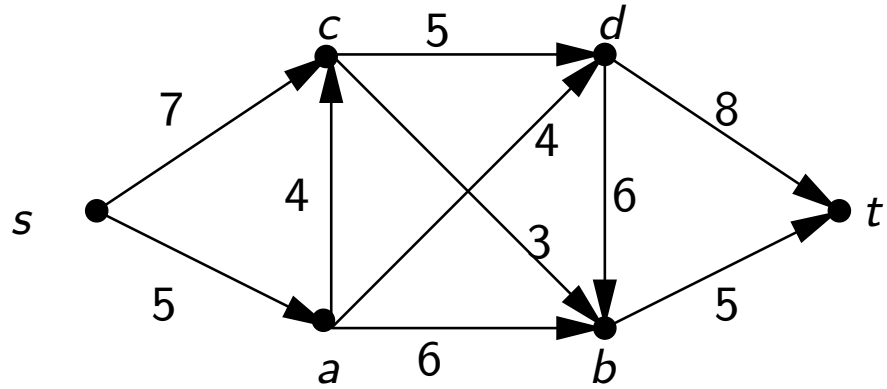
$$\begin{array}{rcl} \text{Falls } \text{DIST}(v) & > & \text{DIST}(u) + c(uv) \text{ setze:} \\ \text{DIST}(v) & := & \text{DIST}(u) + c(uv) \text{ und } \text{VOR}(v) := u. \end{array}$$

4. Sind noch nicht alle Knoten markiert, gehe zu 2.

- Für alle markierten Knoten v ist $\text{DIST}(v)$ die Länge eines kürzesten (s, v)-Weges.
- Ist v markiert und $\text{DIST}(v) < +\infty$, so ist $\text{VOR}(v)$ der Vorgänger von v in einem kürzesten (s, v)-Weg. Durch Rückwärtsgehen bis s kann ein kürzester (s, v)-Weg bestimmt werden.
- Brechen wir das Verfahren nicht in Schritt 2 ab und gilt am Ende $\text{DIST}(v) = +\infty$, so heißt das, dass es in D keinen (s, v)-Weg gibt.

Beispiel für Dijkstra-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT



	$DIST(s)$	$DIST(a)$	$DIST(b)$	$DIST(c)$	$DIST(d)$	$DIST(t)$	*Unmarkiert*
Start	0	∞	∞	∞	∞	∞	$\{s, a, b, c, d, t\}$
Mark. s	0	$5s$	∞s	$7s$	∞s	∞s	$\{a, b, c, d, t\}$
Mark. a	0	5	$11a$	7	$9a$	∞	$\{b, c, d, t\}$
Mark. c	0	5	$10c$	7	9	∞	$\{b, d, t\}$
Mark. d	0	5	10	7	9	$17d$	$\{b, t\}$
Mark. b	0	5	10	7	9	$15b$	$\{t\}$
(Mark. t)							$\emptyset$

Demnach ist $s \rightarrow c \rightarrow b \rightarrow t$ **ein** kürzester Weg von s nach t und hat die Länge 15.

Korrektheit des Dijkstra-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz

Der Dijkstra-Algorithmus arbeitet korrekt.

Beweis

- Über Induktion nach der Anzahl k der markierten Knoten zeigen wir:
  - (a) Ist v markiert, so enthält DIST(v) die Länge eines kürzesten (s, v)-Weges.
  - (b) Ist v unmarkiert, so enthält DIST(v) die Länge eines kürzesten (s, v)-Weges, wobei nur markierte Knoten als innere Knoten zugelassen sind.
- $k = 1$: Dann ist nur s markiert und die Behauptung gilt sicherlich.
- Sei die Behauptung also richtig für $k \geq 1$ markierte Knoten.

Induktionsschritt für (a)

TECHNISCHE UNIVERSITÄT DARMSTADT

Induktionsschritt für (a), Teil 1

- Das Verfahren habe in Schritt 2 einen $(k+1)$-sten Knoten u markiert.
- Nach Ind.-voraussetzung Teil (b) ist DIST(u) die Länge eines kürzesten (s, u)-Weges, der als innere Knoten nur die ersten k markierte Knoten benutzen darf.
- Wir zeigen, dass DIST(u) die Länge eines kürzesten (s, u)-Weges ist.

Induktionsschritt für (a)

TECHNISCHE UNIVERSITÄT DARMSTADT

Induktionsschritt für (a), Teil 2

- Wir zeigen, dass DIST(u) die Länge eines kürzesten (s, u)-Weges ist.
- Angenommen, es existiere ein kürzerer Weg P von s nach u.
  - Dann muß er einen Bogen von einem markierten Knoten zu einem unmarkierten Knoten enthalten.
  - Sei vw der erste derartige Bogen auf dem Weg P.
  - Der Teilweg $\bar{P}$ von P von s nach w benutzt nur markierte Knoten als innere Knoten.
  - Deshalb gilt DIST(w) $\leq c(\bar{P})$ nach Ind.-voraussetzung Teil (b).
  - Weiterhin gilt
    - $c(\bar{P}) \leq c(P)$ (nichtnegative Kantengewichte) und
    - $c(P) <$ DIST(u) (Annahme über P).
  - Zusammen ergibt sich: DIST(w) $<$ DIST(u).
  - Aber dies widerspricht DIST(u) $\leq$ DIST(w) entsprechend der Wahl von u in Schritt 2.

Induktionsschritt für (b)

TECHNISCHE UNIVERSITÄT DARMSTADT

Induktionsschritt für (b)

- Es bleibt zu zeigen: Für die derzeit unmarkierten Knoten v ist DIST(v) die Länge eines kürzesten (s, v)-Weges, der nur markierte Knoten als innere Knoten hat.
- Der vorletzte Knoten auf einem kürzesten (s, v)-Weg ist entweder u oder ein anderer (vor u) markierter Knoten.
- Man beachte jetzt, dass in Schritt 3 die Länge eines (s, v)-Weges über markierte Knoten verschieden von u verglichen wird mit der Länge eines (s, v)-Weges über markierte Knoten, der als vorletzten Knoten den Knoten u enthält.
- Damit folgt die Behauptung. □

Bemerkungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Kürzeste-Wege-Baum

- In der Datenstruktur VOR merken wir uns zu jedem Knoten v seinen Vorgänger in einem kürzesten (s, v)-Weg.
- Einen kürzesten (s, v)-Weg erhält man dann gemäß:

$$v \quad \mathrm{VOR}(v) \quad \mathrm{VOR}(\mathrm{VOR}(v)) \quad \ldots \quad \mathrm{VOR}(\ldots \mathrm{VOR}(v) \ldots) = s$$

- Offensichtlich ist durch VOR( ) eine Arboreszenz definiert.

Satz

- Sei $D = (V, A)$ ein Digraph mit nichtnegativen Bogengewichten und $s \in V$.
- Es gibt eine Arboreszenz B mit Wurzel s, so dass für jeden Knoten $v \in V$, für den es einen (s, v)-Weg in D gibt, der (eindeutig bestimmte) gerichtete Weg in B von s nach v ein kürzester (s, v)-Weg ist.

Anzahl Operationen des Dijkstra-Algorithmus: $O(|V|^2)$

Ein Startknoten, beliebige Gewichte

TECHNISCHE UNIVERSITÄT DARMSTADT

Problem

- Problem, einen kürzesten Weg in einem Digraphen mit beliebigen Bogengewichten zu bestimmen, ist äquivalent zum Problem, einen längsten Weg in einem Digraphen mit beliebigen Gewichten zu finden.
- Wäre dies in polynomialer Zeit lösbar, so könnte man auch das $\mathcal{NP}$-vollständige Problem in polynomialer Zeit lösen, zu entscheiden, ob ein Digraph einen gerichteten hamiltonischen Weg hat.

Vergrößerung der Problemklasse

- Dijkstra-Algorithmus funktioniert nur für nichtnegative Gewichte.
- Betrachte jetzt **azyklische** Digraphen mit **beliebigen** Kantengewichten.

Moore-Bellman-Algorithmus für azyklische Digraphen

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz, Beweis → Übungsaufgabe

- Sei $D = (V, A)$ ein azyklischer Digraph.
- Dann können die Knoten in V so numeriert werden, dass $(u, v) \in A \Rightarrow u < v$.

Input/Output

- **Input:** Azyklischer Digraph $D = (V, A)$, Gewichte $c(a)$ für alle $a \in A$ (auch negative Gewichte sind zugelassen), ein Knoten $s \in V$.

  O.B.d.A. nehmen wir an, dass $V = \{1, 2, \ldots, n\}$ gilt und alle Bögen die Form (u, v) mit $u < v$ haben.
- **Output:** Kürzeste gerichtete Wege von s nach v für alle $v \in V$ und ihre Länge

Datenstrukturen

DIST(v)	=	Länge des kürzesten (s, v)-Weges
VOR(v)	=	Vorgänger von v im kürzesten (s, v)-Weg

Moore-Bellman-Algorithmus im Detail

TECHNISCHE UNIVERSITÄT DARMSTADT

Algorithmus

$$\begin{aligned} \text{DIST}(s) &= 0 \\ \text{DIST}(v) &= +\infty \text{ für alle } v \in V \setminus \{s\} \\ \text{VOR}(v) &= s \text{ für alle } v \in V \end{aligned}$$

FOR $v = s + 1$ TO n DO

FOR $v = s$ TO $v - 1$ DO

Falls $(u, v) \in A$ und $\text{DIST}(u) + c(uv) < \text{DIST}(v)$
setze $\text{DIST}(v) := \text{DIST}(u) + c(uv)$ und $\text{VOR}(v) := u$.

- Gilt am Ende $\text{DIST}(v) = +\infty$, so heißt das, dass es in D keinen (s, v)-Weg gibt.
- Für alle Knoten v mit $\text{DIST}(v) < +\infty$ ist $\text{DIST}(v)$ die Länge eines kürzesten (s, v)-Weges.
- Ist $\text{DIST}(v) < +\infty$, so ist $\text{VOR}(v)$ Vorgänger von v in einem kürzesten (s, v)-Weg. Durch Rückwärtsgehen bis s kann ein kürzester (s, v)-Weg bestimmt werden.

Anzahl Operationen des Moore-Bellman-Algorithmus: $O(|V|^2)$

Korrektheit des Moore-Bellman-Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Satz

Der Moore-Bellman-Algorithmus funktioniert für beliebige azyklische Digraphen mit beliebigen Gewichten.

Beweis

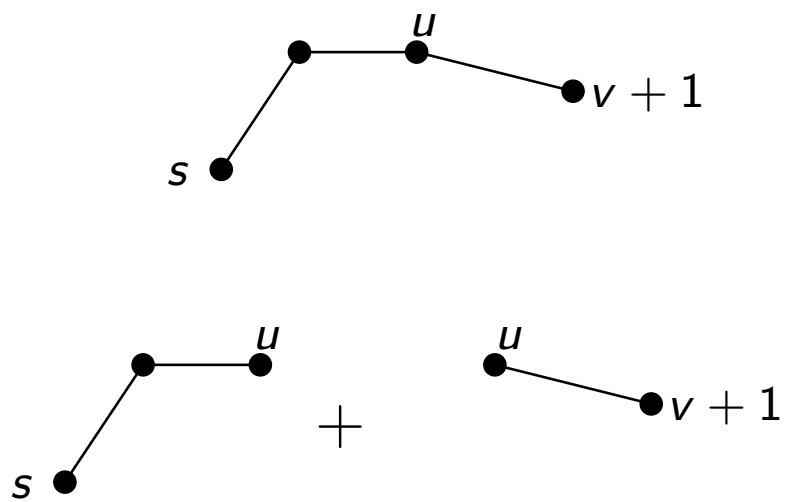
- Nach Voraussetzung haben alle Bögen in D die Form (u, v) mit $u < v$.
- Deshalb enthält jeder (s, v)-Weg mit $v > s$ als innere Knoten nur solche Knoten u mit $s < u < v$.
- Durch Induktion nach $v = s, \ldots, n$ zeigen wir, dass DIST(v) die Länge eines kürzesten (s, v)-Weges ist.
- $v = s$ o.k.
- Sei die Behauptung richtig für alle Knoten $s, \ldots, v$.

Induktionsschritt

TECHNISCHE UNIVERSITÄT DARMSTADT

Induktionsschritt

- Betrachte den Knoten $v+1$.
- Ein kürzester $s, v+1$-Weg besteht entweder nur aus dem Bogen $(s, v+1)$ (falls vorhanden) oder aber er führt zu einem Knoten $s+1 \leq u \leq v$ und benutzt anschließend die Kante $u, v+1$.
- Das Minimum all dieser Wege wird jedoch in der innersten Schleife berechnet. □



Yen-Variante

TECHNISCHE UNIVERSITÄT DARMSTADT

Bemerkungen

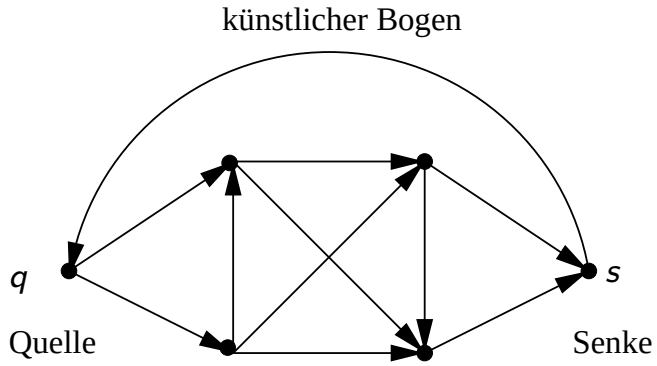
- Variante des Moore-Bellman-Algorithmus, welche für beliebige Gewichte und beliebige Digraphen funktioniert, sofern es keinen gerichteten **negativen** Kreis gibt.
- Laufzeit $O(|V|^3)$
- **Wichtig:** Mit dem Verfahren kann man tatsächlich testen, ob D einen negativen gerichteten Kreis enthält! Ein solcher wird in **polynomialer Zeit** gefunden.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Finden maximaler Flüsse in Netzwerken

Maximalfluss-Problem

TECHNISCHE UNIVERSITÄT DARMSTADT



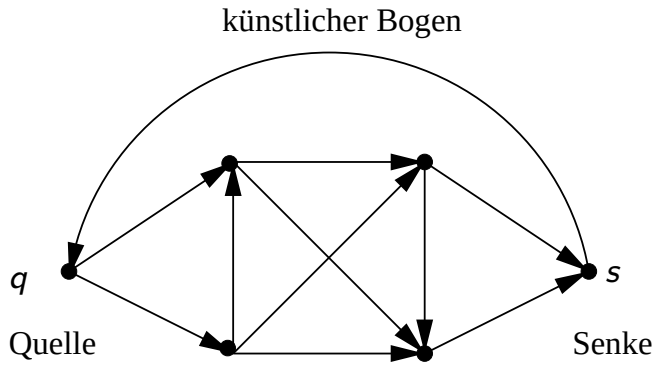
Problemformulierung

- **Gegeben** sei ein Digraph $D = (V, A)$ mit Bogenkapazitäten d_a für alle $a \in A$. Weiterhin mögen ein Knoten $q \in V$ ohne eingehende Bögen (**Quelle**) und ein Knoten $s \in V$ ohne abgehende Bögen (**Senke**) existieren.
  Letztendlich setzen wir $\bar{A} = A \cup \{sq\}$ und wählen $d_{sq} \geq \sum_{a \in A} d_a$ beliebig.
- Ein Vektor $x = (x_a : a \in \bar{A})$ heißt **Fluß** auf D, falls für jeden Knoten $v \in V$ gilt

$$\sum_{wv \in \bar{A}} x_{wv} = \sum_{vw \in \bar{A}} x_{vw} \quad \text{(Flusserhaltung)}.$$

Maximalfluss-Problem (2)

TECHNISCHE UNIVERSITÄT DARMSTADT



Problemformulierung (2)

- Der **Wert** des Flusses ist gleich x_{sq}.
- **Problemstellung:** Man finde einen Fluss $x \geq 0$ über D maximalen Wertes, der auch die Kapazitäten auf den Bögen respektiert, also $x_a \leq d(a)$ für alle $a \in \bar{A}$ erfüllt.

Max-Flow-Min-Cut Theorem

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition

Sei $S \subseteq V$ mit $q \in S$ und $s \in V \setminus S$. Dann heißt S Schnitt mit Kapzität

$$\text{cap}(S) = \sum_{vw \in A, v \in S, w \in V \setminus S} d_{vw}.$$

Minimaler Schnitt

- Kapazität eines jeden Schnittes liefert obere Schranke für maximalen Fluss.
- Finde Schnitt minimaler Kapazität.

Max-Flow-Min-Cut Theorem

- Es existiert ein gültiger Fluss $\bar{x} \leq d$ und ein Schnitt $S \subseteq V$ mit $\bar{x}_{sq} = \text{cap}(S)$.
- Beide sind Optimallösungen der jeweiligen Probleme.

$\rightarrow$ Dualitätstheorie in linearer Optimierung

Formulierung als lineares Optimierungsproblem

TECHNISCHE UNIVERSITÄT DARMSTADT

Wiederholung

Für einen Digraphen $D = (V, A)$ ist die Knoten-Kanten-Inzidenzmatrix definiert durch:

- $M_{ij} = 1$, falls Bogen j Knoten i verlässt,
- $M_{ij} = -1$, falls Bogen j in Knoten i endet,
- $M_{ij} = 0$, sonst.

Lemma → Max-Flow Problem als lineares Optimierungsproblem

$x \geq 0$ ist ein Fluss, genau dann wenn $Mx = 0$. → $\min\{x_{sq} : Mx = 0, 0 \leq x \leq d\}$

Beweis → Betrachte Knoten i

$$\text{(eingehende Bögen)} \sum_{vi \in \bar{A}} x_{vi} = \sum_{iw \in \bar{A}} x_{iw} \text{ (ausgehende Bögen)}$$

$$0 = \sum_{iw \in \bar{A}} 1 \cdot x_{iw} + \sum_{vi \in \bar{A}} (-1) \cdot x_{vi} + \sum_{vw \in \bar{A}, v, w \neq i} 0 \cdot x_{vw} = M_{i.}x$$

Lösung des Max-Flow Problems

TECHNISCHE UNIVERSITÄT DARMSTADT

Idee des Ford-Fulkerson Algorithmus

Verbessere Fluss sukszessive durch **augmentierende Pfade**.

Definition

- Sei $D = (V, A)$ ein Digraph mit $q, s \in V$ und $qs \in \bar{A}$, d ein Kapazitätsvektor und x ein gültiger Fluss für d.
- Ein (nicht notwendigerweise gerichteter) Pfad $q = v_0 e_1 v_1 e_2 v_2 \ldots e_k v_k$ heißt **ungesättigt** (bzgl. x und d), falls
  - $e_i \in A$,
  - $x_{e_i} < d_{e_i}$ für alle Vorwärtsbögen,
  - $x_{e_j} > 0$ für alle Rückwärtsbögen,
- Falls $v_0 = q$ und $v_k = s$, so heißt dieser Pfad ein **augmentierender Pfad**.

Ford-Fulkerson Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Konstruktion eines besseren Flusses aus augmentierendem Pfad

- Sei $\epsilon = \min_i\{d_{e_i} - x_{e_i}, x_{e_i}\}$.
- Besserer Fluss als x:

$$x_e = \begin{cases} x_e + \epsilon & \text{falls } e = sq, \\ x_e + \epsilon & \text{falls } e \text{ Vorwärtsbogen im Pfad ist,} \\ x_e - \epsilon & \text{falls } e \text{ Rückwärtsbogen im Pfad ist,} \\ x_e & \text{falls } \textit{sonst}. \end{cases}$$

Ford-Fulkerson Algorithmus

- Initialisiere $x = 0$.
- Solange ein augmentierender Pfad existiert, verbessere x.

Wie findet man augmentierenden Pfad?

TECHNISCHE UNIVERSITÄT DARMSTADT

Hilfsgraph $D' = (V', A')$

- $V' = V$
- $vw \in A'$ mit Kapazität $d_{vw} - x_{vw}$, falls $d_{vw} - x_{vw} > 0$
- $wv \in A'$ mit Kapazität x_{vw}, falls $x_{vw} > 0$

Bemerkung

- Für jeden Bogen in D sind im Hilfsgraphen D' ein bzw. zwei Bögen jenachdem, ob $d_{vw} - x_{vw} > 0$ und $x_{vw} > 0$ gilt.
- Jeder augmentierende Pfad in D (bzgl. x und d) entspricht einem gerichteten q-s-Weg in D'.
- Ob ein gerichteter q-s-Weg in D' existiert, kann mit dem DFS-Algorithmus entschieden werden.