Inhalt

- 3 Datenstrukturen
  - 3.1 Folgen
  - 3.2 Bäume
    - 3.2.1 Grundlagen
    - 3.2.2. Funktonale Spezifikation und einfache Algorithmen
    - 3.2.3 Anwendung von binären Bäumen (Huffman-Code)
    - 3.2.4 Imperative Implementierungen von Bäumen
  - 3.3 Effiziente Repräsentation von Mengen
    - 3.3.1 Spezifikation
    - 3.3.2 Suchbäume
      Binäre Suchbäume, AVL-Bäume, B-Bäume
    - 3.3.5 Hashverfahren
    - 3.3.6 Tries, Patricia-Bäume und Heaps
  - 3.4 Graphen und Graphalgorithmen

3.4 Graphen

- Terminologie (teilweise Wiederholung)

  $G = (E, K)$

  E: Knotenmenge

  K: Kantenmenge $\subseteq E \times E$ — Gerichteter Graph

  Kanten besitzen Richtung

  oder $K \subseteq \{\{x,y\} \mid x,y \in E\}$ — Ungerichteter Graph

  Schreibweise für Kanten meist gleich: (x,y)
  Gerichtet / ungerichtet geht aus dem Kontext hervor

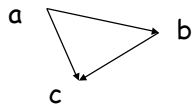
  Weg (Pfad) der Länge n von a nach b: $(x_0,x_2,....x_{n-1},\overrightarrow{x_n}) = (a,b)$
  $x_i \in E$, $x_0 = a$, $x_n = b$

  Zyklus: Weg von x nach x
  in gerichteten Graphen Zyklen der Länge 1 möglich: (a,a)

Graphen

- Begriffe...

  Zyklenfrei (kreisfrei): es gibt keine Zyklen



zykelfrei!

G zusammenhängend: für alle a,b ∈ E gibt es Weg von a nach b

G gerichtet, stark zusammenhängend: für alle a, b ∈ E gibt es Weg von a nach b und von b nach a

Spezialfall: gerichteter, zyklenfreier Graph
DAG (directed acyclic graph)

Graphen

- Gewichtete Graphen

  ... besitzen eine Kantenmarkierung

  w: E -> Num (Num oft ganze Zahlen, auch reelle)

  Kantengewicht kann z.B. Entfernung, Leitungskapazität, Zeitdauer,... ausdrücken

- Anwendungen: Graphen als Modellierungsmittel
  - Straßentopologie: Finde den kürzesten Weg von a nach b
  - Vernetzung: finde eine möglichst billiges Netz (einen zusammenhängenden Graphen mit möglichst wenigen(?) Kanten)
  - Netzplan von Aktivitäten: Kreisfrei(!),
  - Spiele: Labyrinth etc.
  - In der Informatik .....

Graphen

- Typische Algorithmen
  - alle Knoten durchlaufen
    - Breitensuche ("erst alle Nachbarn")
    - Tiefensuche ("Pfad so lange wie möglich verlängern")
  - Weg von a nach b finden (evtl. kürzesten)
  - Alle kürzesten Wege von a zu beliebigem anderen Knoten
  - Minimalen spannenden Baum finden
    Graph, der keine "überflüssigen" Kanten enthält und gewichtsminimal ist
- Laufzeit?
  abhängig von ....
- Implementierung

Graphen

- Implementierung
  - Uungewöhnliche Implementierung:
    Externspeicherrelationen
    intern: "Kantenliste"

R = { (x,y,w) | (x,y) ∈ K mit Kantengewicht w}

S	T	w
....	.....	...
x	y	4
....	...	...

"Finde alle Pfade der Länge 2 mit Anfangspunkt x und Kantengewichten w, w' < 5 "

Mit "Zugriffspfad" für Attribut S - z.B. Hash oder B-Baum, S als Schlüssel - lässt sich das selbst für sehr große Graphen effizient realisieren.
Bsp: Positionierungssysteme mit Straßentopologie

Graph-Implementierung

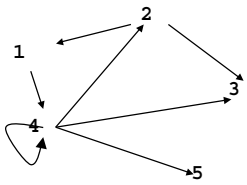
- Adjazenzmatrix

```
int E = 100; // 100 Knoten
boolean a[][] = new boolean [E][E];
```

`a[i][j] = 1` genau dann wenn es Kante von i nach j gibt

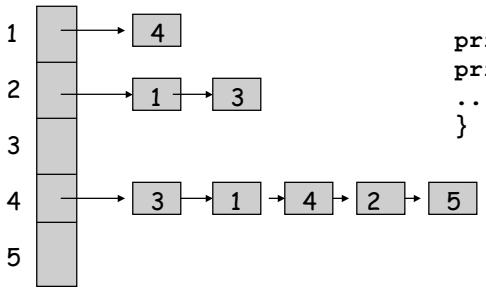
Ungerichteter Graph: Diagonalmatrix reicht wegen Symmetrie

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

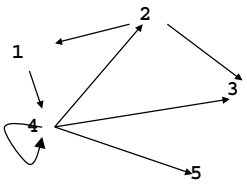


Graph-Implementierung

- Adjazenzliste

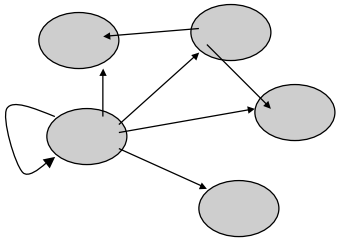



```
public class Graph {
 class Edge {// only
     destination req. ...}
private Hashtable nodeLabels;
private Vector nodes;
....
}
```



Graphimplementierung

- Objektorientiert als Geflecht, direkte Repräsentation des Graphen



```
interface Node {
Node create();           // creates an unlabeled node
void setLabel(int k);       // labels node
boolean getLabel();      // label of node
void connect(int n);     // creates edge to node with
                         // label  n
Node findNode(int n);    // return node with label n
Set succs();             // delivers successors
...
}
```