Algorithmen und Datenstrukturen

Graphen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Danksagung

Die nachfolgenden Präsentationen wurden mit ausdrücklicher Erlaubnis des Autors übernommen aus:

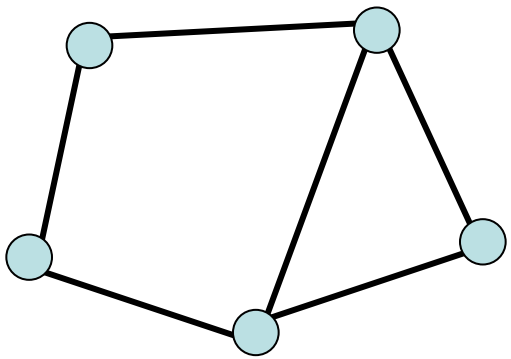
- „Effiziente Algorithmen und Datenstrukturen“ (Kapitel 7,8,9) gehalten von Christian Scheideler an der TUM http://www14.in.tum.de/lehre/2008WS/ea/index.html.de

Es wurden umfangreiche Veränderungen vorgenommen.

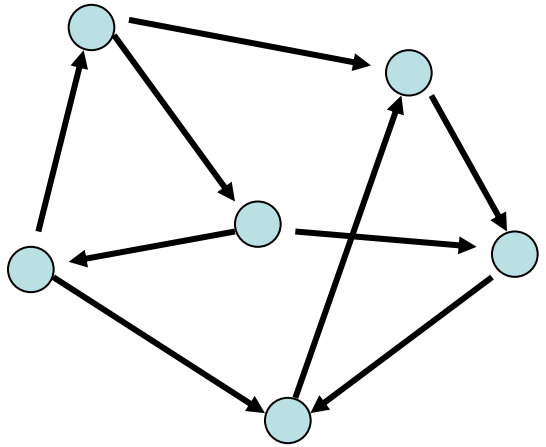
Graphen

Graph G=(V, E) besteht aus

- Knotenmenge V
- Kantenmenge E



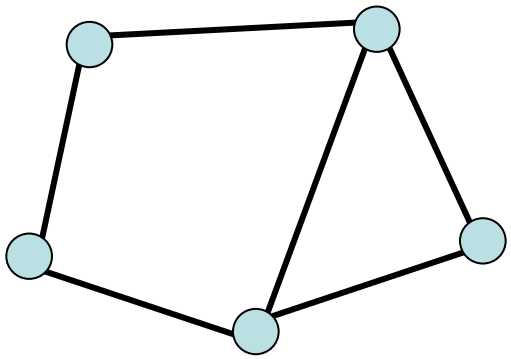
ungerichteter Graph



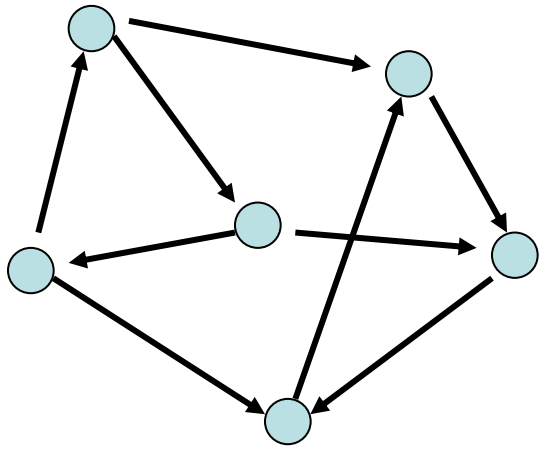
gerichteter Graph

Graphen

- Ungerichteter Graph: Kante repräsentiert durch Menge $\{v,w\}$ mit $v, w \in V$
- Gerichteter Graph: Kante repräsentiert duch Paar $(v,w) \in V \times V$ (bedeutet $v \longrightarrow w$)



ungerichteter Graph



gerichteter Graph

Graphen

- Ungerichtete Graphen: Symmetrische Beziehungen jeglicher Art
  - z.B. $\{v,w\} \in E$ genau dann, wenn Distanz zwischen v und w maximal 1 km
- Gerichtete Graphen: Asymmetrische Beziehungen
  - z.B. $(v,w) \in E$ genau dann, wenn Person v einer Person w eine Nachricht sendet

- Grad eines Knotens: Anzahl der ausgehenden Kanten

Graphen

Im Folgenden: nur gerichtete Graphen.

Modellierung eines ungerichteten Graphen als gerichteter Graph:

Ungerichtete Kante ersetzt durch zwei gerichtete Kanten.

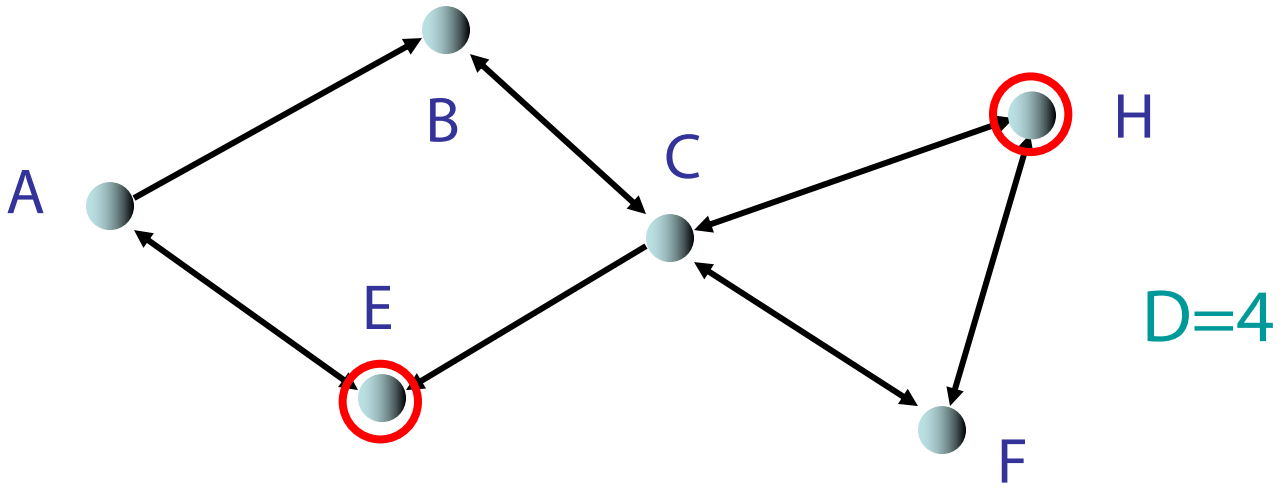
- n: aktuelle Anzahl Knoten
- m: aktuelle Anzahl Kanten

Graphen

- $\delta(v,w)$: Distanz

  Länge eines kürzesten gerichteten Weges
  von w zu v in G, ∞ wenn v von w nicht erreichbar

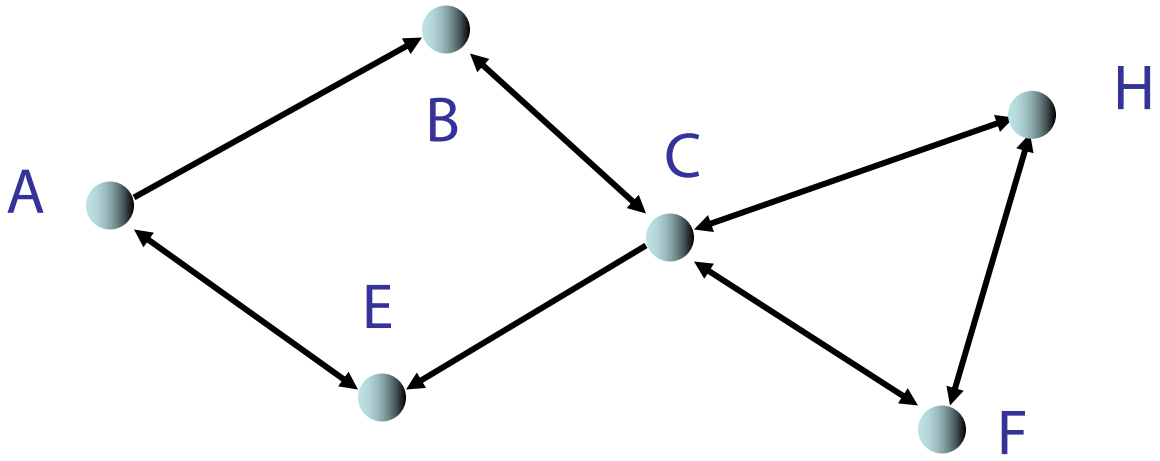
- $D=\max_{v,w} \delta(v,w)$: Durchmesser von G



Graphen

G heißt

- (schwach) zusammenhängend:
  Durchmesser D endlich, wenn alle Kanten als ungerichtet betrachtet werden
- stark zusammenhängend: wenn D endlich



Operationen auf Graphen

Sei $G=(V, E)$ ein Graph,
$e \in E$ eine Kante und $v \in V$ ein Knoten

Operationen:

- insert(e, G): $E:=E \cup \{e\}$
- remove(i, j, G): $E:=E \setminus \{e\}$ für die Kante $e=(v,w)$ mit key(v)=i und key(w)=j
- insert(v, G): $V:=V \cup \{v\}$
- remove(i, G): Sei $v \in V$ der Knoten mit key(v)=i
  $V := V \setminus \{v\}$, $E := E \setminus \{(x,y) \mid x=v \vee y=v\}$
- find(i, G): gib Knoten v aus mit key(v)=i
- find(i, j, G): gib Kante (v,w) aus mit key(v)=i und key(w)=j

Operationen auf Graphen

Anzahl der Knoten oft fest. In diesem Fall:

- V={1,…,n} (Knoten hintereinander nummeriert, identifiziert durch ihren Schlüssel aus {1,…,n})

Relevante Operationen:

- insert(e, G): E:=E U {e}
- remove(i, j, G): E:=E \ {e} für die Kante e=(i, j)
- find(i, j, G): gib Kante e=(i, j) aus

Operationen auf Graphen

Anzahl der Knoten variabel:

- Hashing kann verwendet werden, um Schlüssel von n Knoten in Bereich {1,…,O(n)} zu hashen
- Damit kann variabler Fall auf den Fall einer statischen Knotenmenge reduziert werden. (Nur O(1)-Vergrößerung gegenüber statischer Datenstruktur)

Operationen auf Graphen

Im Folgenden:

Konzentration auf statische Anzahl an Knoten.

Parameter für Laufzeitanalyse:

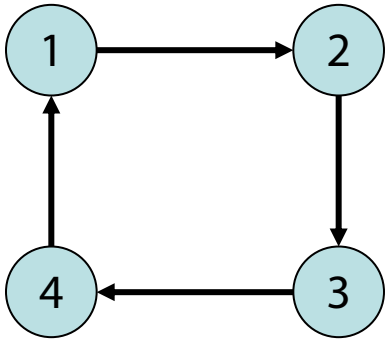
- n: Anzahl Knoten
- m: Anzahl Kanten
- d: maximaler Knotengrad (maximale Anzahl ausgehender Kanten von Knoten)

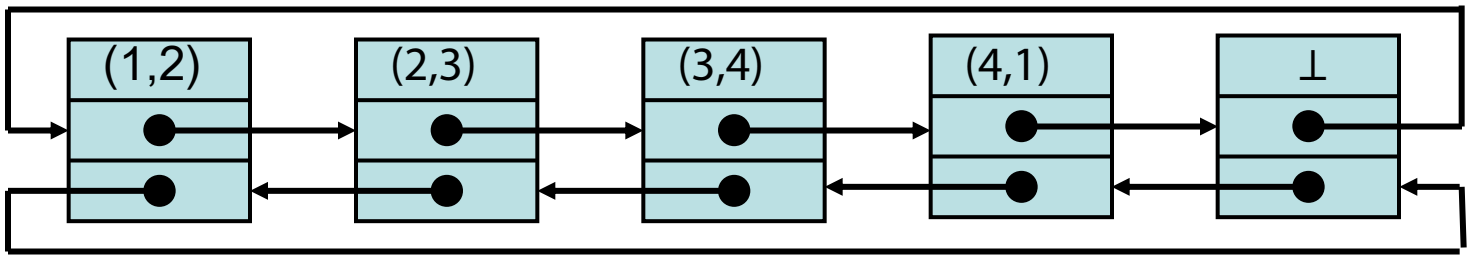
Graphrepräsentationen

1. Sequenz von Kanten
2. Adjazenzfeld
3. Adjazenzliste
4. Adjazenzmatrix
5. Adjazenzliste + Hashtabelle
6. Implizite Repräsentationen

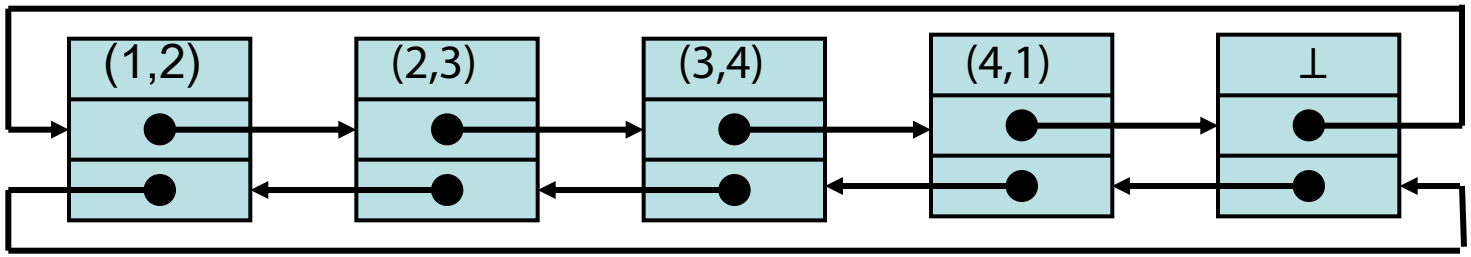
Graphrepräsentationen

1: Sequenz von Kanten





Sequenz von Kanten

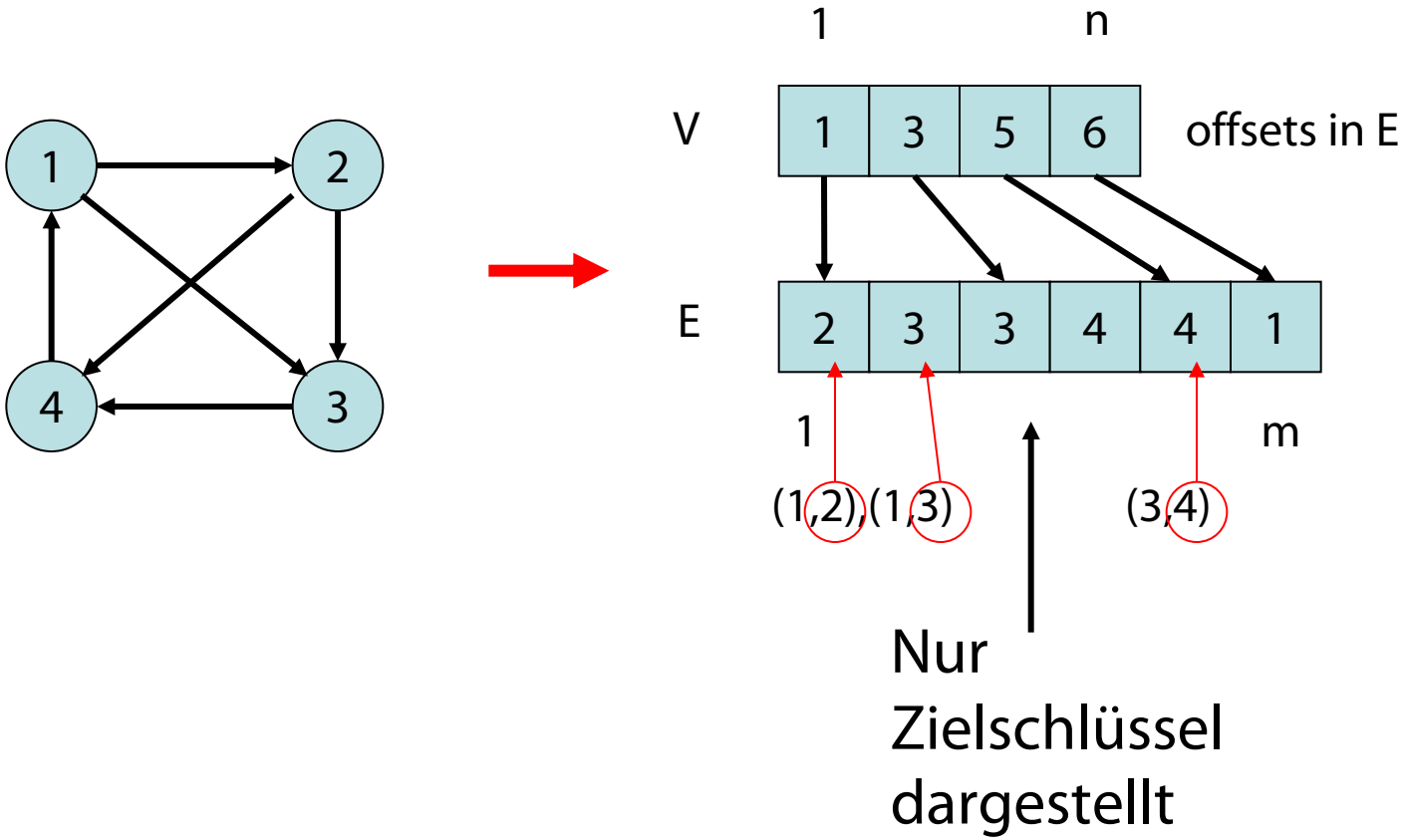


Zeitaufwand:

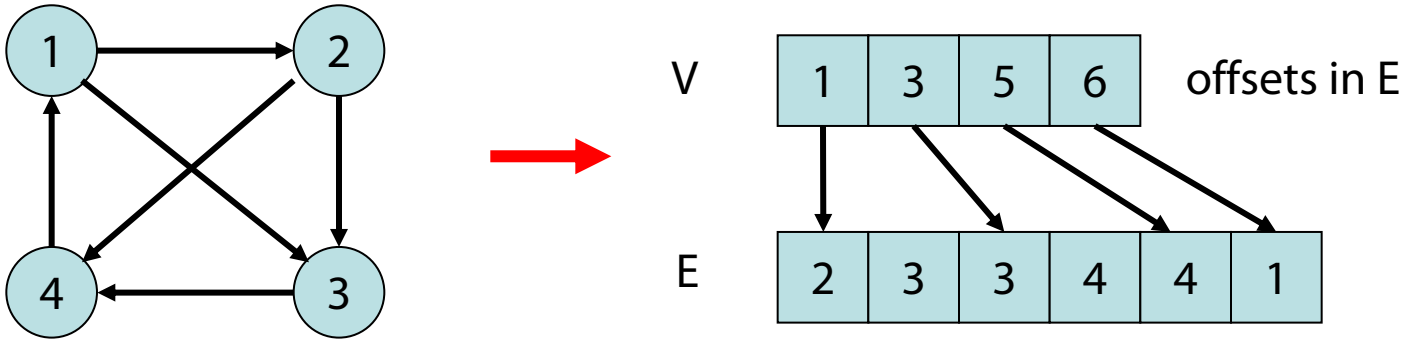
- find(i, j, G): Θ(m) im schlimmsten Fall
- insert(e, G): O(1)
- remove(i, j, G): Θ(m) im schlimmsten Fall

Graphrepräsentationen

2: Adjazenzfeld



Adjazenzfeld

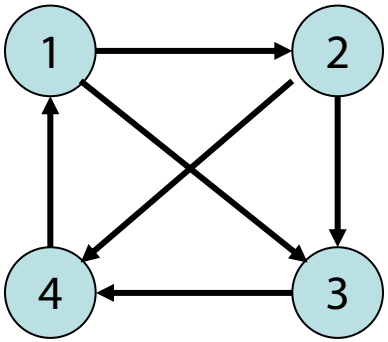


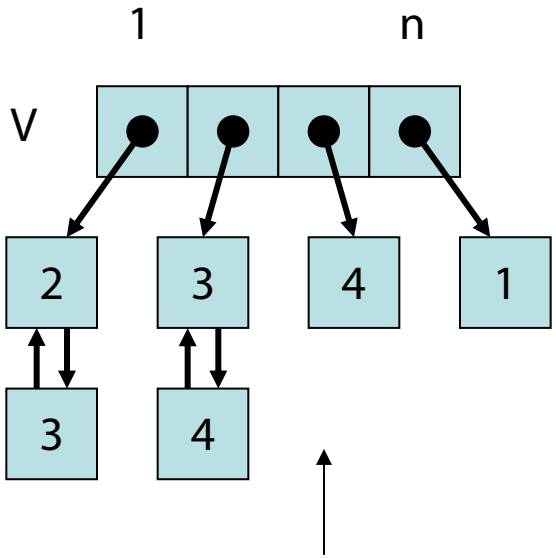
Zeitaufwand:

- find(i, j, G): Zeit O(d)
- insert(e, G): Zeit O(m) (schlimmster Fall)
- remove(i, j, G): Zeit O(m) (schlimmster Fall)

Graphrepräsentationen

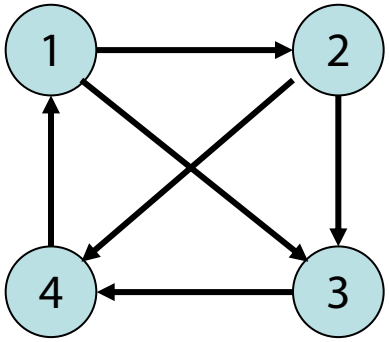
3: Adjazenzliste

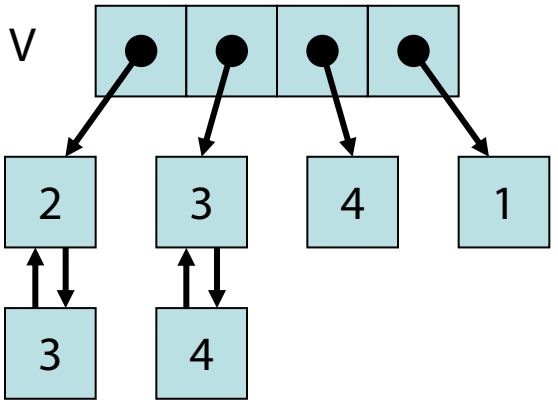




Nur Zielschlüssel dargestellt

Adjazenzliste





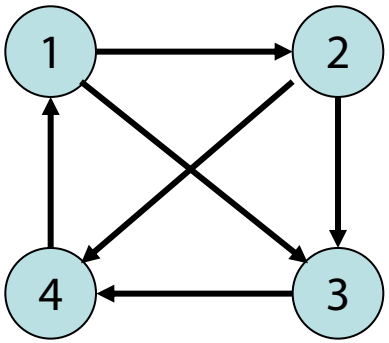
Zeitaufwand:

- find(i, j, G): Zeit O(d)
- insert(e, G): Zeit O(d)
- remove(i, j, G): Zeit O(d)

Problem: d kann auch groß sein!

Graphrepräsentationen

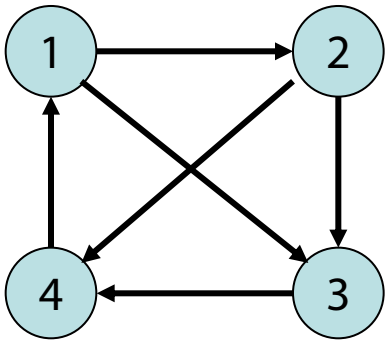
4: Adjazenzmatrix



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

- $A[i,j] \in \{0,1\}$ (bzw. Zeiger auf ein $e \in E$)
- $A[i,j]=1$ genau dann, wenn $(i,j) \in E$

Adjazenzmatrix



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

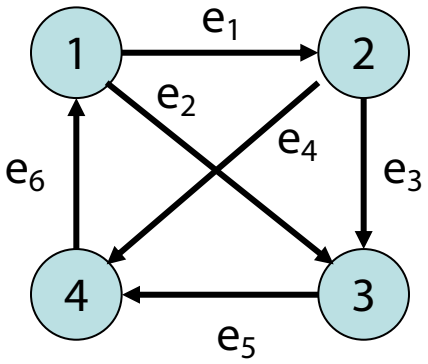
Zeitaufwand:

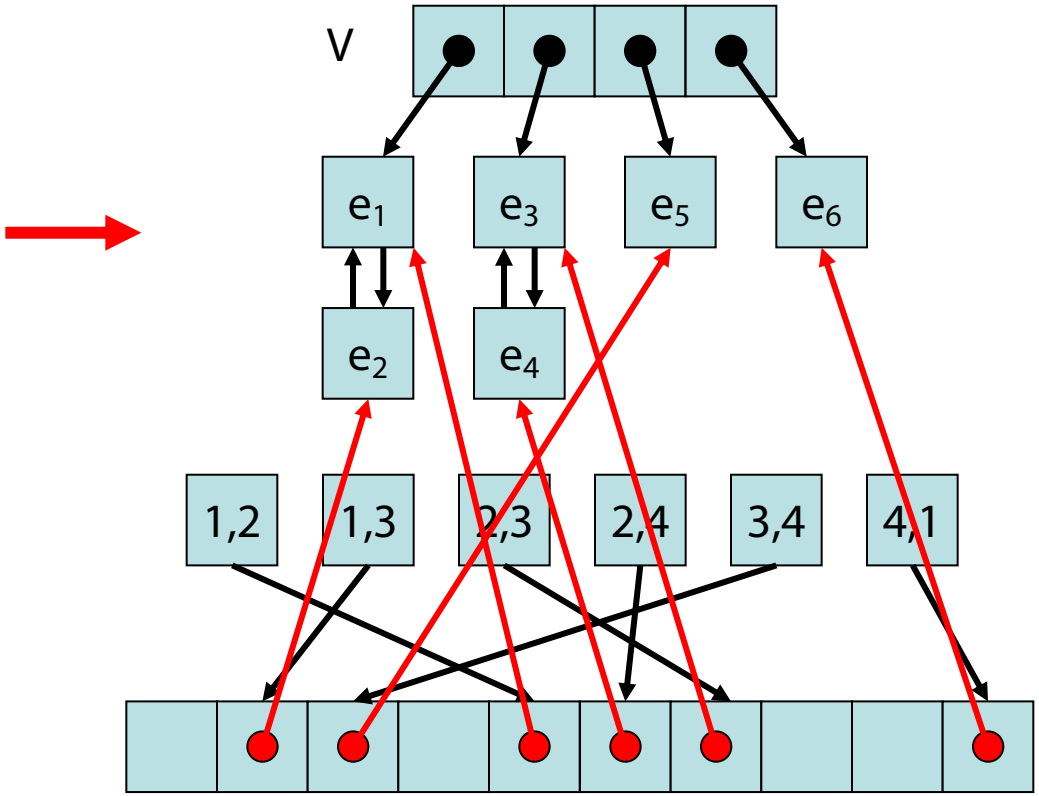
- find(i, j, G): Zeit O(1)
- insert(e, G): Zeit O(1)
- remove(i, j, G): Zeit O(1)

Aber: Speicher-aufwand $O(n^2)$

Graphrepräsentationen

5: Adjazenzliste + Hashtabelle

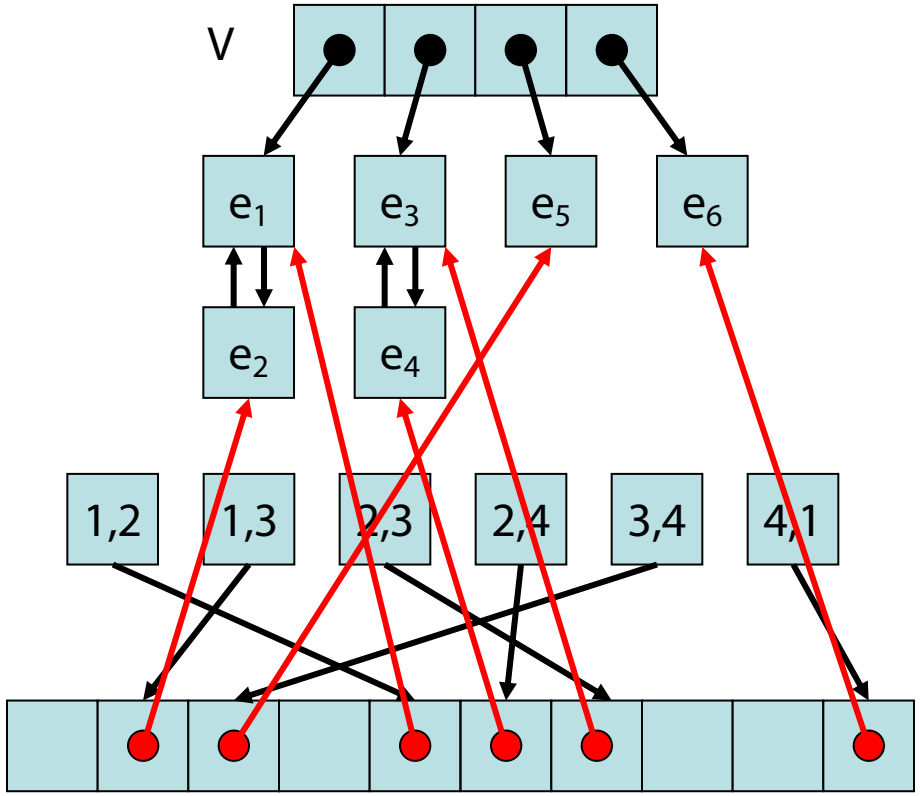




Adjazenzliste+Hashtabelle

Zeitaufwand (grob):

- find(i, j, G):
  O(1) (worst case)
- insert(e, G):
  O(1) (amortisiert)
- remove(i, j, G):
  O(1) (amortisiert)
- Speicher: O(n+m)



Gustavson, F. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw. 4*, 3, 250–269, 1978

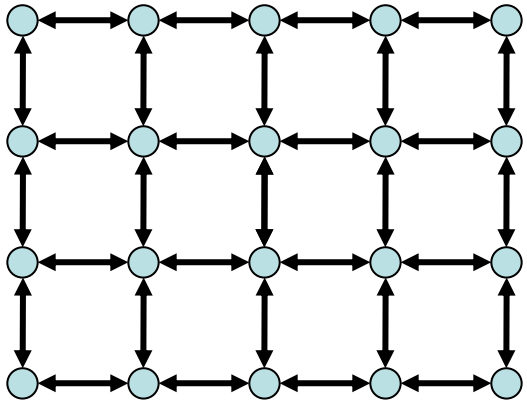
Graphrepräsentationen

6: Implizite Repräsentationen

(k,l)-Gitter G=(V,E):

- $V=[k] \times [l]$ ($[a]=\{0,\ldots,a-1\}$ für $a \in \mathbb{N}$)
- $E=\{((v,w),(x,y)) \mid (v=x \wedge |w-y|=1) \vee (w=y \wedge |v-x|=1)\}$

Beispiel: (5,4)-Gitter



Graphrepräsentationen

6: Implizite Repräsentationen

(k,l)-Gitter G=(V,E):

- $V=[k] \times [l]$ ($[a]=\{0,\ldots,a-1\}$ für $a \in \mathbb{N}$)
- $E=\{((v,w),(x,y)) \mid (v=x \wedge |w-y|=1) \vee (w=y \wedge |v-x|=1)\}$

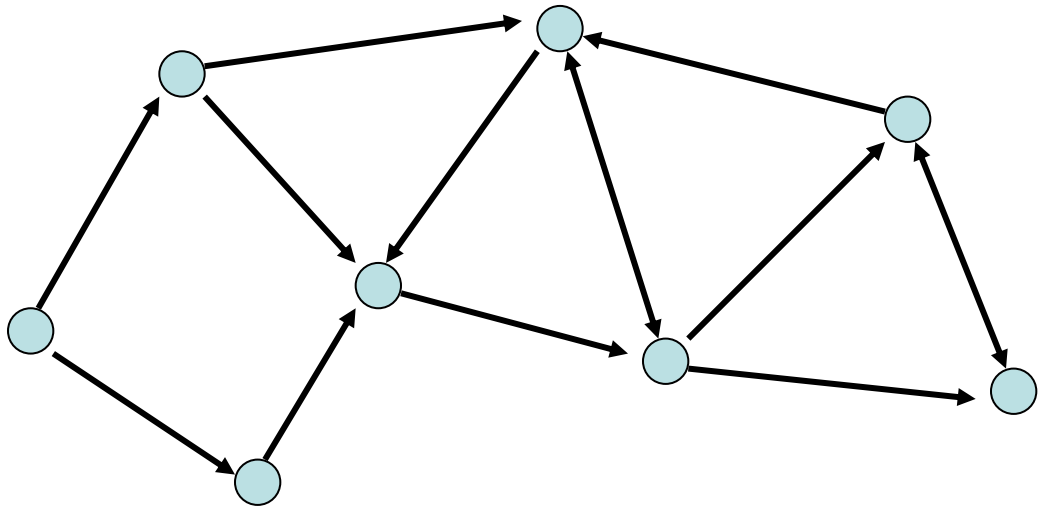
- Speicheraufwand: $O(\log k + \log l)$
  (speichere Kantenregel sowie k und l)
- Find-Operation: $O(1)$ Zeit (reine Rechnung)

Graphen

Traversierung

Graphdurchlauf

Zentrale Frage: Wie können wir die Knoten eines Graphen durchlaufen, so dass jeder Knoten mindestens einmal besucht wird?



Graphdurchlauf

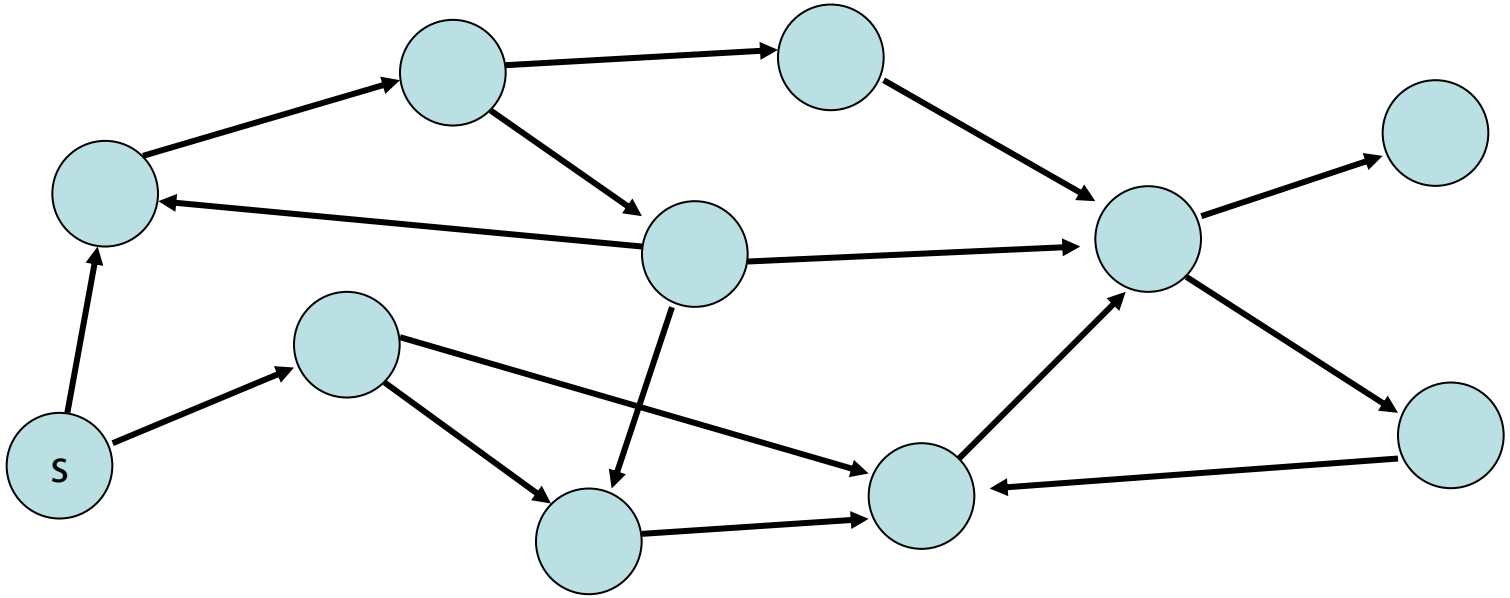
Zentrale Frage: Wie können wir die Knoten eines Graphen durchlaufen, so dass jeder Knoten mindestens einmal besucht wird?

Grundlegende Strategien:

- Breitensuche
- Tiefensuche

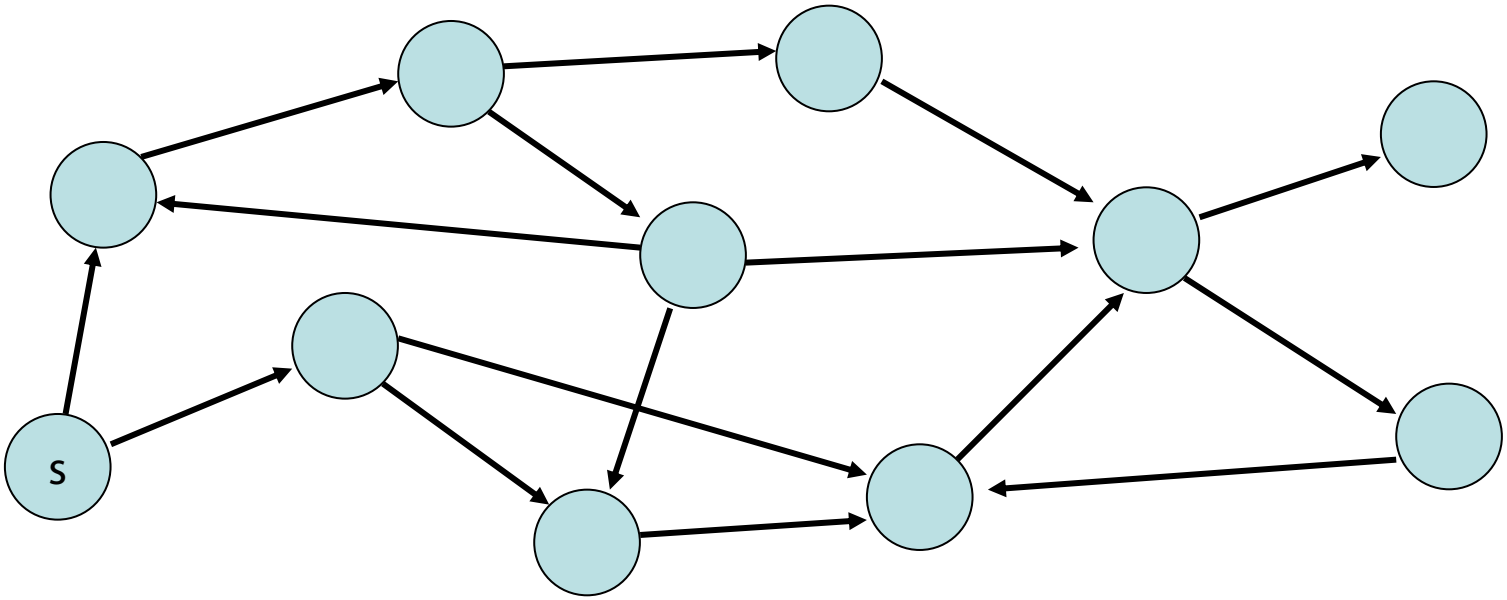
Breitensuche

- Starte von einem Knoten s
- Exploriere Graph Distanz für Distanz



Tiefensuche

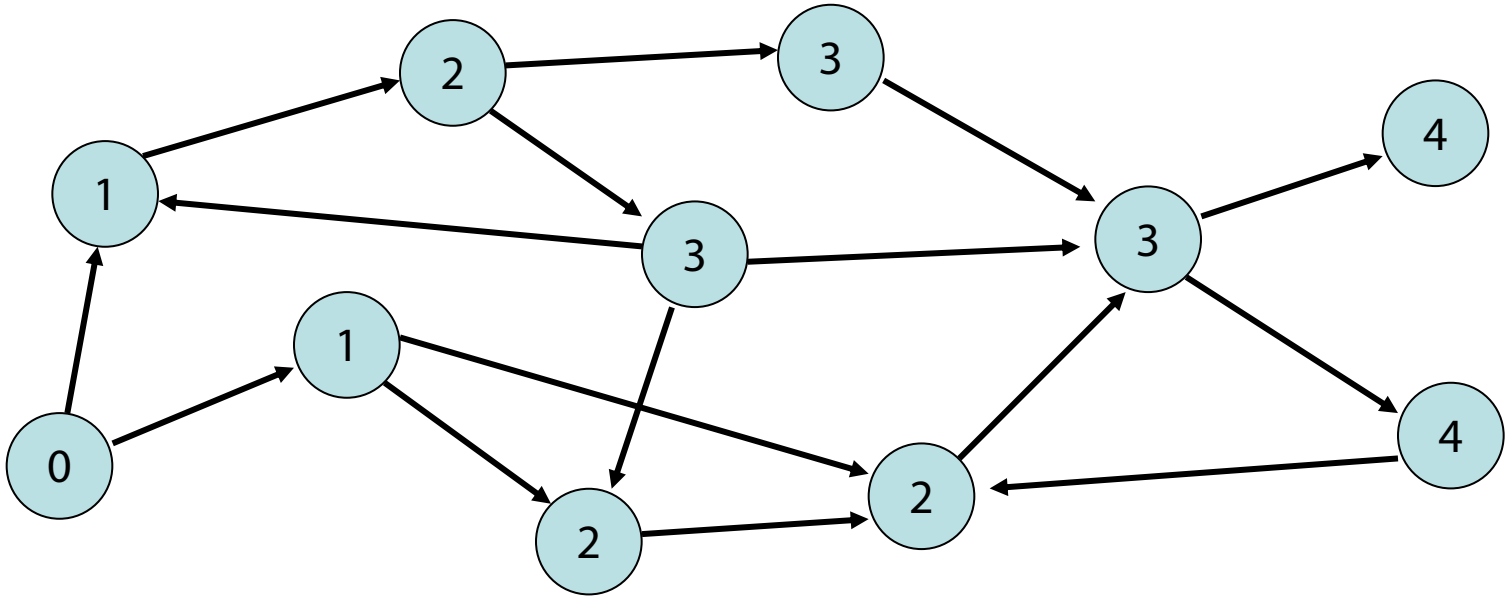
- Starte von einem Knoten s
- Exploriere Graph in die Tiefe
  (●: aktuell, ●: noch aktiv, ●: fertig)



Breitensuche

- d(v): Distanz von Knoten v zu s (d(s)=0)
- parent(v): Knoten, von dem v besucht

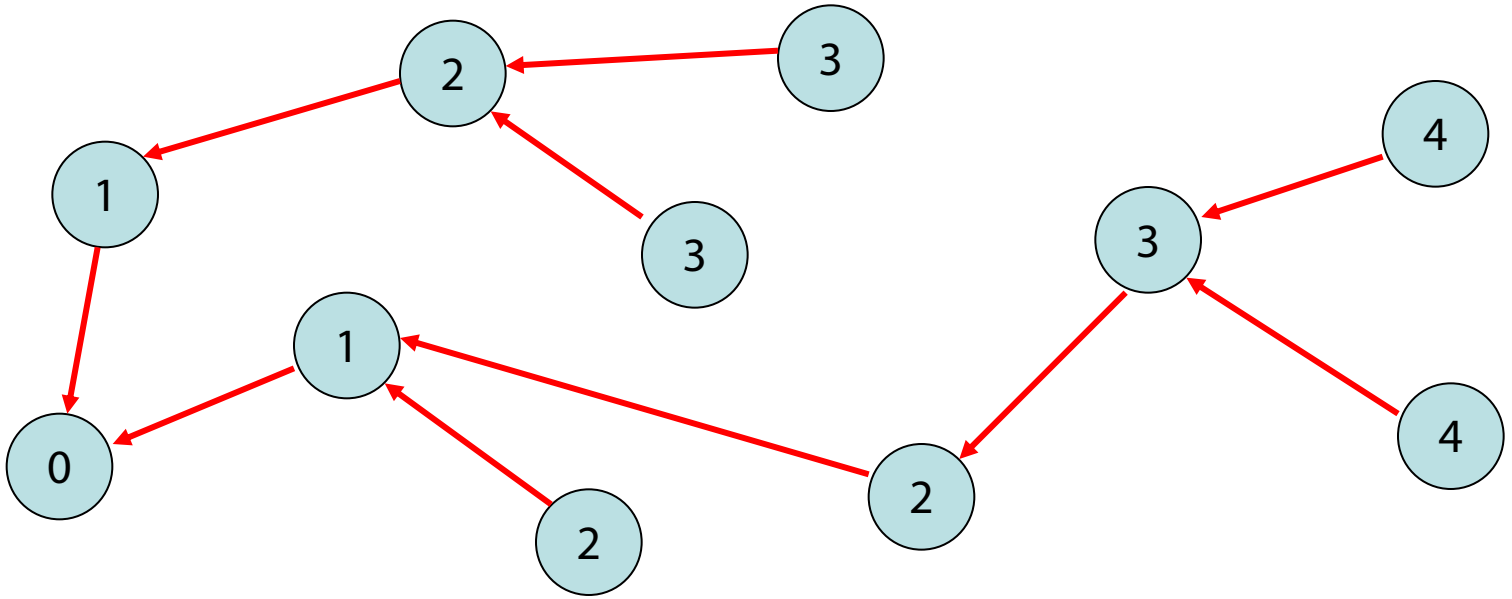
Distanzen:



Breitensuche

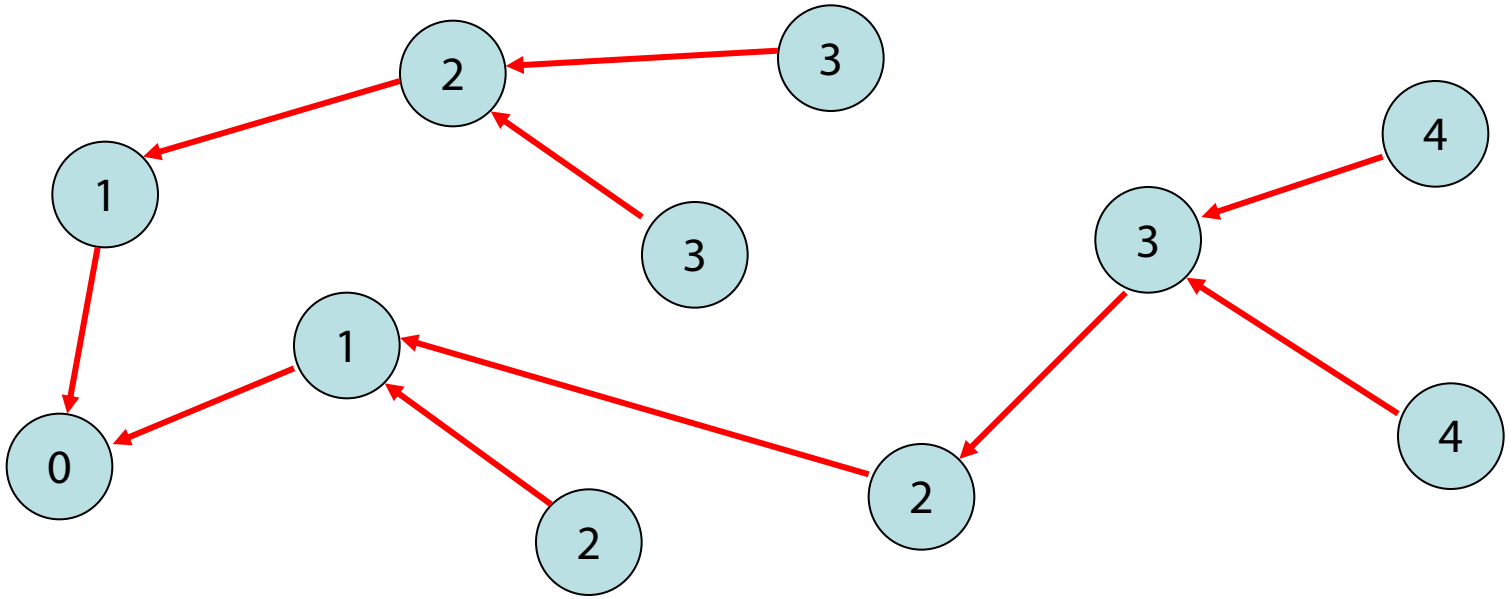
- d(v): Distanz von Knoten v zu s (d(s)=0)
- parent(v): Knoten, von dem v besucht

Mögliche Parent-Beziehungen in rot:



Breitensuche

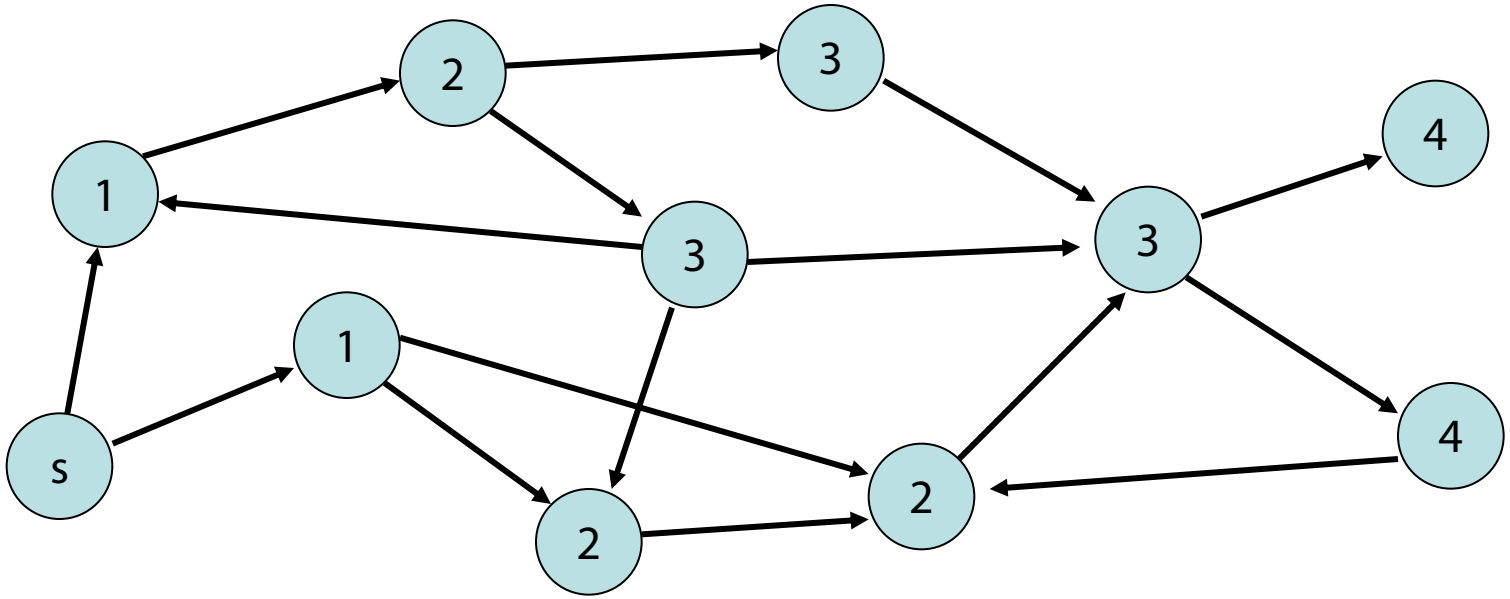
Parent-Beziehung eindeutig: wenn Knoten v zum erstenmal besucht wird, wird parent(v) gesetzt und dadurch v markiert, so dass v nicht nochmal besucht wird



Breitensuche

Kantentypen:

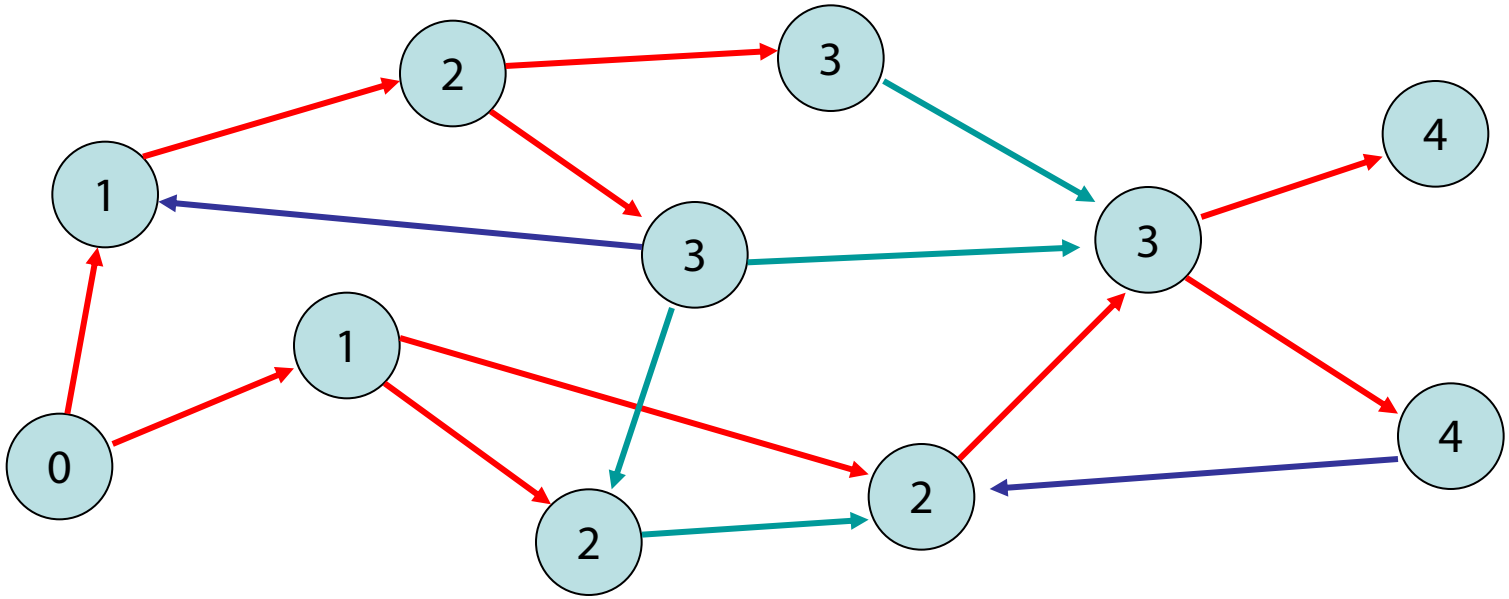
- Baumkante: zum Kind
- Rückwärtskante: zu einem Vorfahr
- Kreuzkante: alle sonstige Kanten



Breitensuche

Kantentypen:

- Baumkante: zum Kind
- Rückwärtskante: zu einem Vorfahr
- Kreuzkante: alle sonstige Kanten



Iteratoren

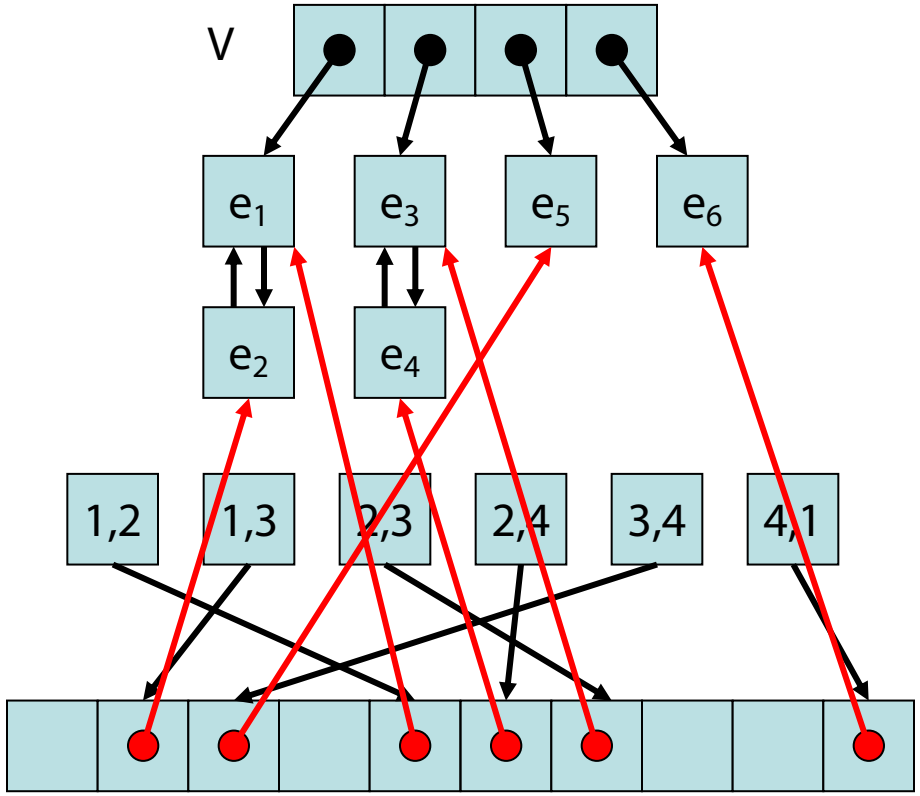
Sei G=(V, E) ein Graph

- Iteration über Kanten (u und v nicht instantiiert):
  - **foreach** (u,v)∈ E **do**
- Iteration über Knoten:
  - **foreach** v∈ V **do**
  - **foreach** (u,v)∈ E **do**, wobei
    - u instantiiert
      - Finde alle von u ausgehenden Kanten
    - oder v instantiiert
      - Finde alle bei v eintreffenden Kanten

Iteratoren

Zeitaufwand (grob):

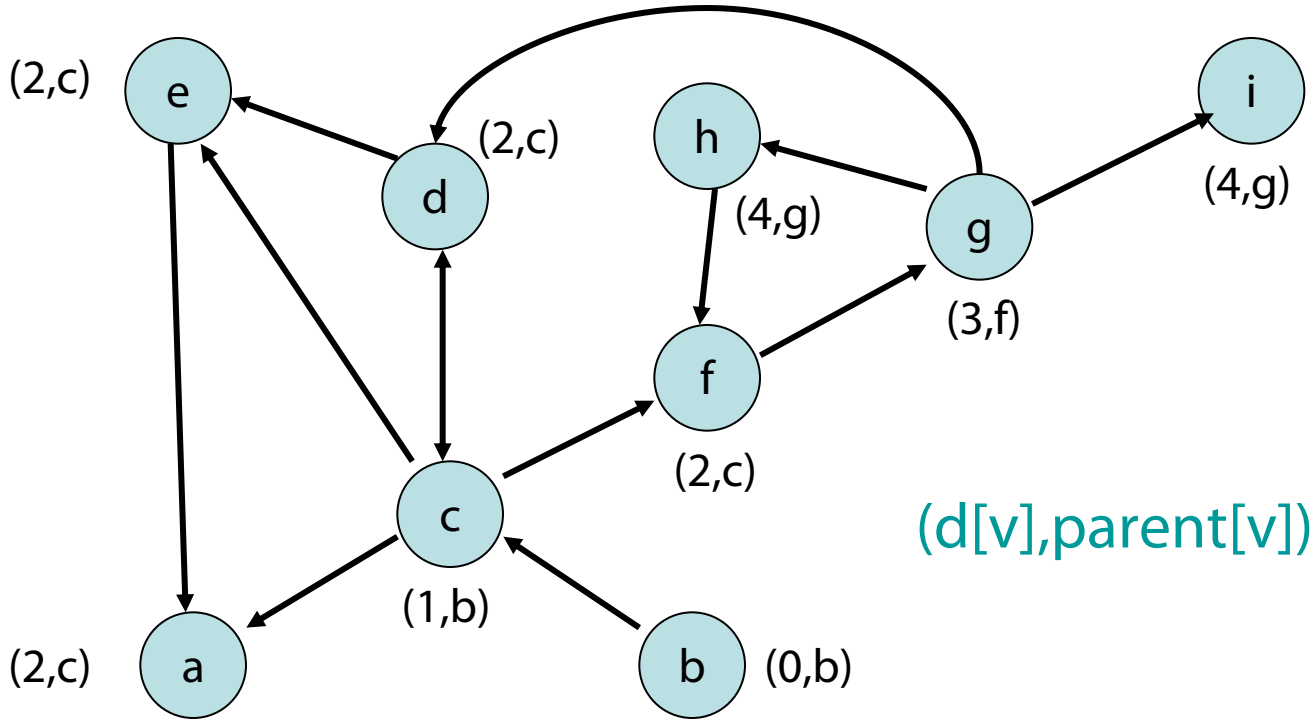
- getIteratorOut(v, G):
  O(1) (worst case)
- getIteratorIn(v, G):
  O(?) (worst case)



Breitensuche

```
Procedure BFS(s: Node)
    d = <∞,…,∞>: Array [1..n] of N
    parent = <⊥,…, ⊥>: Array [1..n] of Node
    d[key(s)]:=0              // s hat Distanz 0 zu sich
    parent[key(s)]:=s         // s ist sein eigener Vater
    q:=<s>: Queue             // q:Queue zu besuchender Knoten
    while not mtQueue?(q) do  // solange q nicht leer
       u:= dequeue(q)         // nimm Knoten nach FIFO-Regel
       foreach (u,v)∈ E do
          if parent[key(v)]=⊥ then  // v schon besucht?
             enqueue(v, q)  // nein, dann in q hinten einfügen
             d[key(v)]:=d[key(u)]+1
             parent[key(v)]:=u
```

BFS(b)

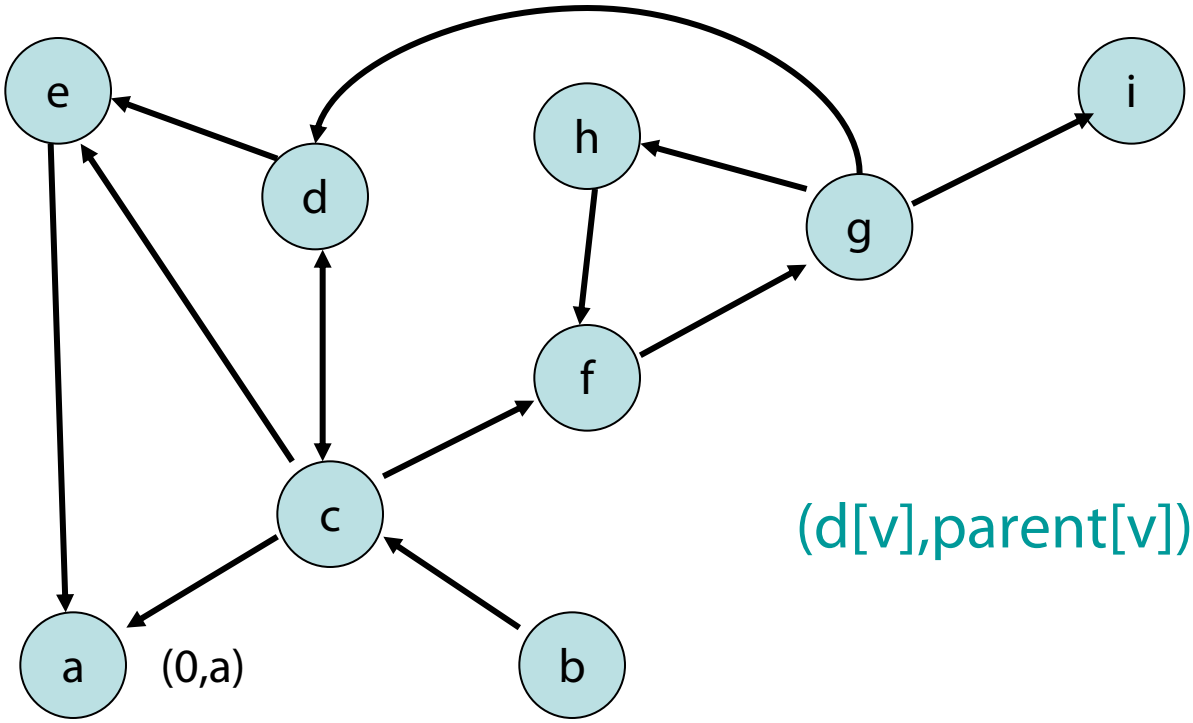


(d[v],parent[v])-Werte

: besucht, noch in q

: besucht, nicht mehr in q

BFS(a)



(d[v],parent[v])-Werte

Von a kein anderer Knoten erreichbar.

Zusammenfassung

- Repräsentationen für Graphen
  - Traversierung: Breitensuche

- Im nächsten Teil:
  - Traversierung: Tiefensuche
  - Gerichtete azyklische Graphen
  - Zusammenhangskomponenten