LMU LUDWIG-MAXIMILIANS-UNIVERSITY MUNICH
DEPARTMENT INSTITUTE FOR INFORMATICS
DATABASE SYSTEMS GROUP

Algorithmen und Datenstrukturen

Kapitel 5: Graphen und Graphalgorithmen

Skript zur Vorlesung

Algorithmen und Datenstrukturen

Sommersemester 2015

Ludwig-Maximilians-Universität München

basierend auf dem Skript von Prof. Dr. Martin Ester, Prof. Dr. Daniel A. Keim, Dr. Michael Schiwietz und Prof. Dr. Thomas Seidl

Allgemein I

- **Graph:** Menge von Objekten („Knoten“) und Beziehnungen zwischen den Objekten („Kanten“). Ein Graph stellt diese Beziehungen „grafisch“ dar.
- Beispiele:

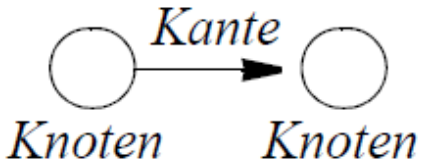
Objekte	Beziehungen zwischen den Objekten
Städte	Es gibt eine Flugverbindung zwischen Städten
Stellungen beim Schachspiel	Es gibt einen Zug von einer Stellung zur anderen
Zustände eines Automaten	Es gibt einen Übergang zwischen zwei Zuständen
www-Seiten	Es gibt einen Link zwischen den beiden Seiten
Soziale Netzwerke	Es gibt eine Verbindung zwischen Benutzern

Definition: (Endlicher) Graph:

Ein **Graph** G ist ein Paar $G = (V, E)$ mit:

V: endliche, nichtleere Menge von Knoten

E: Menge von Kanten: $E \subseteq V \times V$.



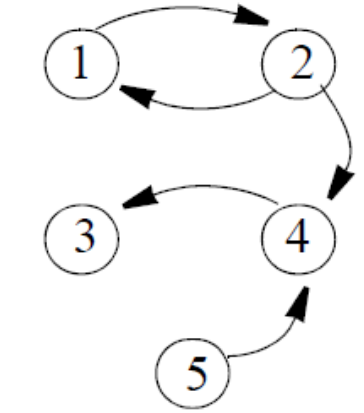
Allgemein III

Bezeichnungen:

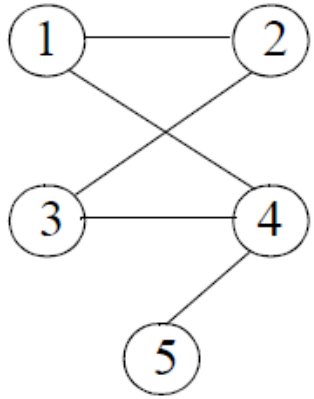
Unterscheidung zwischen:

- **Gerichtete Graphen**:
  $\big((v_1, v_2) \neq (v_2, v_1), \{v_1, v_2\} \subseteq V\big)$
  Kanten werden als → dargestellt.
- **Ungerichtete Graphen**:
  $\big((v_1, v_2) \equiv (v_2, v_1), \{v_1, v_2\} \subseteq V\big)$
  Kanten werden als – dargestellt.

Ungerichtete Graphen sind Spezialfälle gerichteter Graphen und lassen sich stets als solche darstellen!



gerichteter Graph



ungerichteter Graph

Allgemein IV

- Als **Teilgraph** eines Graphen $G = (V, E)$ bezeichnet man einen Graphen $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$.

- In einem ungerichteten Graphen heißen zwei Knoten v_1 und v_2 **benachbart** oder **adjazent**, falls $(v_1, v_2) \in E$.
- In einem gerichteten Graphen heißen zwei Knoten v_1 und v_2 **benachbart** oder **adjazent**, falls $(v_1, v_2) \in E$ oder $(v_2, v_1) \in E$.

- Der **Grad** eines Knotens ist die Anzahl der von ihm ausgehenden Kanten

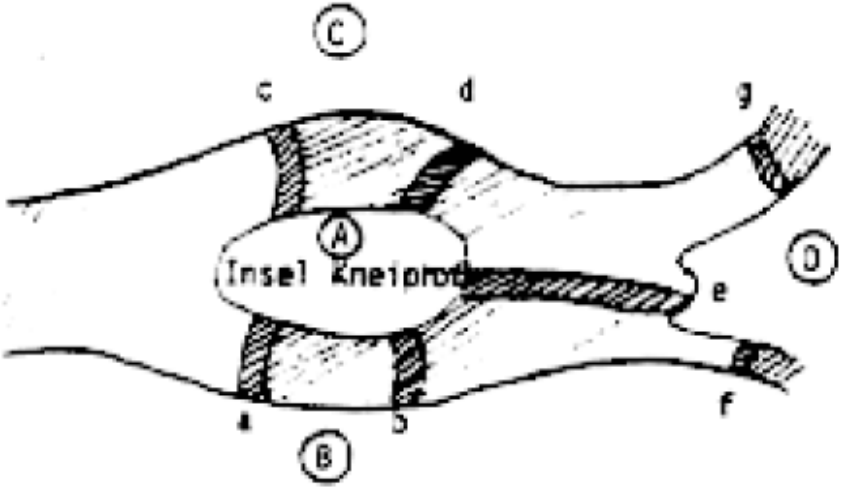
Allgemein V

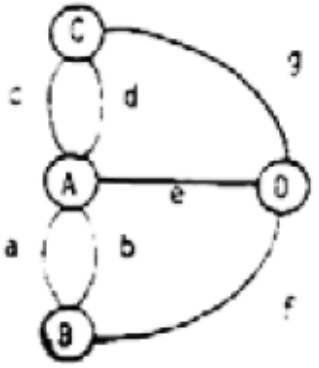
- Ein Graph ist ein **Multigraph**, wenn zwei Knoten durch mehr als eine Kante verbunden sind.

Beispiel:
Königsberger Brückenproblem

Ist es möglich, ausgehend von einem der Punkte A, B, C oder D, alle Brücken genau einmal zu überqueren und wieder an den Ausgangspunkt zurückzukehren?

→ Genau dann lösbar, wenn der Grad jedes Knotens gerade ist (Euler 1736).





Allgemein VI

- Ein **Weg (Pfad) der Länge n** vom Knoten v_1 zum Knoten v_2 ist eine Folge von Knoten $(w_0, w_1, \ldots, w_n)$ mit

  $$w_0 = v_1, w_n = v_2, w_i \in V, \qquad 0 \leq i \leq n, (w_i, w_{i+1}) \in E$$

- In einem **einfachen Weg** sind alle Knoten paarweise verschieden:

  $$w_i \neq w_j, \qquad 0 \leq i, j \leq n, i \neq j$$

  Ausnahme: $w_0 = w_n$ ist erlaubt.

- Ein **Zyklus** ist ein einfacher Weg mit $w_0 = w_n$, $n \geq 2$.
- Ein **Eulerscher Kreis** ist ein spezieller Zyklus, bei dem jede Kante genau einmal durchlaufen wird.
- Ein **Hamiltonscher Kreis** ist ein Zyklus, der jedem Knoten genau einmal durchläuft.

Allgemein VII

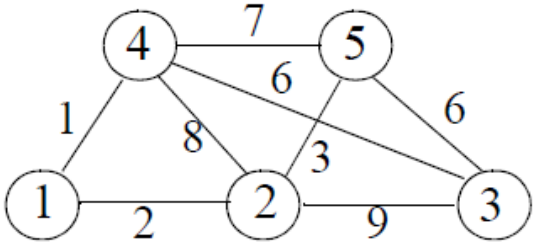
- In einem ungerichteten Graphen heißt ein Knoten v_1 mit dem Knoten v_2 **verbunden**, falls er einen Weg von v_1 nach v_2 gibt.
- In einem gerichteten Graphen heißen zwei Knoten v_1 und v_2 **stark verbunden**, falls es einen Weg von v_1 nach v_2 und von v_2 nach v_1 gibt.
- Ein ungerichteter Graph heißt **zusammenhängend**, wenn alle Knoten paarweise miteinander verbunden sind. Eine **(Zusammenhangs-)Komponente** von G ist ein maximaler zusammenhängender Teilgraph von G.
- Eine **stark verbundene Komponente** eines gerichteten Graphen ist ein maximaler Teilgraph (maximale Knoten- und Kantenmenge), in dem alle Knoten paarweise stark verbunden sind.
- Bemerkung: Bäume sind Spezialfälle von Graphen

Allgemein VIII

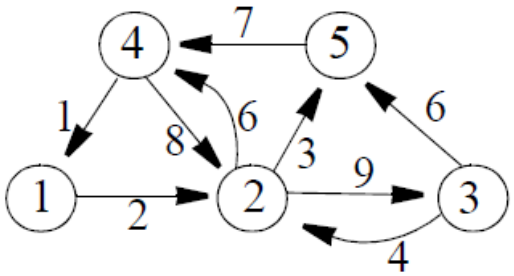
- Den Knoten und speziell den Kanten eines Graphen sind häufig weitere Informationen zugeordnet
  → **markierte Graphen**.

Beispiele für Kantenmarkierungen:

- Bedingungen für den Übergang von einem Knoten zum anderen (z.B.: endliche Automaten)
- Entfernungen oder Reisezeiten in einem Verkehrsnetz
- Allgemein: „Kosten“ für den Übergang von einem Knoten auf einen benachbarten Knoten



Graph G_1



Graph G_2

Speicherdarstellung von Graphen I

- Verschiedene Datenstrukturen zur Implementierung
- Geeignete Speicherdarstellung abhängig vom Anwendungsfall

Es stehen sich dabei gegenüber:

- Der benötigte **Speicheraufwand**
- Die **Effizienz**, mit der Zugriffe auf Knoten/Kanten durchgeführt werden

Im Folgenden:

- Vorstellung der **beiden gebräuchlichsten Datenstrukturen** zur Implementierung der abstrakten Klasse `Graph`.
- Sind als **statische Strukturen** zu verstehen (Unterstützung dynamischer Einfügungen und Entfernungen während der Laufzeit des darauf basierenden Algorithmus ist kein Ziel)

Speicherdarstellung von Graphen II


```
public abstract class NodeSet {
   public NodeSet() {}
   // Knoten sind durch ihre int-Schlüssel repräsentiert
   public abstract void insert(int);
   public abstract void delete(int);
   public abstract boolean isEmpty();
   public abstract boolean contains(int);
   public abstract boolean hasNext();
   public abstract int nextElement();
   public abstract void union(NodeSet);
}
```

```
public abstract class Graph {
   public int n; // Anzahl der Knoten
   public abstract NodeSet getNeighbors(int);
   public abstract float cost(int, int);
   public void breadthFirstTraversal(int) {...}
}
```

Adjazenzmatrix I

- Gegeben sei ein Graph $G = (V, E)$ mit $|V| = n$.
- Seien im folgenden die Knoten mit 1 bis n durchnummeriert.
- Eine **Adjazenzmatrix** ist eine boolsche $n \times n$-Matrix A mit:

$$A[i,j] = \begin{cases} 1 & falls\ (v_i, v_j) \in E \\ 0 & sonst \end{cases}$$

- Adjazenzmatrix zu Graph G_1:

	1	*2*	*3*	*4*	*5*
1	0	1	0	1	0
2	1	0	1	1	1
3	0	1	0	1	1
4	1	1	1	0	1
5	0	1	1	1	0

Adjazenzmatrix II

- Adjazenzmatrix eines ungerichteten Graphen ist symmetrisch
  → Speicherung der oberen Dreiecksmatrix genügt.
- Werden in A die Kosten der Kanten eines markierten Graphen eingetragen, so nennt man A eine **markierte Adjazenzmatrix** oder **Kostenmatrix**.
- Nichtexistierende Kanten werden durch ein ausgewähltes Symbol (hier: ∞) bezeichnet.
- Für alle i, $1 \leq i \leq n$ gilt: $A[i,i] = 0$.
- Markierte Adjazenzmatrix zu Graph G_2:

	1	2	3	4	5
1	0	2	∞	∞	∞
2	∞	0	9	6	3
3	∞	4	0	∞	6
4	1	8	∞	0	∞
5	∞	∞	∞	7	0

Adjazenzmatrix III

Für die Graphendarstellung mit Adjazenzmatrizen gilt:

Vorteil:

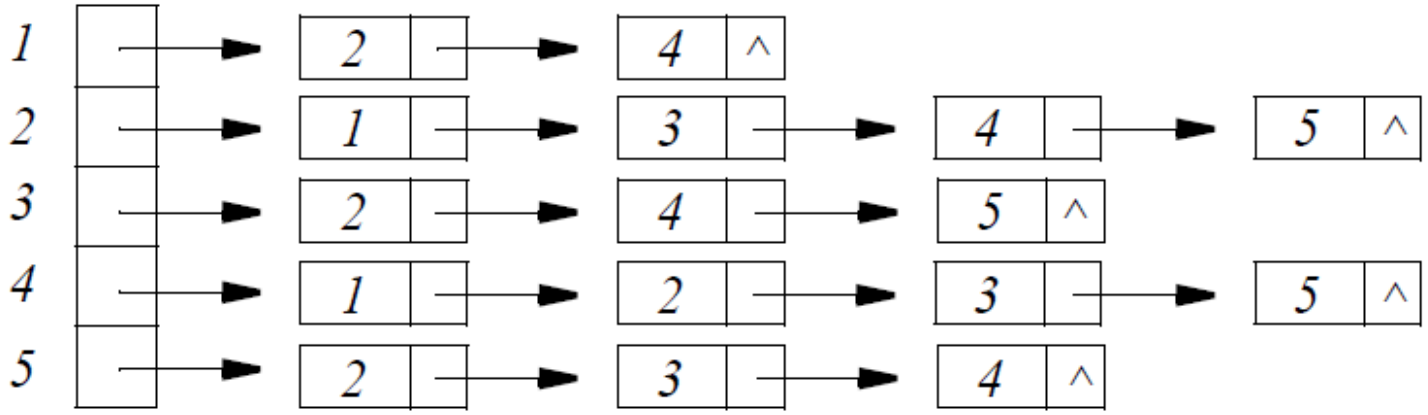
- Die Existenz einer bestimmten Kante kann in $O(1)$ Zeit ermittelt werden

Nachteile:

- Hoher Platzbedarf: $O(n^2)$
  (bei kleinerer Kantenanzahl unverhältnismäßig)
- Hohe Initialisierungszeit von $O(n^2)$
- Algorithmen, die den gesamten Graphen betrachten: $\Theta(n^2)$ Aufwand

Adjazenzliste I

- Jedem der n Knoten wird eine Liste aller seiner adjazenten Knoten (Nachfolger) zugeordnet
- Für einen effizienten Zugriff werden die Listenköpfe in einem Array vom Typ ‚**Adjazenzliste**‘ organisiert:
- Adjazenzliste zu Graph G_1:



- Bei markierten Graphen: Listenelemente können weitere Informationen (z.B. Kosten der Kante) enthalten.

Adjazenzliste II

Für die Darstellung mit Adjazenzlisten gilt:

Vorteile:

- Der Platzbedarf beträgt nur $O(n + |E|)$.
- Alle Nachbarn eines Knotens werden in linearer Zeit (Anzahl der Kanten, die von diesem Knoten ausgehen) gefunden

Nachteil:

- Zugriff auf eine bestimmte Kante (v, w) ist im Allgemeinen nicht in konstanter Zeit möglich, da die Adjazenzliste von v zu durchlaufen ist.

Graphendurchläufe I

Problem:

Man besuche systematisch alle Knoten eines Graphen $G = (V, E)$, die von einem Knoten v aus erreichbar sind.
→ Die meisten Graphalgorithmen bauen auf die Lösung dieses Grundproblemes auf.

Definition:

Ein Graphdurchlauf beginnt mit einem Knoten $v \in V$, der sogenannten **Wurzel**, und sucht jeden von v aus erreichbaren Knoten in G auf. Die Menge dieser Knoten bildet zu v einen **Wurzelgraphen**.

Graphendurchläufe II

Erläuterung:

- Der Graphdurchlauf erzeugt einen Baum der besuchten Knoten
- Dieser wird unendlich, falls der Wurzelgraph mindestens einen Zyklus enthält
- Der Durchlauf wird daher dort abgebrochen, wo er auf einen bereits zuvor besuchten Knoten trifft.

Ansätze, einen Graphen systematisch zu durchlaufen:

- **Tiefendurchlauf**
- **Breitendurchlauf**

Tiefendurchlauf I

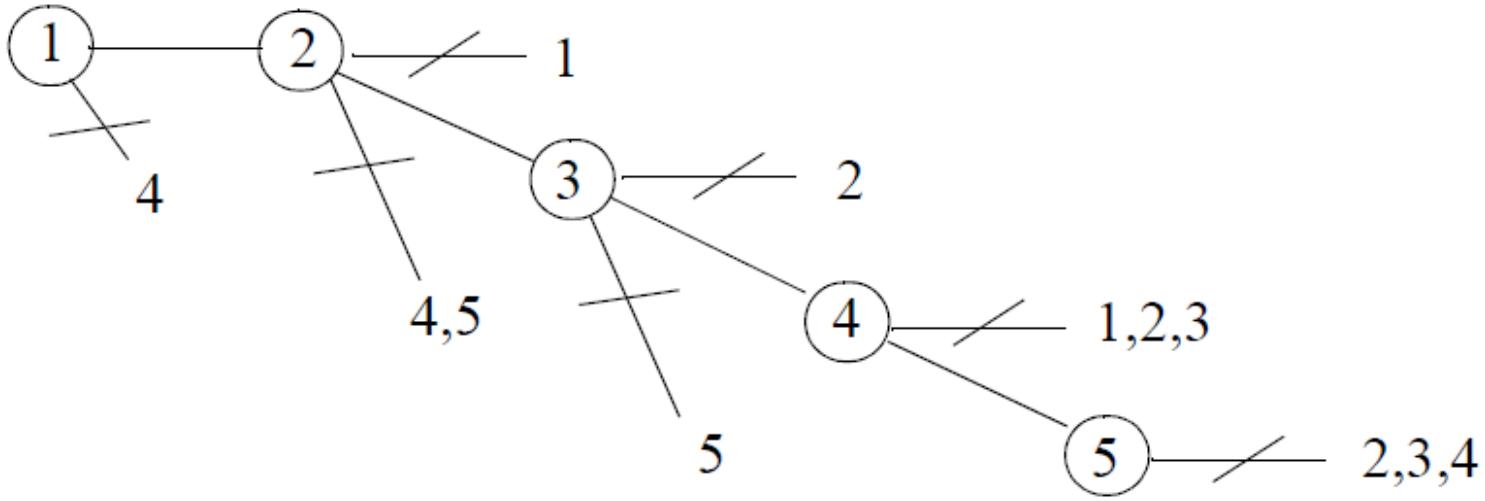
Tiefendurchlauf (depth-first-search)**:**

Prinzip:

- Ausgehend von v wird ein zu v adjazenter Knoten $v' \in V$ besucht.
- Besuchte Knoten werden stets markiert.
- Bevor nun weitere zu v adjazente Knoten betrachtet werden, wird das Verfahren rekursiv auf v' angewandt.
- Der auf diese Weise erzeugte Baum heißt **depth-first Spannbaum** (depth-first spanning tree)

Tiefendurchlauf II

Depth-first Spannbaum zu Graph G_1 (Startknoten $v = 1$)



- Die folgende Methode `depthFirstTraversal` erzeugt bzw. durchläuft für jede Komponente des Graphen einen depth-first Spannbaum.
- Die Knoten seien dabei mit $0, \ldots, n-1$ bezeichnet.
- Verwendung einer rekursiven Hilfsmethode ‚`depthSearch`'.

Tiefendurchlauf III


```
public void depthSearch(int v, boolean[] visited) {
   visited[v] = true;
   // Verarbeitung des Knotens v ...;
   NodeSet neighbors = getNeighbors(v);
   while (neighbors.hasNext()) {
      int w = neighbors.nextElement();
      if (!visited[w])
         depthSearch(w, visited);
   }
}
public void depthFirstTraversal() {
   boolean visited[] = new boolean[n];
   int v;
   for (v = 0; v < n; v++)
      visited[v] = false;
   for (v = 0; v < n; v++)
      if (!visited[v])
         depthSearch(v, visited);
}
```

Tiefendurchlauf IV

Laufzeitverhalten des Tiefendurchlaufs im schlechtesten Fall:

Mit Adjazenzliste:

$$O(n + |E|) = O(\max(n, |E|))$$

Mit Adjazenzmatrix:

$$O(n^2)$$

Breitendurchlauf I

Breitendurchlauf (breadth-first-search)**:**

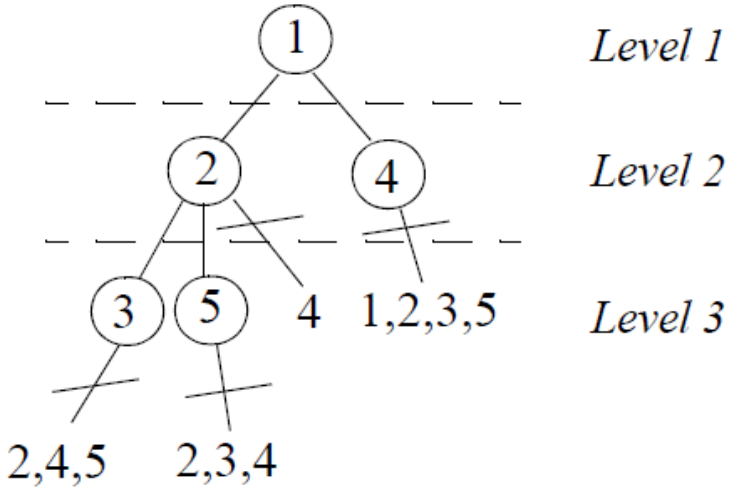
Prinzip:

- Ausgehend von v werden zunächst *alle* zu v adjazenten Knoten besucht und in einer Schlange notiert.
- Anschließend werden nacheinander die Knoten in der Schlange auf dieselbe Weise behandelt, wobei bereits besuchte Knoten übergangen werden.
  → ‚ebenenweiser' Durchlauf

Der auf diese Weise erzeugte Baum heißt
breadth-first Spannbaum (breadth-first spanning tree).

Breitendurchlauf II

Breadth-first Spannbaum zu Graph G_1 (Startknoten $v = 1$)



- Die folgende Methode `breadthFirstTraversal` erzeugt bzw. durchläuft für jede Komponente des Graphen einen breadth-first Spannbaum.
- Als Datenstruktur zur Verwaltung der noch zu besuchenden Knoten wird eine Schlange (Queue) verwendet.
- Die Knoten seien dabei mit $0, \ldots, n-1$ bezeichnet.

Breitendurchlauf III


```
public void breadthFirstTraversal() {
   boolean visited[] = new boolean[n];
   Queue q; // Schlange von Knoten mit Queue-Methoden
   for (int v = 0; v < n; v++) visited[v] = false;
   q = new Queue();
   for (int v = 0; v < n; v++) {
      if (!visited[v]) {
         q.add(v);
         visited[v] = true;
         while (!q.isEmpty()) {
            // Verarbeitung des Knotens w ...;
            int w = q.poll(); // entfernt erstes Element der Queue
            NodeSet neighbors = getNeighbors(w);
            while (neighbors.hasNext()) {
               int x = neighbors.nextElement();
               if (!visited[x]) {
                  q.add(x);
                  visited[x] = true;
} } } } } }
```