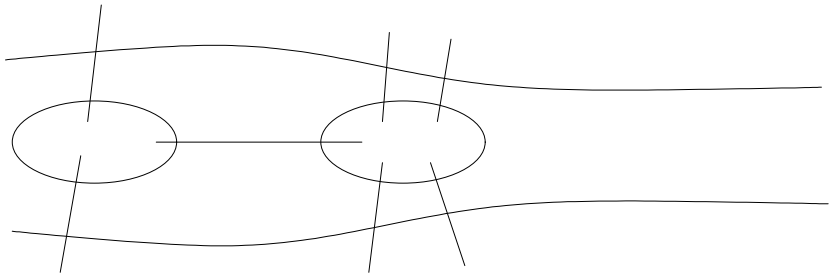
3 Graphen - Grundlagen

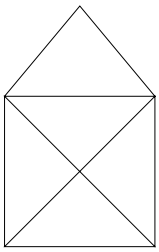
Einführende Beispiele

Beispiel 3.1 Königsberger Brückenproblem (Euler, 1736)



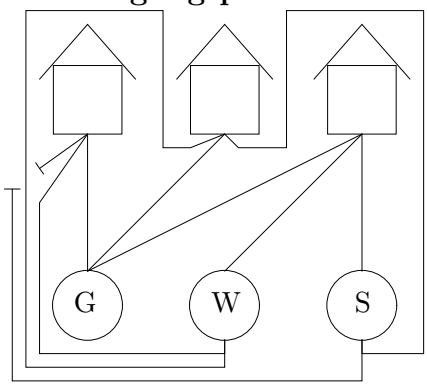
Finde eine Rundreise durch Königsberg, so dass jede Brücke genau einmal durchlaufen wird. Eine solche Rundreise gibt es nicht!

Beispiel 3.2 Das Haus vom Nikolaus



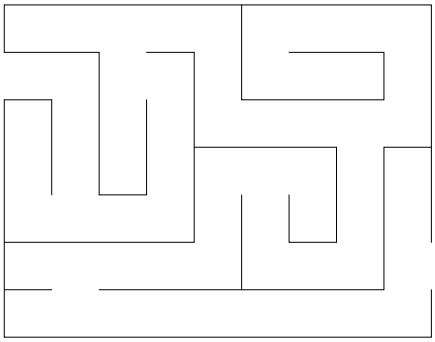
Kann man das Haus vom Nikolaus zeichnen ohne den Stift abzusetzen?

Beispiel 3.3 Ein Grundversorgungsproblem



Verbinde alle 3 Häuser mit Gas, Wasser und Strom, ohne dass sich zwei Verbindungen kreuzen.

Beispiel 3.4 Labyrinthe



Finde einen Weg vom Start zum Ziel durch das Labyrinth.

Definitionen

In diesem Kapitel wollen wir wichtige graphentheoretische Begriffe und Bezeichnungen zusammenstellen. Wir stellen diese in einer Form dar, wie sie im Weiteren benutzt werden sollen.
Ein **(ungerichteter) Graph** ist ein Tripel (V, E, ψ) bestehend aus einer nichtleeren Menge V, einer Menge E und einer **Inzidenzfunktion** $\psi : E \mapsto V^{(2)}$. Dabei bezeichnet $V^{(2)}$ die Menge der ungeordneten Paare von (nicht notwendigerweise verschiedenen) Elementen aus V. Ein Element aus V heißt **Knoten**, ein Element aus E heißt **Kante**. Die Funktion ψ weist also jeder Kante e ein Paar von Knoten u und v zu durch $\psi(e) = uv = vu$. Häufig schreibt man in der Literatur auch $[u, v]$ oder $\{u, v\}$ anstelle von uv. Wir werden soweit möglich immer die Schreibweise uv verwenden, gegebenfalls greifen wir auf das Klammersymbol $[u, v]$ zurück. Ein Graph heißt **endlich**, falls V und E endlich sind, andernfalls heißt G **unendlich**. Wir beschäftigen uns im Rahmen dieser Vorlesung nur mit endlichen Graphen, so dass wir von nun auf das Präfix "endlich" verzichten.

Ist $\psi(e) = uv$ für eine Kante $e \in E$, dann nennen wir $u, v \in V$ **Endknoten** von e. Wir sagen, u und v sind **inzident zu** e beziehungsweise **liegen auf** e, die Kante e **verbindet** u und v, und die Knoten u und v sind **Nachbarn** beziehungsweise **adjazent**. Dagegen verwenden wir für zwei Kanten, die den selben Endknoten haben, sowohl den Begriff **inzident** als auch **adjazent**. Eine Kante e mit $\psi(e) = vv$ heißt **Schlinge**, und Kanten $e, f \in E$ mit $\psi(e) = \psi(f)$ heißen **parallel**. Ein Graph, der weder Schlingen noch parallele Kanten enthält, heißt **einfach**.

Neben ungerichteten Graphen werden auch gerichtete Graphen betrachtet. Ein **gerichteter Graph** oder **Digraph** $D = (V, A, \psi)$ besteht aus einer (endlichen) nicht-leeren Menge V, einer (endlichen) Menge A von **Bögen** und einer Inzidenzfunktion $\psi : A \mapsto V \times V$. Jeder Bogen $a \in A$ ist ein geordnetes Paar von Knoten, also $a = (u, v)$ für $u, v \in V$. Sprechen wir im folgenden von Graphen, so meinen wir immer einen ungerichteten Graphen. In dem Fall, dass wir uns mit gerichteten Graphen befassen, werden wir immer explizit das Präfix "gerichtet" mitführen.

Die Verwendung der Inzidenzfunktion ψ führt zu einem ziemlich aufwendigen Formalismus. Wir wollen soweit als möglich auf die Inzidenzfunktion verzichten. Wir schreiben von nun an $e = uv$ anstelle von $\psi(e) = uv$. Sofern der Graph einfach ist, ist diese Schreibweise korrekt. Gibt es mehrere zu e parallele Kanten und sprechen wir von einer Kante uv, so meinen wir irgendeine der parallelen Kanten. Manchmal wird es notwendig sein zwischen verschiedenen parallelen Kanten zu unterscheiden, dann greifen wir auf die Inzidenzfunktion ψ zurück. Ansonsten bezeichne von nun das Tupel $G = (V, E)$ einen Graphen.

Für eine Kantenmenge $F \subseteq E$ bezeichnen wir mit $V(F)$ die Menge aller Knoten, die zu einer Kante in F inzident sind, und umgekehrt bezeichnen wir für eine Knotenmenge $W \subseteq V$ mit $E(W)$ die Menge aller Kanten, für die beide Endknoten in W sind. Für zwei Knotenmengen $U, W \subseteq V$ bezeichnen wir mit $[U : W]$ beziehungsweise $\delta(U, W)$ die Menge der Kanten mit einem Endknoten in U und einem Endknoten in W. Anstelle von $\delta(W, V \setminus W) = [W : V \setminus W]$ schreiben wir auch kurz $\delta(W)$. Ebenso kürzen wir $\delta(\{v\})$ für ein $v \in V$ mit $\delta(v)$ ab. Eine Kantenmenge $F \subseteq E$ heißt **Schnitt**, wenn es eine Knotenmenge $W \subseteq V$ gibt mit $F = \delta(W)$. Wir nennen $\delta(W)$ auch den **von W induzierten Schnitt**. Wollen wir betonen, dass ein Schnitt $\delta(W)$ zwei Knoten $s \in W$ und $t \in V \setminus W$ trennt, so sprechen wir von einem **$[s, t]$-Schnitt**. Wir nennen nicht-leere Knotenmengen $V_1, \ldots, V_p$ eine **Partition** von V, falls die Vereinigung der Knotenmengen V ergibt und die Knotenmengen paarweise disjunkt sind.
Im Falle eines gerichteten Graphen $D = (V, A)$ bezeichne $\delta^+(W) := \{(i, j) \in A : i \in W, j \in V \setminus W\}$ die Menge der ausgehenden Bögen und $\delta^-(W) := \{(i, j) \in A : i \in V \setminus W, j \in W\}$ die Menge der eingehenden Bögen. Entsprechend ist $\delta^+(v)$ und $\delta^-(v)$ für $v \in V$ definiert.

Sei $F \subseteq E$ eine Kantenmenge von G. Wir bezeichnen mit $d_F(v) = |\delta(v) \cap F|$ den **Grad eines Knoten v bezüglich der Kantenmenge F**. Anstelle von $d_E(v)$ schreiben wir oft kurz $d(v)$ und sprechen vom Grad des Knoten v. Ist $d(v) = 0$, so nennen wir v einen **isolierten Knoten**. Ein Knoten v heißt **gerade** (**ungerade**), falls $d(v)$ gerade (ungerade) ist.

Wir nennen einen Graphen $G = (V, E)$ **vollständig**, falls $uv \in E$ für alle $u, v \in V$. Manchmal bezeichnen wir den vollständigen Graphen mit K_n, wenn n die Kardinalität der Knotenmenge ist. Sprechen wir vom vollständigen Graphen G, so gehen wir immer davon aus, dass G einfach ist. Ein Graph, dessen Knotenmenge V in zwei disjunkte nicht-leere Teilmengen V_1, V_2 mit $V_1 \cup V_2 = V$ eingeteilt werden kann, so dass keine zwei Knoten in V_1 und keine zwei Knoten in V_2 benachbart sind, heißt **bipartit**. Falls $uv \in E$ für alle $u \in V_1$, $v \in V_2$ so spricht man von einem **vollständig bipartiten** Graphen. Manchmal bezeichnen wir diesen (bis auf Isomorphie) eindeutig bestimmten Graphen mit $K_{m,n}$, wobei $m = |V_1|$ und $n = |V_2|$. Der **Kantengraph** von G ist der Graph, dessen Knotenmenge die Kantenmenge von G ist, und in dem zwei Knoten genau dann adjazent sind, wenn die zugehörigen Kanten in G inzident sind.

Betrachten wir mehrere Graphen gleichzeitig, so versehen wir jedes der eingeführten Symbole mit dem Index "G", also zum Beispiel $V_G(F)$, $\delta_G(v)$ etc.

Eine endliche Folge $K = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$, $k \geq 0$, in einem Graphen heißt **Kette**, falls die Folge mit einem Knoten beginnt und endet, die Knoten und Kanten alternierend auftreten und jede Kante e_i mit den beiden Knoten v_{i-1} und v_i inzidiert. Der Knoten v_0 heißt **Anfangsknoten**, der Knoten v_k **Endknoten**, und die Knoten $v_1, \ldots, v_{k-1}$ **innere Knoten**. Wir nennen K auch eine $[v_0, v_k]$**-Kette**. Ist der zugrundeliegende Graph gerichtet, so nennt man K eine **gerichtete Kette**, falls alle Bögen e_i von der Form (v_{i-1}, v_i) sind. Die Zahl k heißt die **Länge** der Kette. Jede Kante, die zwei Knoten aus K verbindet und nicht aus K ist, heißt **Diagonale**. Eine Kette heißt **geschlossen**, falls ihre Länge positiv ist und Anfangs- und Endknoten übereinstimmen.

Eine Kette, in der alle Knoten voneinander verschieden sind, heißt **Weg**. Beachte, dass in einem Weg auch alle Kanten voneinander verschieden sind. Sind in einer Kette nur alle Kanten verschieden, so sprechen wir von einem **Pfad**. Wie bei Ketten sprechen wir von $[u, v]$-Wegen und $[u, v]$-Pfaden. In einem gerichteten Graphen heißt ein Weg oder Pfad, der eine gerichtete Kette ist, ein **gerichteter Weg** oder **gerichteter Pfad**. Ist C eine geschlossene Kette, deren Knoten alle verschieden sind, so nennen wir C einen **Kreis**. Ein Pfad in G, der jede Kante in G enthält, heißt **Eulerpfad**. Ein geschlossener Eulerpfad heißt **Eulertour**. Ein Graph, der eine Eulertour enthält, heißt **eulersch**. Es ist nicht schwer einzusehen, dass ein Graph genau dann eulersch ist, wenn jeder Knoten einen geraden Grad hat, siehe Übung. Ein (gerichteter) Kreis (Weg) der Länge $|V|$ (bzw. $|V| - 1$) heißt (gerichteter) **Hamiltonkreis (Hamiltonweg)**. Ein (gerichteter) Graph, der einen Hamiltonkreis enthält, heißt **hamiltonsch**. Oft sagt man statt Hamiltonkreis einfach **Tour**.

Wir interessieren uns im Rahmen der Vorlesung meist für die Kantenmenge einer Kette, eines Weges oder Kreises. Wir werden daher von nun an eine Kette, einen Weg oder einen Kreis K immer als Kantenmenge, also $K \subseteq E$, auffassen. Die zugehörigen Knoten erhalten wir über $V(K)$.

Jeder Graph $G = (V, E)$ kann in die Ebene gezeichnet werden, indem man jeden Knoten durch einen Punkt in der Ebene repräsentiert und jeder Kante eine Kurve (gerade Linie) zuordnet, die diejenigen Punkte verbindet, die die beiden Endknoten der Kante repräsentieren. Ein Graph heißt **planar**, falls er in die Ebene gezeichnet werden kann, so dass sich zwei Kanten (genauer, die sie repräsentierenden Kurven) höchstens in ihren Endknoten schneiden. Eine solche Darstellung eines planaren Graphen in der Ebene heißt **Einbettung** von G in der Ebene. Offensichtlich kann es zu einem Graphen G verschiedene Einbettungen geben. Wir gehen in dieser Vorlesung immer davon aus, dass wir für einen planaren Graphen implizit eine Einbettung gegeben haben. Gibt es verschiedene Einbettungen, so

meinen wir eine beliebige Einbettung, sofern wir diese nicht genauer spezifizieren. Wir unterscheiden also nicht zwischen einem planaren Graphen und seiner Einbettung.

Ein planarer Graph (genauer gesagt, die Einbettung des planaren Graphen) teilt die Ebene in eine Menge von Flächen $\mathcal{F}_G$ ein. Diejenige Fläche, die den gesamten Graphen umschließt, heißt **äußere Fläche**. Zu jeder Fläche $F \in \mathcal{F}_G$, die nicht die äußere Fläche ist, gibt es genau einen Kreis $C_F \subseteq E$, der diese Fläche umschließt. Ein solcher Kreis heißt **Flächenkreis**. Ebenso gibt es eine Kantenmenge in G, die die äußere Fläche umschließt. Führen wir einen Knoten für jede Fläche aus $\mathcal{F}_G$ ein und Kanten zwischen Knoten, deren zugehörige Flächen benachbart sind (genauer gesagt, deren zugehörige Flächenkreise eine gemeinsame Kante haben), so erhalten wir einen neuen Graphen, den sogenannten **dualen Graphen** von G (in Zeichen $G^* = (V^*, E)$). Beachte, dass jede Kante im dualen Graphen genau einer Kante in G entspricht, so dass wir für beide Graphen die gleiche Kantenmenge E verwenden. In diesem Zusammenhang heißt G der **primale Graph**.

Exkurs: Algorithmen

Im Laufe der Vorlesung wollen wir unterscheiden zwischen einfachen und schweren Problemen, "guten" und "schlechten" Algorithmen. Um mit der Komplexität der Lösung von Problemen umgehen zu können, müssen wir formalisieren, was wir unter einem Problem oder Algorithmus verstehen. In diesem Abschnitt skizzieren wir die wichtigsten Konzepte der Komplexitätstheorie, die notwendig sind, die Vorlesung zu verstehen, ohne dabei zu sehr ins Detail zu gehen. Weiterführende Literatur hierzu findet sich zum Beispiel in [2].
Ein *Problem* ist eine Fragestellung mit offenen Parametern und eine Spezifikation, wie eine Lösung aussehen soll.
Zum Beispiel sind beim Traveling Salesman Problem (TSP) die offenen Parameter ein Graph $G = (V, E)$ und Kantengewichte $c_{ij} \in E$. Die Fragestellung lautet: Entscheide, ob G eine Tour enthält. Falls ja, finde eine Tour minimalen Gewichts. Eine Lösung des TSP wäre entweder (a) ein Zertifikat, dass es keine Lösung gibt, oder (b) eine Tour minimalen Gewichts.
Wenn alle Parameter spezifiziert sind, so spricht man von einem *Problembeispiel*. Die „Größe" eines Problembeispiels wird angegeben durch Verwendung eines Kodierschemas. Wir verwenden die Binärkodierung. Mit $\langle n \rangle$ bezeichnen wir die Kodierungslänge (Größe) einer Zahl $n \in \mathbb{Z}$.

$$\begin{aligned} \langle n \rangle &= \lceil \log_2 |n| + 1 \rceil + 1 \qquad n \in \mathbb{Z} \\ \langle r \rangle &= \langle p \rangle + \langle q \rangle \qquad r \in \mathbb{Q},\, r = \frac{p}{q},\, p, q \text{ teilerfremd} \end{aligned}$$

Die *Kodierungslänge* eines Graphen $G = (V, E)$ mit Kantengewichten $c_e \in E$ ist definiert durch:

$$\langle G \rangle = |V| + |E| + \sum_{e \in E} \langle c_e \rangle$$

Ein *Algorithmus* ist eine Anleitung zur schrittweisen Lösung eines Problems.
Wir sagen ein Algoritmus A *löst* ein Problem Π, falls für alle Problembeispiele $I \in \Pi$, A eine Lösung in einer *endlichen* Anzahl an Schritten findet. Algorithmen werden wir messen bzgl. ihrer *Laufzeit*, d.h. die Anzahl der Schritte, die notwendig zur Lösung des Problems sind.
Ein *Schritt* ist eine *elementare Operation*. Addieren, Subtrahieren oder Vergleichen sind solche Operationen. Multiplikation bzw. Division können hinzugenommen werden, wenn garantiert werden kann, dass die Größe der Zahlen, die auftreten, polynomial in der Größe des Inputs bleiben.
Sei $T(l)$ die Laufzeit (die Anzahl der Schritte) eines Algorithmus A zur Lösung eines Problembeispiels mit Kodierungslänge höchstens $l \in \mathbb{N}$.

Beispiel 3.5 TSP:

$$V = \{1, \ldots, l_v\}, \qquad c_e, \qquad e \in E, \qquad \{(i, j) | i, j \in V\}$$

$$\sum_{i=1}^{l_v} \langle i \rangle + \sum_{i,j} (\langle i \rangle + \langle j \rangle) + \sum_{i,j} \langle c_{ij} \rangle = l$$

Ein Algorithmus läuft in *Polynomialzeit*, falls es ein Polynom p gibt mit $T(l) \leq p(l)$, für alle $l \in \mathbb{N}$.
Die Klasse von Problemen, für die es einen polynomialen Algorithmus gibt, bezeichnen wir mit $\mathcal{P}$.
Schließlich wollen wir noch einen Begriff einführen, der in der Vorlesung wichtig sein wird.
Ein Algorithmus heißt *streng polynomial*, falls seine Laufzeit nur von der Anzahl der Eingabedaten abhängt, z.B. im Falle von kürzesten Wegen nur von $|V|$ und $|E|$, aber nicht von c_e, $e \in E$.

Definition 3.6 Größenordnung von Funktionen/Laufzeiten:
Sei $M = \{f | f : \mathbb{N} \to \mathbb{R}\}$ die Menge der reellwertigen Funktionen.
Für $g \in M$ sei

$$\begin{aligned} \mathcal{O}(g) &= \{f \in M : \exists c, n_0 \in \mathbb{N} : f(n) \leq cg(n), \forall n \geq n_0\} \\ \otimes(g) &= \{f \in M : \exists c, n_0 \in \mathbb{N} : f(n) \geq cg(n), \forall n \geq n_0\} \\ \Theta(g) &= \mathcal{O}(g) \cap \otimes(g). \end{aligned}$$

Speicherung von Graphen

Es gibt eine Vielzahl von Möglichkeiten, Graphen in Computerprogrammen zu speichern. In diesem Abschnitt diskutieren wir einige der gängigsten Methoden. Mehr Informationen zu diesem Thema sind in Aho, Hopcroft, Ullman (1974) [1] zu finden.

Kanten-Bogen-Liste

Ist $G = (V, E)$. Die Bogenliste ist dann:

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

wobei

$$\begin{aligned} a_i &= \text{Anfangsknoten von Kante } i \\ e_i &= \text{Endknoten von Kante } i \end{aligned}$$

Vorteil: Effiziente Speicherung $\mathcal{O}(m)$
Nachteil: Schlechter Zugriff $\mathcal{O}(m)$

Adjazenzmatrizen

Die (symmetrische) $n \times n$-Matrix $A = (a_{ij})$ mit

$$\begin{aligned} a_{ij} &= \text{Anzahl Kanten, die i und j verbinden } (i \neq j) \\ a_{ii} &= 2 \times \text{Anzahl der Schleifen an Knoten } i \end{aligned}$$

heißt *Adjazenzmatrix* von G.

Vorteil: schneller Zugriff $\mathcal{O}(1)$
Nachteil: Speicherung $\mathcal{O}(n^2)$

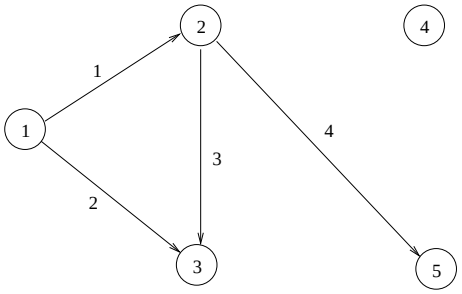
Adjazenzlisten

Für $G = (V, E)$ speichert man :

- Anzahl Knoten
- Anzahl Kanten/Bögen
- Für jeden Knoten seinen Grad und die Liste seiner inzidenten Kanten/ Bögen oder seiner benachbarten Knoten (eventuell mit Kosten) (oder weitere Varianten)

Speicheraufwand: $\mathcal{O}(n + m)$
Zugriff: $\mathcal{O}(n)$
Obiger Speicheraufwand von $\mathcal{O}(n+m)$ ist nur möglich mit Hilfe der dynamischen Speicherallocierung. Ansonsten wäre quadratischer Speicherplatz notwendig (Aus welchem Grund?). Folgende Datenstrukturen erlauben ebenfalls eine Speicherung der Größe $\mathcal{O}(n+m)$ ohne die Verwendung von dynamischer Allocierung. Betrachten wir folgendes Beispiel:

Beispiel 3.7 (Speicherung von Graphen)



Knoten	1	2	3	4	5
fstt	1	3	-1	-1	-1
fsth	-1	1	2	-1	4

Bögen	1	2	3	4
nxth	-1	3	-1	-1
nxtt	2	-1	4	-1

Dabei steht `fst` für *first*, `nxt` für *next*, `t` für *tail* und `h` für *head*. Beispielsweise bezeichnet also `fstt[v]` den ersten Bogen, dessen *tail* an dem Knoten v anliegt.

Folgende Unterroutine in `C` initialisiert die Datenstrukturen `fstt, fsth, nxtt` und `nxth` in $O(m)$, gegeben die Arrays `tail` und `head`.

```
#define NOEDGE   -1  /* definition for a non-edge */
#define m         4  /* maximal nr of edges */
#define n         5  /* maximal nr of nodes */

void init_adj_list (int *tail, int *head, int *fstt,
                    int *nxtt, int *fsth, int *nxth)
{
   int prein[n]; /* previous incoming edge */
   int preou[n]; /* previous outgoing edge */
   int e, f, v;

   for (v = 0; v < n; v++) {
       prein[v] = NOEDGE;
       preou[v] = NOEDGE;
   }

   for (e = 0; e < m; e++) {
       v = tail[e];
       f = preou[v];
       if ( f == NOEDGE ) fstt[v] = e;
       else               nxtt[f] = e;
       preou[v] = e;

       v = head[e];
       f = prein[v];
       if( f == NOEDGE ) fsth[v] = e;
       else              nxth[f] = e;
       prein[v] = e;
   }
```

```
    for (v = 0; v < n; v++) {
        e = preou[v];
        if( e == NOEDGE ) fstt[v] = NOEDGE;
        else              nxtt[e] = NOEDGE;

        e = prein[v];
        if( e == NOEDGE ) fsth[v] = NOEDGE;
        else              nxth[e] = NOEDGE;
    }

    return;
}
```

Suchalgorithmen auf Graphen

Algorithmen, die einen Graphen nach bestimmten Eigenschaften durchsuchen, sind in der Graphentheorie von zentraler Bedeutung. Einige Anwendungen:

- Anzahl der Zusammenhangskomponenten eines Graphen bestimmen
- alle Knoten in einem Digraphen finden, die von einem bestimmten Knoten aus erreichbar sind
- alle Knoten in einem Digraphen finden, von denen aus ein bestimmter Knoten erreichbar ist
- Bestimmung gerichteter Kreise in einem Digraphen
- topologische Sortierung der Knoten in einem azyklischen Digraphen, siehe dazu auch Kapitel 3.1

Im folgenden besprechen wir die zwei einfachsten, aber wichtigen Suchalgorithmen.

3.0.1 Breadth-First-Search

Breadth-First-Search arbeitet nach dem FIFO-Prinzip (First-In-First-Out).

Algorithmus 3.8 (BFS)

INPUT: Graph $G = (V, E)$, Liste L
OUTPUT: Anzahl der Zusammenhangskomponenten und die Menge der Knoten jeder Zusammenhangskomponente

(1) Initialisierung:
MARK$(v) := -1$ für alle $v \in V$
NEXT $:= 1, L = \emptyset$

(2) FOR $v \in V$

(3) Füge v ans Ende der Liste L an

(4) WHILE $L \neq \emptyset$

(5) Wähle v vom Anfang der Liste und entferne v aus L

(6) IF MARK$(v) < 0$ THEN

(7) MARK$(v) :=$ NEXT

(8) END IF

(9) FOR $vw \in \delta(v)$ DO

(10) IF MARK$(w) < 0$ THEN

(11) MARK(w) := NEXT
(12) Füge w am Ende der Liste an
(13) End If
(14) End For
(15) End While
(16) NEXT = NEXT + 1
(17) End For

NEXT gibt die Anzahl der Zusammenhangskomponenten an. Die Knoten einer Zusammenhangskomponente sind mit demselben Wert markiert.

3.0.2 Depth-First-Search

Depth-First-Search arbeitet nach dem LIFO-Prinzip (Last-In-First-Out).

Algorithmus 3.9 (DFS)

Input: Graph $G = (V, E)$
Output: Anzahl der Zusammenhangskomponenten und die Menge der Knoten jeder Zusammenhangskomponente

(1) Initialisierung:
MARK(v) := -1 für alle $v \in V$
NEXT := 1
(2) For $v \in V$ Do
(3) If MARK(v) < 0 Then
(4) MARK(v) := NEXT
(5) For $vw \in \delta(v)$ Do
(6) check_nachbar(w, NEXT)
(7) NEXT := NEXT + 1
(8) End If
(9) End For
(10) Gib NEXT aus

check_nachbar(w, NEXT)
Input: $G = (V, E)$, $w \in V$, NEXT, MARK($\cdot$)

(1) If MARK(w) < 0 Do
(2) MARK(w) = NEXT

(3) For $wu \in \delta^+(w)$ Do

(4) check_nachbar(u, NEXT)

(5) End For

(6) End If

Literatur

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Company, 1974.

[2] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.