Kap. 5: Graphen

Carsten Gutwenger

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

17. VO DAP2 SS 2009 23. Juni 2008

Motivation

„Warum soll ich heute hier bleiben?"

Graphen sind wichtig und machen Spaß!

„Was gibt es heute Besonderes?"

Reicht *Spaß* alleine nicht aus?

Überblick

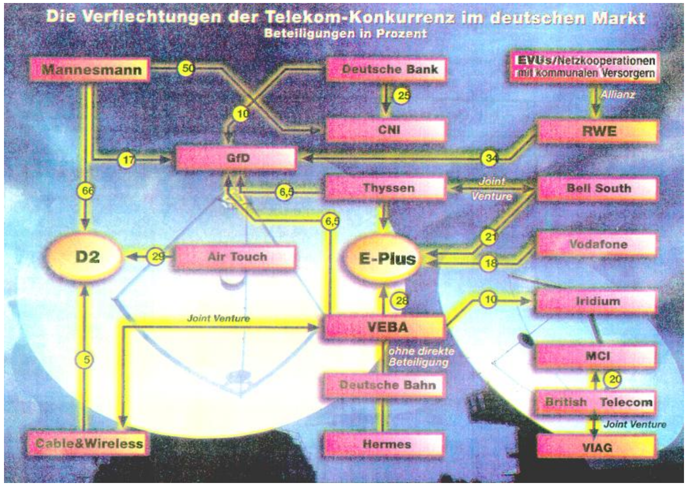
- Motivation

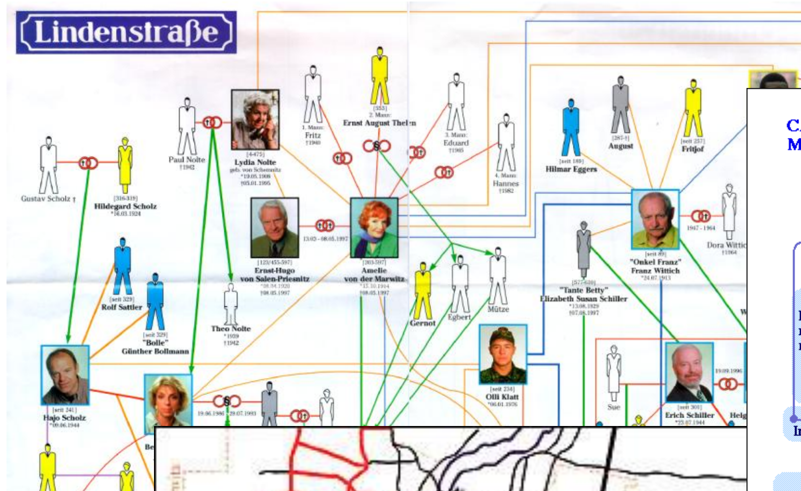
- Einführung von Graphen

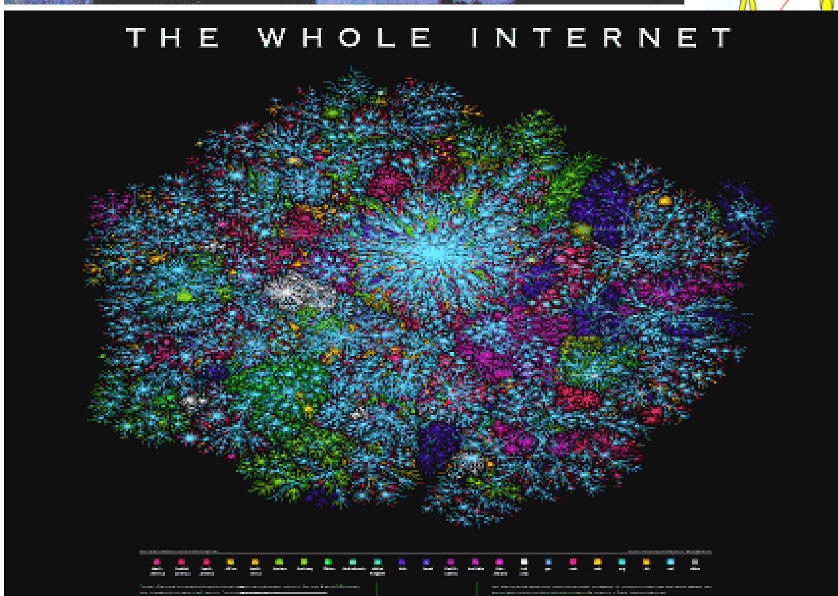
- Datenstrukturen

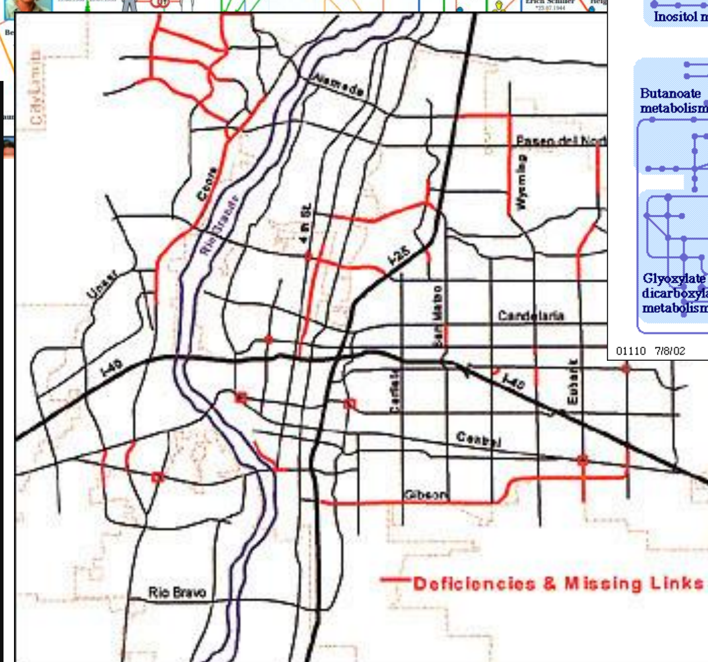
- Traversieren von Graphen:
  - Breitensuche (BFS)
  - Tiefensuche (DFS)

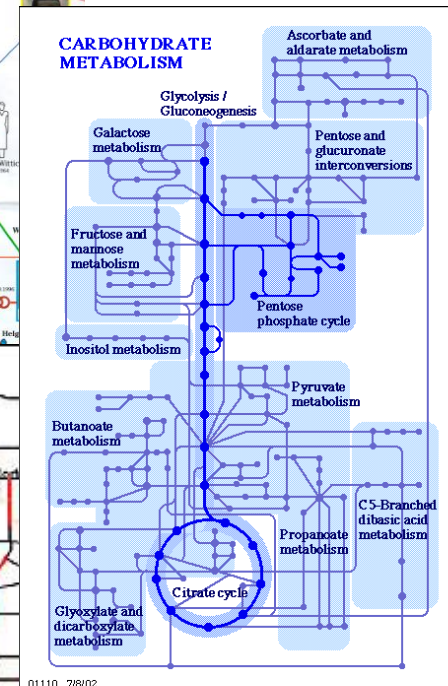
Rückblick


Die Verflechtungen der Telekom-Konkurrenz im deutschen Markt
Beteiligungen in Prozent
Mannesmann
Deutsche Bank
EVUs/Netzkooperationen mit kommunalen Versorgern
Allianz
CNI
RWE
GfD
Thyssen
Joint Venture
Bell South
D2
Air Touch
E-Plus
Vodafone
Joint Venture
Iridium
VEBA
ohne direkte Beteiligung
MCI
Deutsche Bahn
British Telecom
Joint Venture
Cable&Wireless
Hermes
VIAG


Lindenstraße


THE WHOLE INTERNET


Alameda
Paseo del Norte
Coors
Rio Grande
4th St.
I-25
Wyoming
Unser
San Mateo
Candelaria
I-40
Carlisle
Eubank
Central
Gibson
Rio Bravo
Deficiencies & Missing Links


CARBOHYDRATE METABOLISM
Ascorbate and aldarate metabolism
Glycolysis / Gluconeogenesis
Galactose metabolism
Pentose and glucuronate interconversions
Fructose and mannose metabolism
Pentose phosphate cycle
Inositol metabolism
Pyruvate metabolism
Butanoate metabolism
C5-Branched dibasic acid metabolism
Propanoate metabolism
Citrate cycle
Glyoxylate and dicarboxylate metabolism
01110 7/8/02

Motivation

- Graphen modellieren diskrete Strukturen
- hilfreich zur Analyse und Optimierung

- Straßen-, Bahnnetze: kürzeste Wege
- Modellierung von Prozessen, z.B. Geschäftsprozesse, Betriebsabläufe
- Proteininteraktionsnetzwerke in der molekularen Biologie

Kap. 6.1 Definition (Graph)

Graph $G=(V,E)$ besteht aus

- einer Menge V von Knoten
- einer (Multi-)menge E von Kanten, die Paaren von Knoten entsprechen.

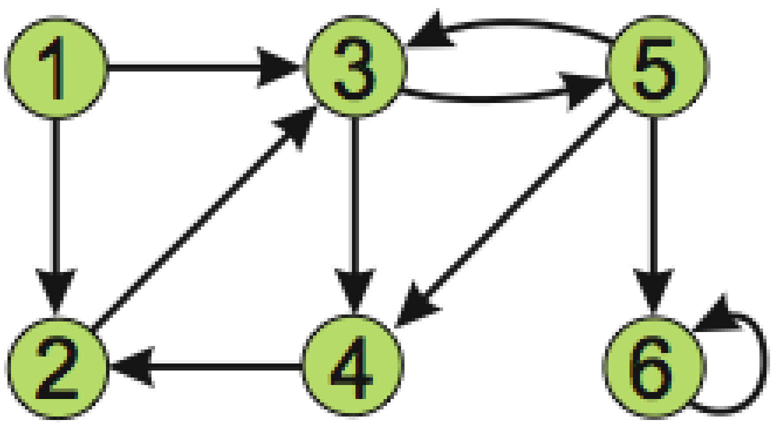
- Bei Multimenge kann ein Paar (v,w) mehrfach in E vorkommen $\rightarrow$ Mehrfachkanten
- Eine Kante (v,v) heißt Schleife (self-loop)

- Annahmen:
- V und E sind endliche Mengen
- Mehrfachkanten erlaubt

Gerichtete Graphen

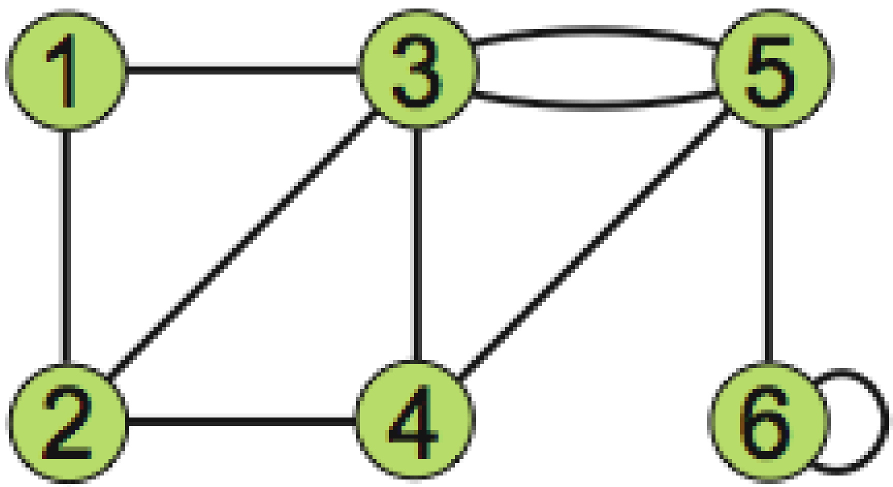
- Sind die Paare in E geordnet: $E \subseteq V \times V \rightarrow$ gerichteter Graph (Digraph)
- Kanten heißen dann: gerichtete Kanten (Bögen, directed edges, arcs)

- Maximale Kantenanzahl eines Digraphen ohne Schleifen und Mehrfachkanten: $|E| \leq |V| (|V|-1)$



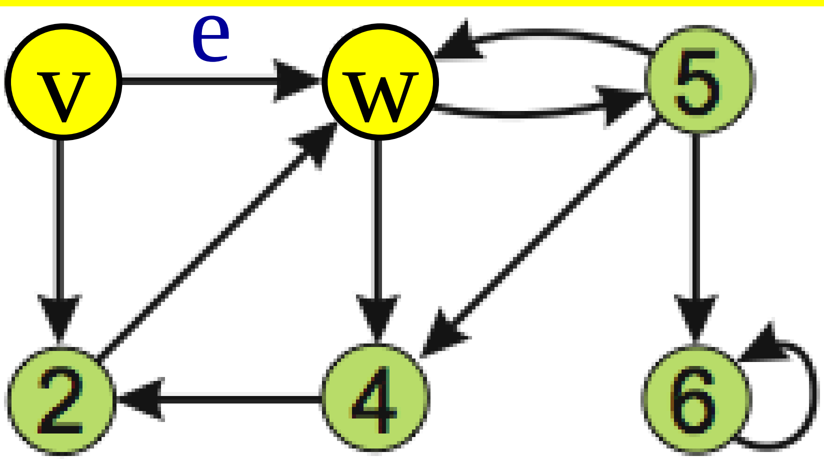
Ungerichtete Graphen

- Sind die Paare in E ungeordnet $\rightarrow$ (ungerichteter) Graph
- Kanten heißen dann: Kanten (edges)
- Maximale Kantenanzahl ohne Schleifen und Mehrfachkanten: $|E| \leq \frac{1}{2}\,|V|\,(|V|-1)$



Definitionen (Nachbarn)

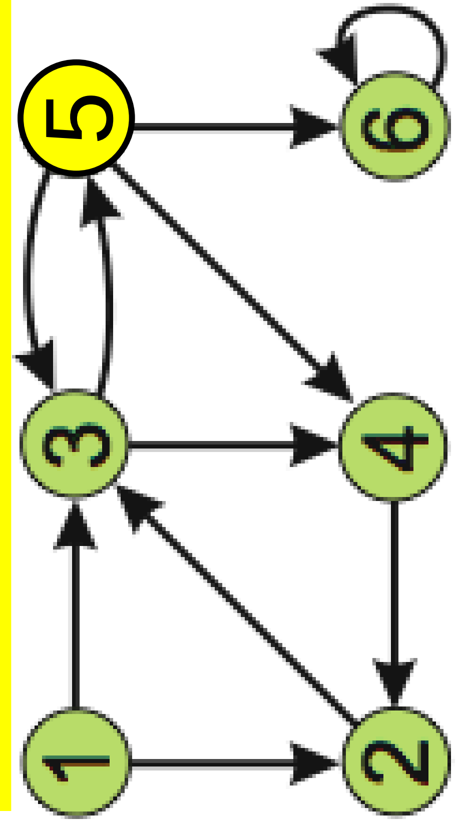
- Sei **e=(v,w)** eine Kante in E, dann sagen wir:
- v und w sind adjazent
- v (bzw. w) und e sind inzident
- v und w sind Endpunkte von e
- v und w sind Nachbarn
- e ist eine ausgehende Kante von v und eine eingehende Kante von w (falls G Digraph)



Definitionen für gerichtete Graphen G=(V,A)

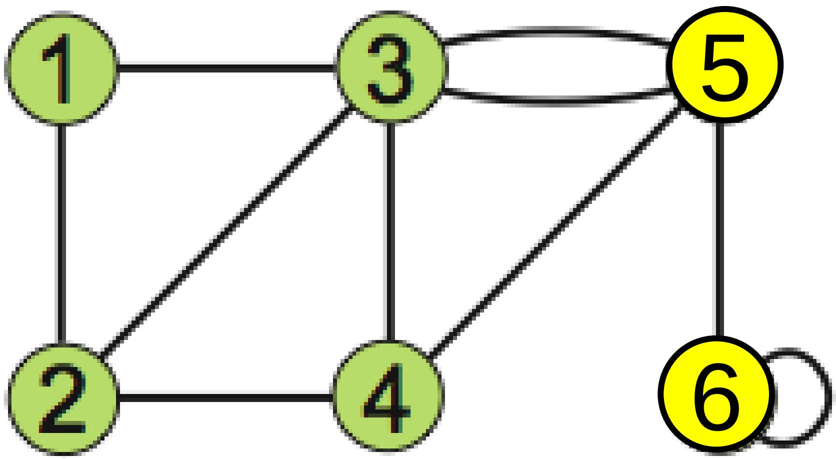
- Eingehende Nachbarmenge von $v \in V$: $N^+(v) := \{u \in V \mid (u,v) \in A\}$
- Ausgehende Nachbarmenge von $v \in V$: $N^-(v) := \{w \in V \mid (v,w) \in A\}$
- $A^+(v) :=$ Menge der eingehenden Kanten von v
- $A^-(v) :=$ Menge der ausgehenden Kanten von v
- $A(v) := A^+(v) \cup A^-(v)$
- Eingangsgrad $d^+(v) := |A^+(v)|$
- Ausgangsgrad $d^-(v) := |A^-(v)|$
- Knotengrad $d(v) := d^+(v) + d^-(v)$

$N^+(5) = \{3\}$
$N^-(5) = \{3,4,6\}$
$d^+(5) = 1$
$d^-(5) = 3$



Definitionen für ungerichtete Graphen G=(V,E)

- Nachbarmenge von $v \in V$:
  $N(v) := \{w \in V \mid (v,w) \in E\}$
- Menge der zu v inzidenten Kanten
  $E(v) := \{(u,v) \mid (u,v) \in E\}$
- Knotengrad $d(v)$ ist die Anzahl der zu v inzidenten Kanten, wobei eine Schleife 2 Mal gezählt wird



$N(5)=\{3,4,6\}$

$d(5)=4$

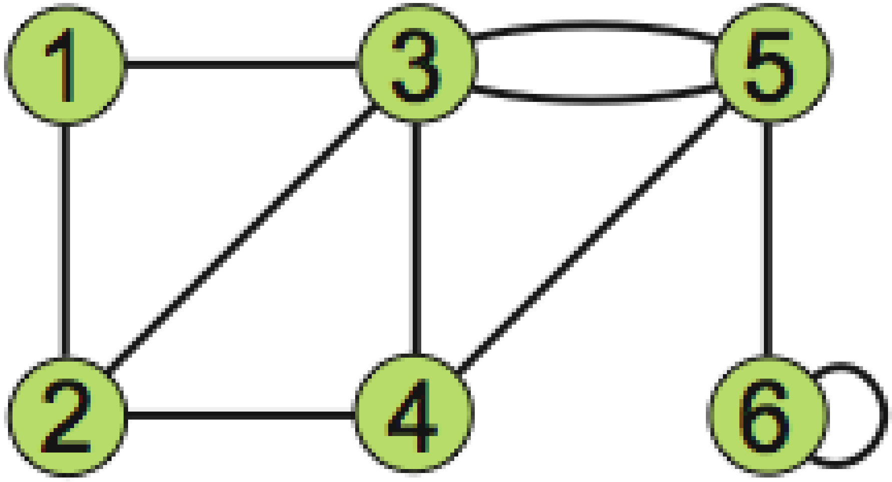
$d(6)=3$

Lemma (gerade Knotengrade)

- In einem ungerichteten Graphen $G=(V,E)$ ist die Anzahl der Knoten mit ungeradem Knotengrad gerade.

- Summiert man über alle Knotengrade, so zählt man jede Kante genau zweimal:

$$\sum_{v \in V} d(v) = 2\,|E|$$



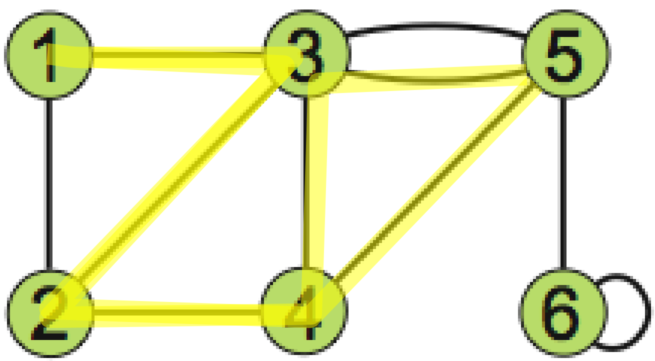
L.S.: gerade

R.S.: gerade

also auch die Anzahl der ungeraden Summanden

Definitionen (Wege)

- Sei $G=(V,E)$ gerichtet oder ungerichtet:
- Ein Kantenzug (walk) der Länge k ist eine nicht-leere Folge $v_0,e_1,v_1,e_2,\ldots, e_k,v_k$ von abwechselnd Knoten und Kanten aus G mit $e_i=(v_{i-1},v_i)$ für $i=1,\ldots,k$.
- Man schreibt auch: $v_0,v_1,\ldots, v_k$
- Ein Weg (path) ist ein Kantenzug in dem alle Knoten verschieden sind.

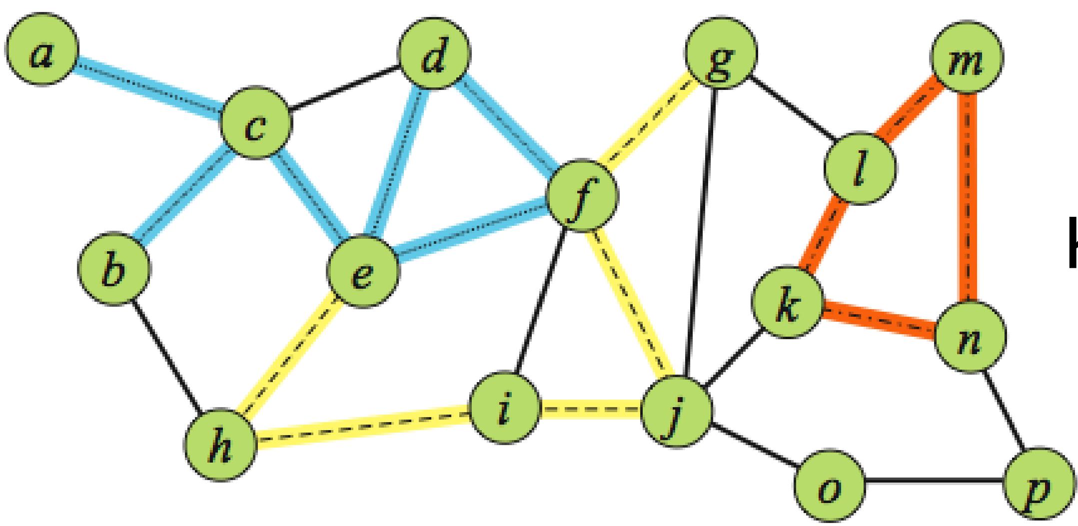


Kantenzug: 1,3,2,4,3,5

Weg: 1,3,2,4,5

Definitionen (Kreis)

- Sei $G=(V,E)$ gerichtet oder ungerichtet:
- Ist $v_0,e_1,v_1,e_2,\ldots, e_{k-1},v_{k-1}$ ein Weg mit $k \geqq 3$ und $e_k=(v_{k-1},v_0)$ eine Kante aus G, dann ist $v_0,e_1,v_1,e_2,\ldots,e_{k-1},v_{k-1},e_k,v_0$ ein Kreis der Länge k in G.



Kantenzug: a,c,e,f,d,e,c,b

Weg: g,f,j,i,h,e

Kreis: k,l,m,n,k

Darstellung von Graphen im Rechner: Statische Graphen

- Im Folgenden sei $V=\{v_1,v_2,\ldots,v_n\}$

- 1. Möglichkeit: Adjazenzlisten

- **Idee:** Speichere für jeden Knoten seine Nachbarmenge in einer Liste

- Realisierung: z.B. Knoten in Array und Nachbarkanten jedes Knotens als einfach verkettete Liste

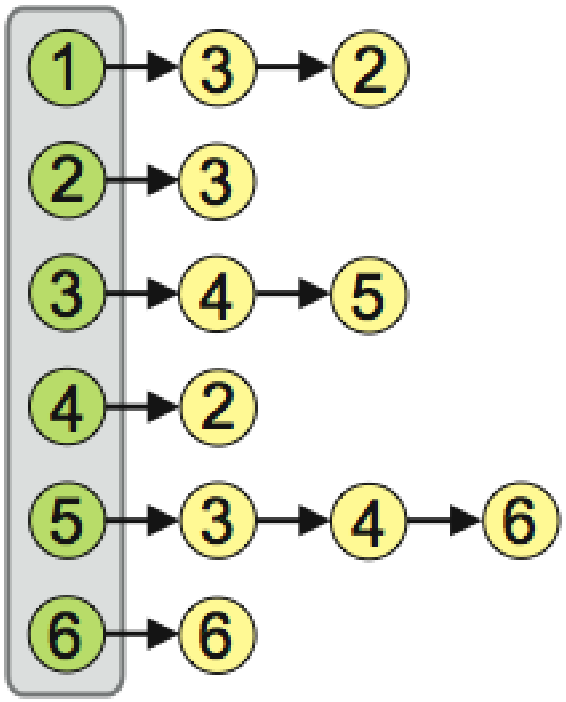
Darstellung von Graphen im Rechner: Statische Graphen

- 2. Möglichkeit: Adjazenzmatrix

- **Idee:** Eine $V x V$ Matrix enthält 0/1-Einträge für jedes Knotenpaar $\{u,v\}$

- Realisierung:
- Sei $M=(m_{i,j})$ eine n x n Matrix mit m_{ij}:=1 falls $(v_i,v_j) \in E$, und $m_{ij}:=0$ sonst.
- bei Mehrfachkanten schreibe statt 1 die Anzahl der Kanten

Darstellung gerichteter Graphen

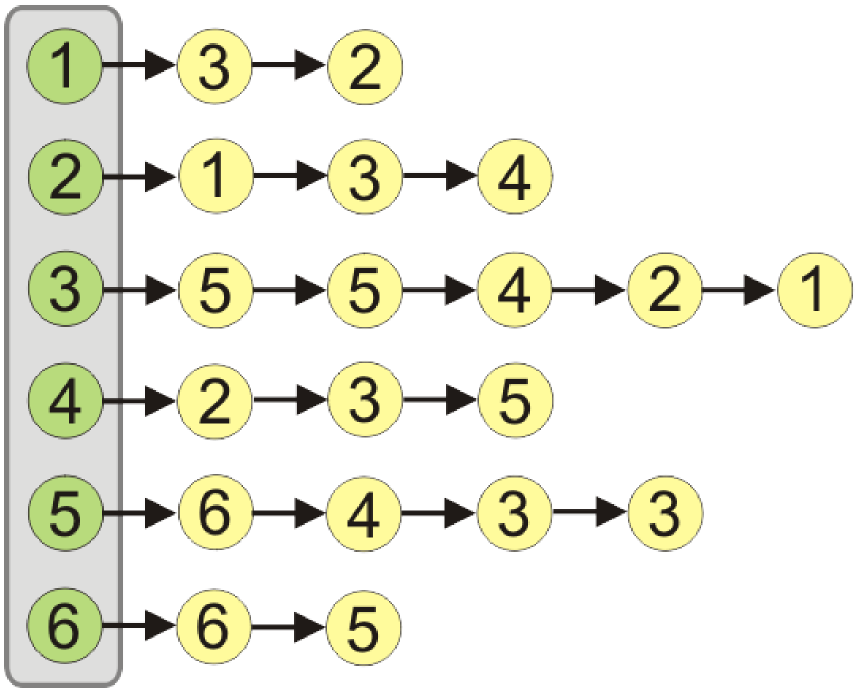


Adjazenzlisten

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	1	0
4	0	1	0	0	0	0
5	0	0	1	1	0	1
6	0	0	0	0	0	1

Adjazenzmatrix

Darstellung ungerichteter Graphen



Adjazenzlisten

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	1	1	0	0
3	1	1	0	1	2	0
4	0	1	1	0	1	0
5	0	0	2	1	0	1
6	0	0	0	0	1	1

Adjazenzmatrix

ist symmetrisch: nur Speicherung der oberen Hälfte

Diskussion

HIER: ab jetzt Adjazenzlisten

für **dünne** Graphen vorzuziehen!

Adjazenzliste:

- Speicherplatzverbrauch: linear: $\Theta(|V|+|E|)$
- Zeit für Aufbau: linear: $\Theta(|V|+|E|)$
- Abfrage, ob Kante (u,v) existiert: $\Theta(d(v))$
- Iteration über alle Nachbarn von $v \in V$: $\Theta(d(v))$

Adjazenzmatrix:

- Speicherverbrauch immer quadratisch: $\Theta(|V|^2)$
- Zeit für Aufbau: immer quadratisch: $\Theta(|V|^2)$
- Abfrage, ob Kante (u,v) existiert: $\Theta(1)$
- Iteration über alle Nachbarn von $v \in V$: $\Theta(|V|)$

Definitionen

- Die Dichte (density) eines Graphen G ist das Verhältnis $|E| / |V|$.
- G heißt dünn, falls seine Dichte $O(1)$ ist
- G heißt dicht, falls seine Dichte $\Omega(|V|)$ ist.

Eigentlich:
Betrachte Familie G_1, G_2, … von Graphen!

Darstellung von Graphen im Rechner: Dynamische Graphen

- Dynamisch unter den Operationen:
  - Hinzufügen neuer Knoten und Kanten
  - Entfernen von Knoten und Kanten

- **Idee:** für gerichtete Graphen:
- Inzidenzlisten: speichere ein- und ausgehende Knoten bei v
- Knoten in doppelt verketteter Liste (damit Entfernen in konstanter Zeit)
- Inzidenzlisten in doppelt verketteten Listen

Realisierung dynamischer Graphen: Liste für die Knoten

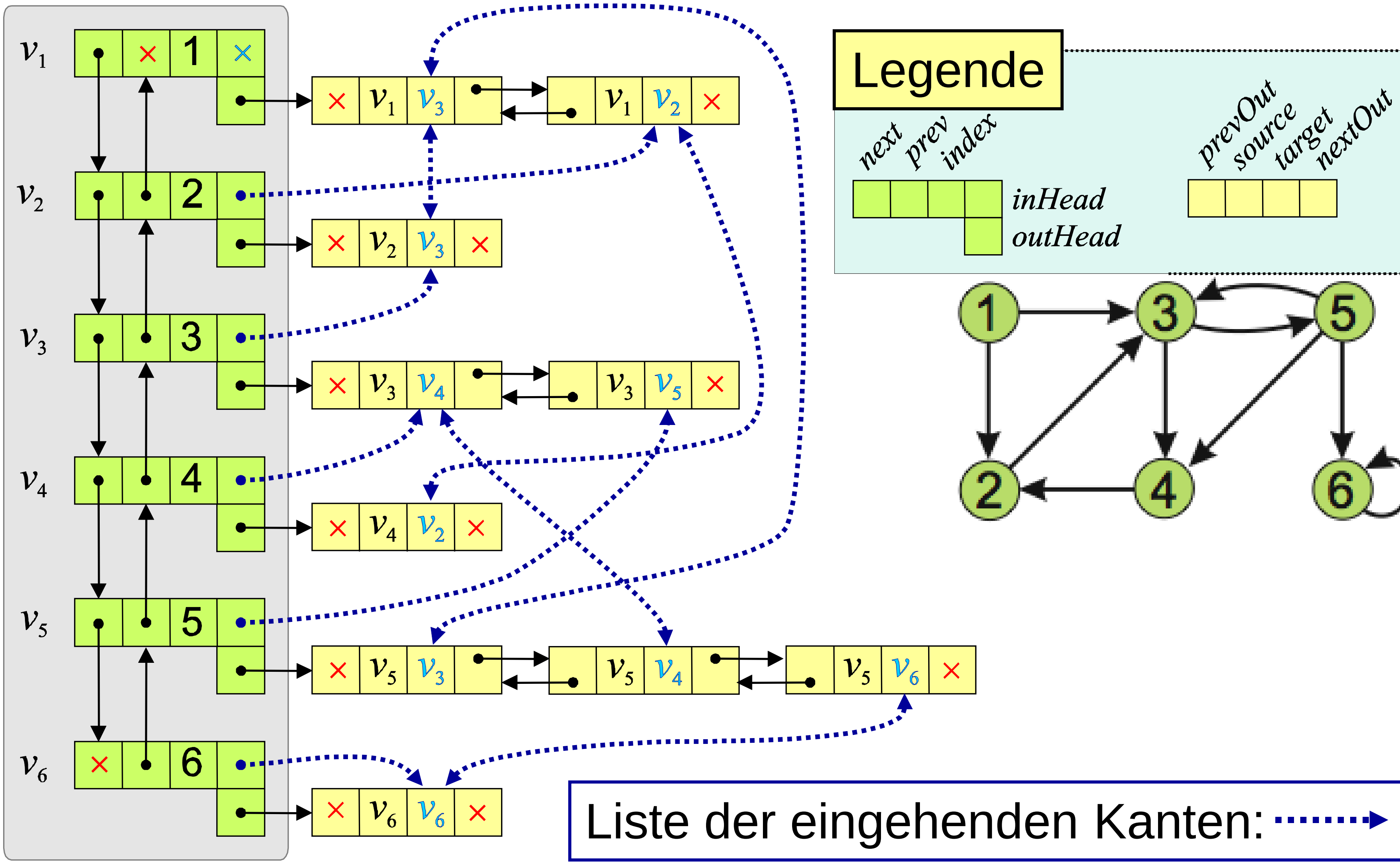
```
struct Node
   var Node prev         // Vorgänger Knotenliste
   var Node next         // Nachfolger in Knotenliste
   var Edge outHead      // Listenanfang ausgeh. Kanten
   var Edge inHead       // Listenanfang eingeh. Kanten
   var int index         // fortlaufender Index
end struct
```

Realisierung dynamischer Graphen: Liste für die Kanten

```
struct Edge
   var Edge prevOut  // Vorgänger in Liste ausg. Kanten
   var Edge nextOut  // Nachfolger in Liste ausg. Kanten
   var Edge prevIn   // Vorgänger in Liste eing. Kanten
   var Edge nextIn   // Nachfolger in Liste eing. Kanten
   var node source   // Anfangsknoten der Kante
   var node target   // Endknoten der Kante
end struct
```

Achtung: jeder Kanteneintrag ist genau einmal in Liste enthalten

Darstellung von Graphen im Rechner: Dynamische Graphen



Analyse Dynamischer Graphen

- Speicherplatzverbrauch: linear: $\Theta(|V|+|A|)$
- Zeit für Aufbau: linear: $\Theta(|V|+|A|)$
- Abfrage, ob Kante (u,v) existiert: $\Theta(d(v))$
- Iteration über alle Nachbarn von $v \in V$: $\Theta(d(v))$
- Iteration über alle ausg. Kanten von $v \in V$: $\Theta(d^-(v))$
- Iteration über alle eing. Kanten von $v \in V$: $\Theta(d^+(v))$
- Einfügen eines Knotens bzw. Kante: $\Theta(1)$
- Entfernen einer Kante: $\Theta(1)$
- Entfernen eines Knotens: $\Theta(d(v))$

Man geht davon aus, dass man jeweils Zeiger auf die Knoten und beim Entfernen auch auf die Kanten gegeben hat

Kap. 6.3 Traversieren von Graphen

Traversieren: systematisches Durchwandern von Graphen

HIER: ungerichtete Graphen

Def.: Der graphentheoretische Abstand zweier Knoten u, v eines ungerichteten Graphen G ist die Länge des kürzesten Weges von u nach v, falls ein solcher existiert, sonst ∞.

Achtung: hierbei werden keine vorgegebenen Kantenlängen bzw. Kantengewichte berücksichtigt.

Kap. 6.3.1 Breitensuche (BFS)

engl.: Breadth-first-search, BFS

Idee: besuche die Knoten nach aufsteigendem graphentheoretischen Abstand zu einem vorher festgelegten Startknoten.

1. **Fall:** Wir sehen v zum ersten Mal (von u aus):
   - Dann muss $\mathrm{dist}(v)$ um genau 1 größer sein als $\mathrm{dist}(u)$.
   - Wir können v besuchen, nachdem wir alle bisher gesehenen Knoten besucht haben.
2. **Fall:** wir haben v schon gesehen $\rightarrow$ nichts zu tun

Breitensuche (BFS)

Methode:

(1) Starte am Knoten $u:=s$. Sei $Q:=\emptyset$ eine Queue.

(2) Für alle Knoten $v \in N(u)$ // erforsche Knoten u

(3) Falls wir v zum ersten Mal sehen:

(4) markiere v als „gesehen“

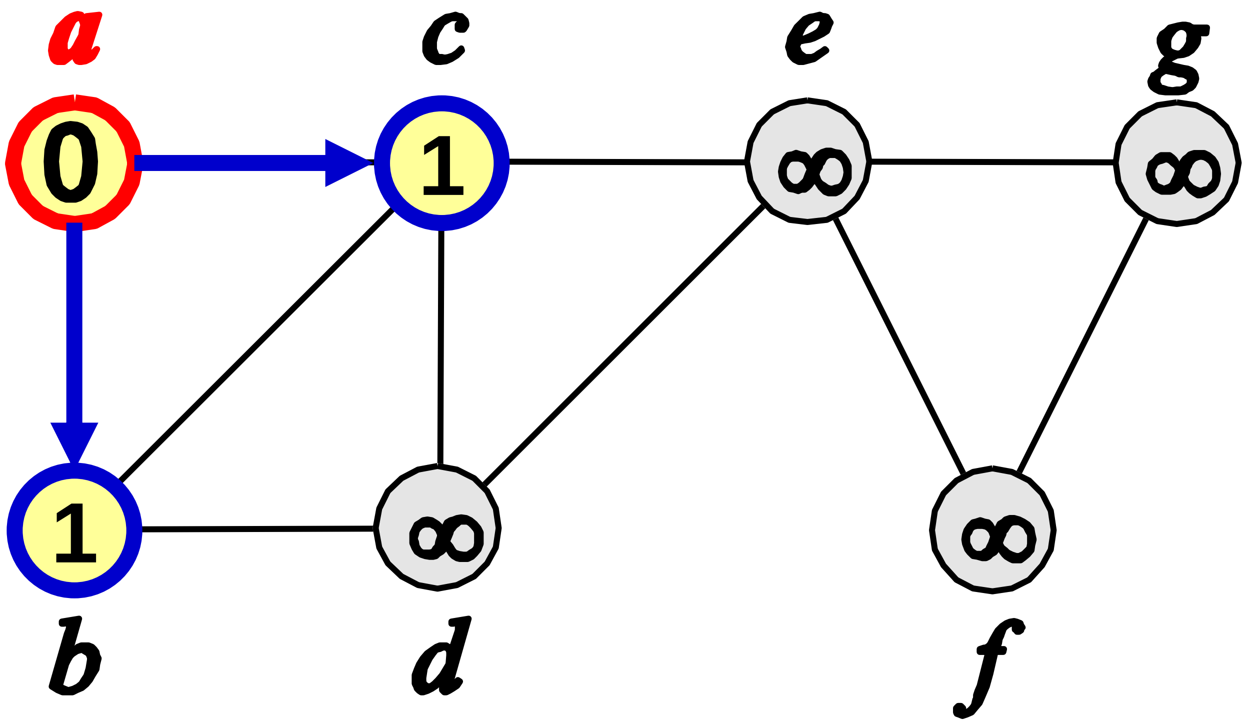
(5) $dist(v):=dist(u)+1$; merke Vorgänger;

(6) hänge v hinten an Q an.

(7) Sei u der nächste Knoten in Q. Gehe zu (2)

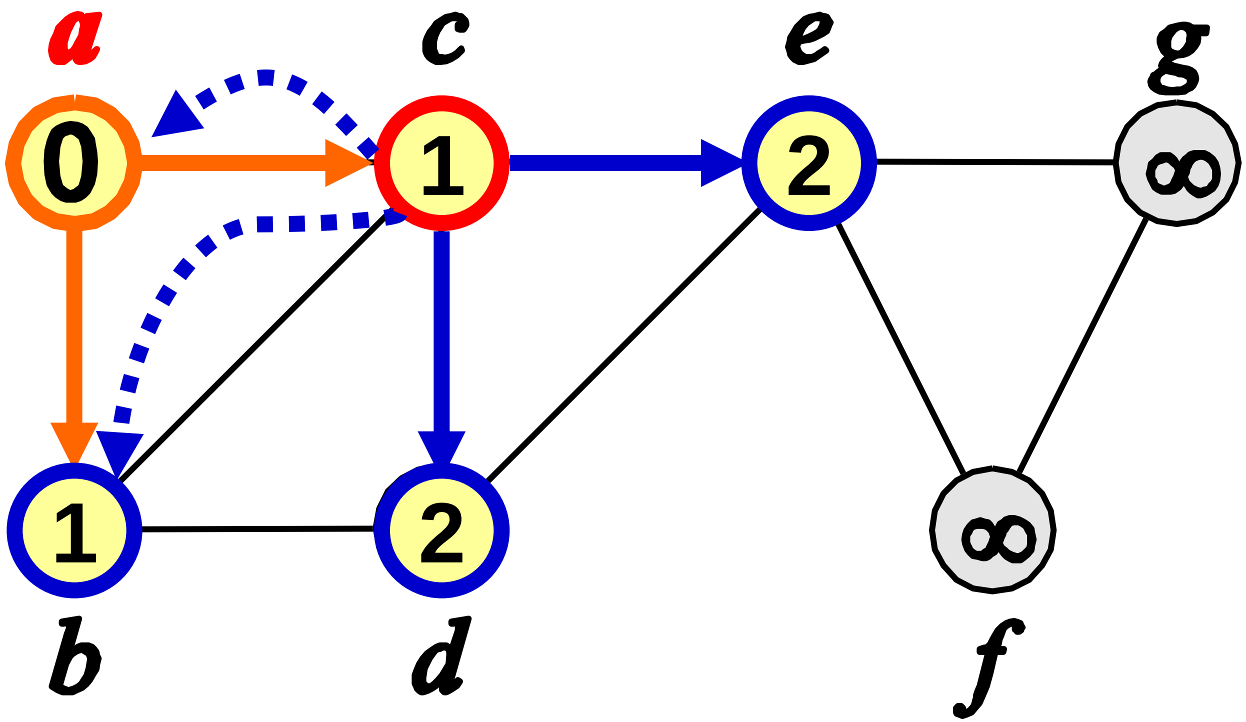
$dist(v)$ enthält den graphentheoretischen Abstand von u nach v; $\pi(v)$ den Vorgänger des kürz. Weges

Beispiel für BFS



Q	a	c	b

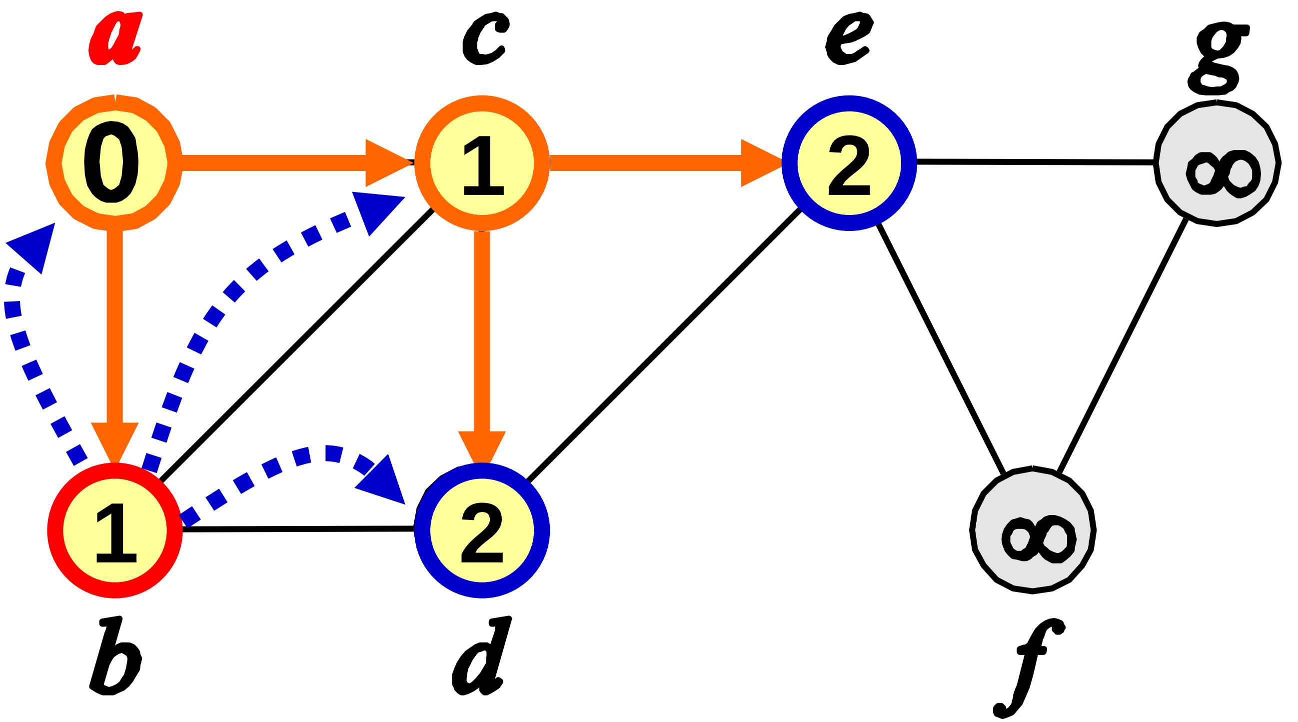
Beispiel für BFS



Q

c	b	e	d

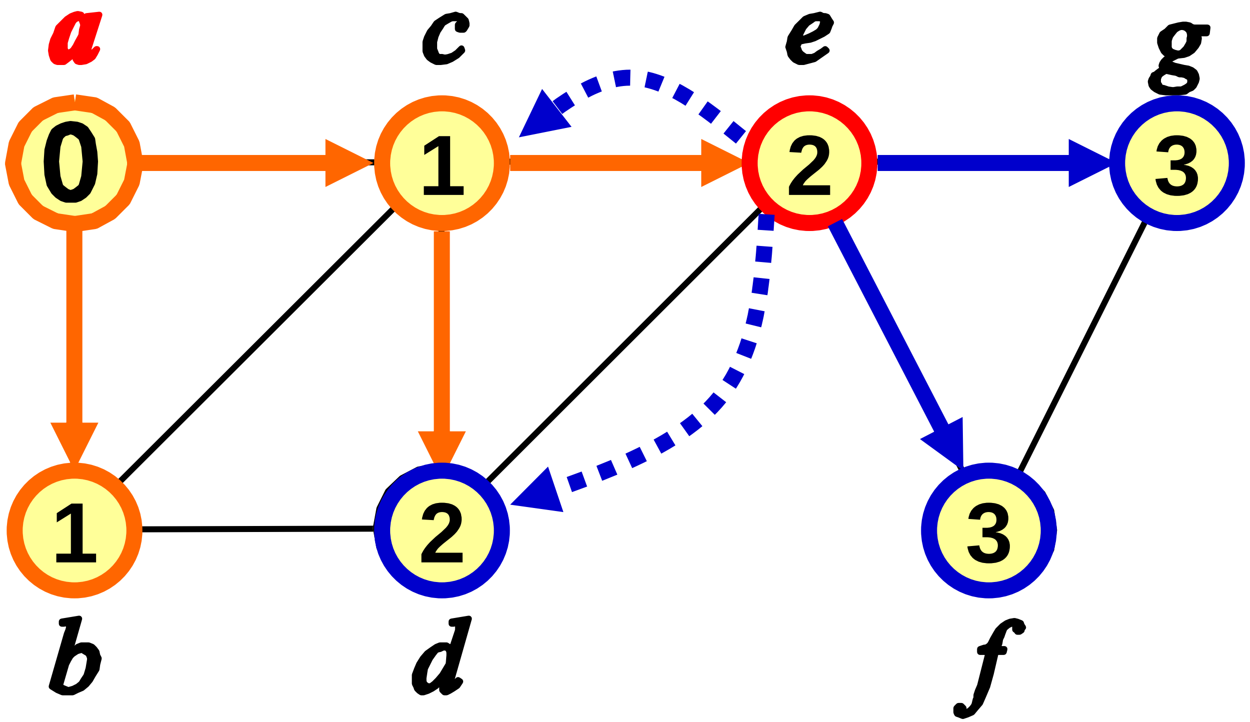
Beispiel für BFS



Q

b	e	d

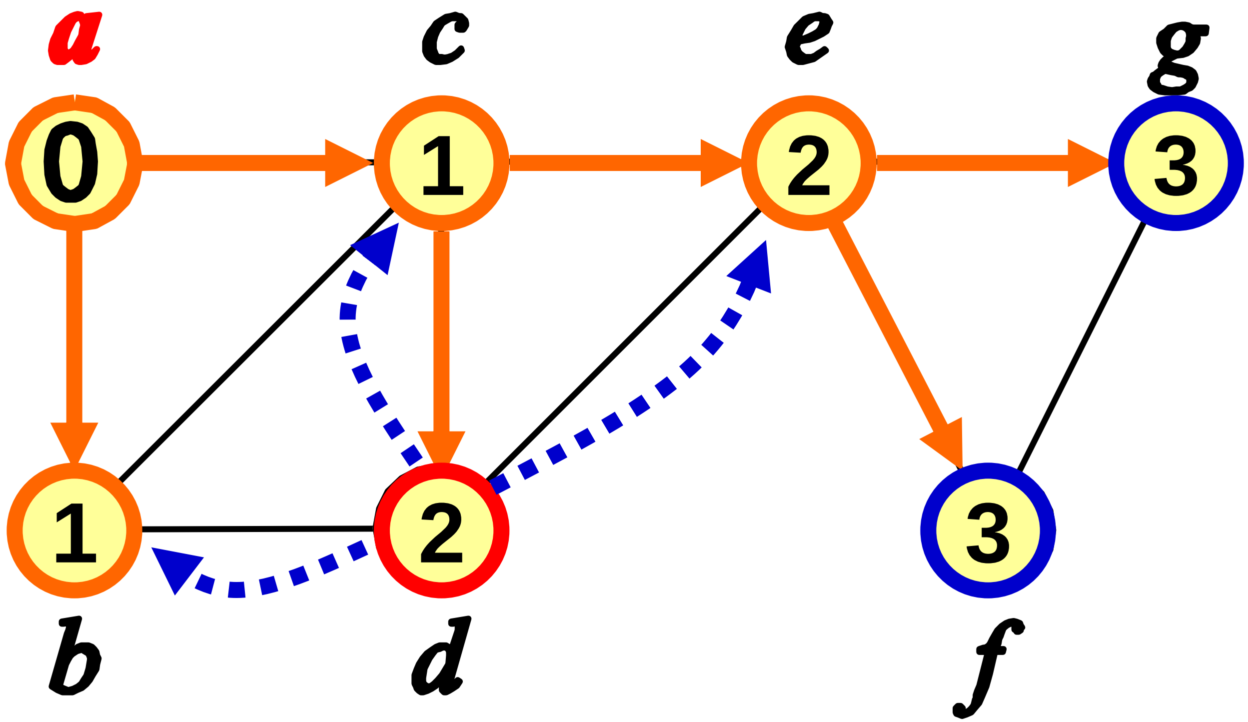
Beispiel für BFS



Q

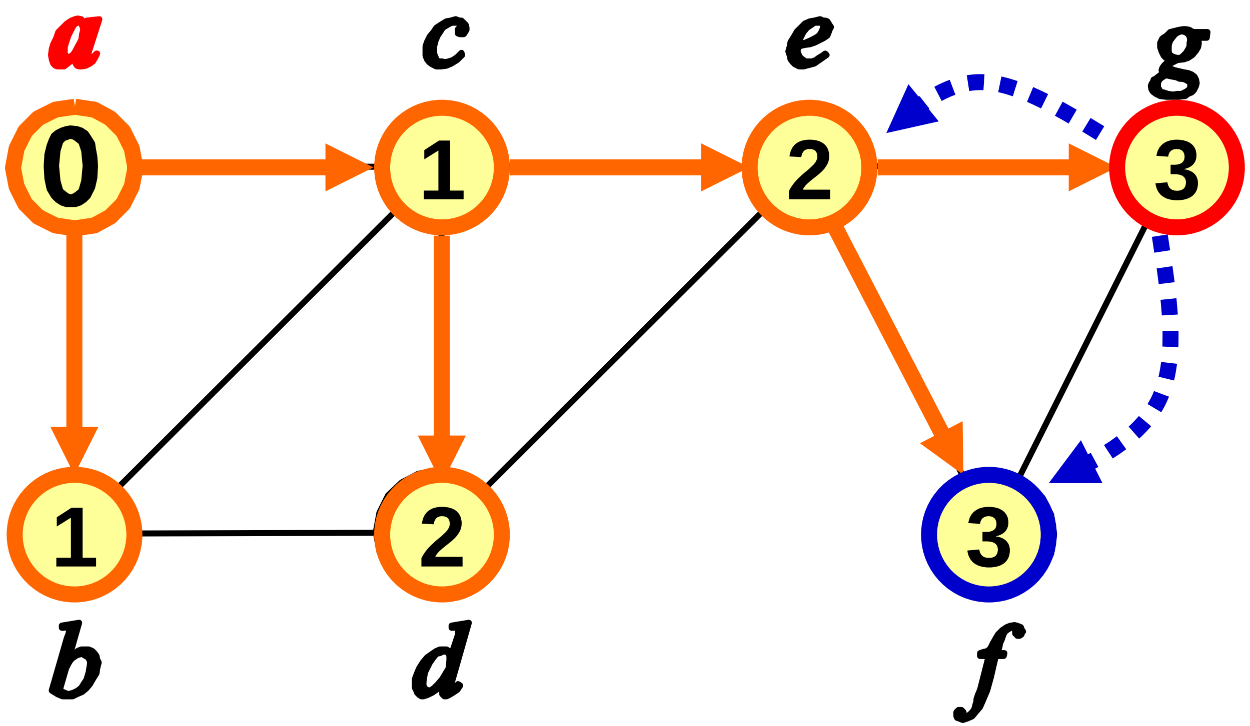
e	*d*	*g*	*f*

Beispiel für BFS



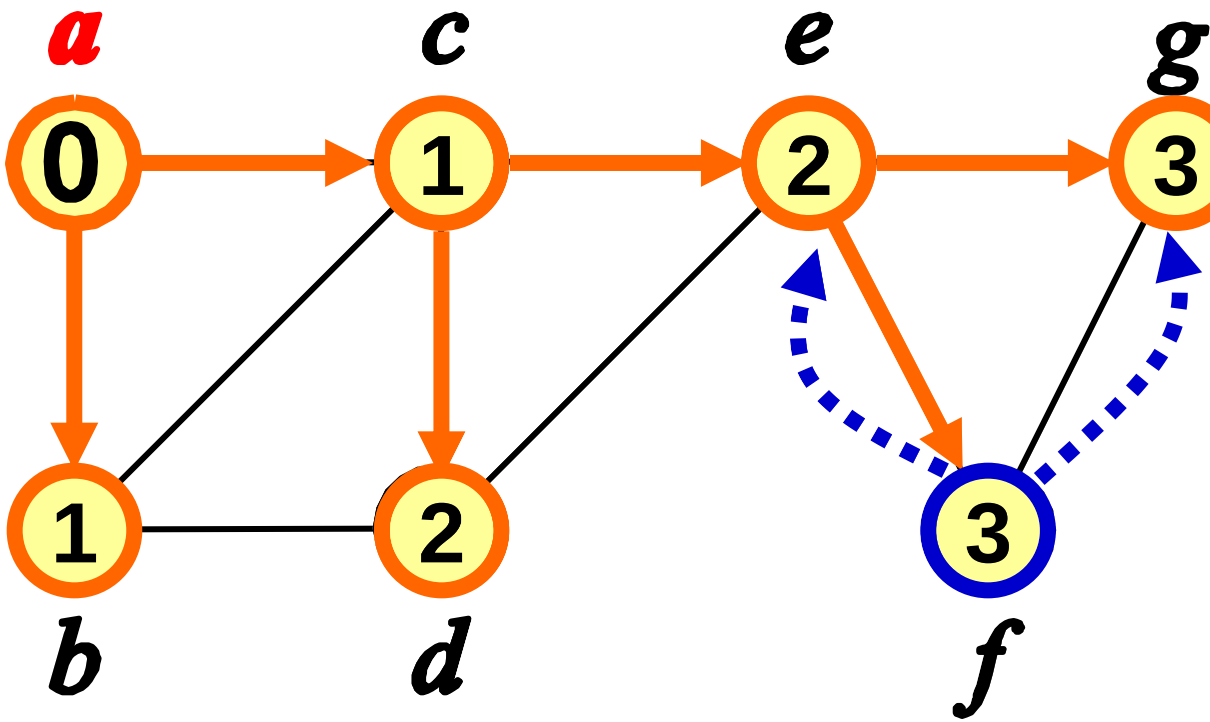
Q d g f

Beispiel für BFS



Q | *g* | *f*

Beispiel für BFS

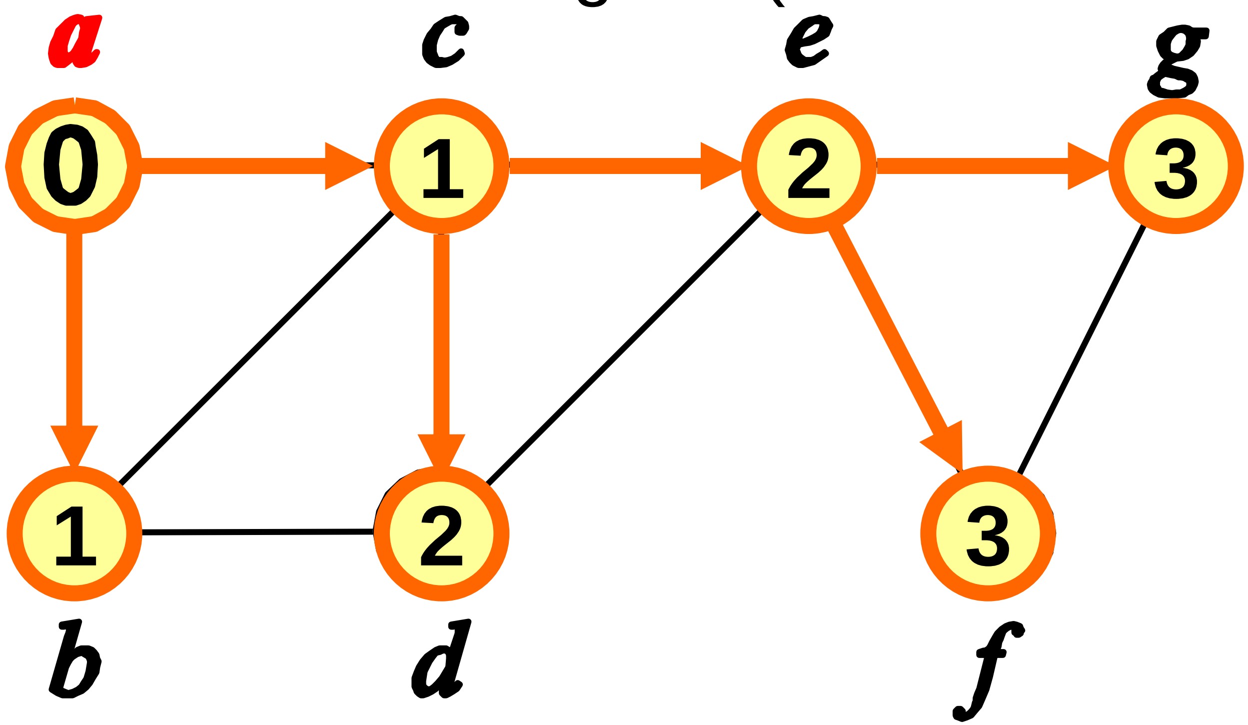


Q

f

Breitensuche entspricht einer
Traversierung in diesem BFS-Baum.

G zusammenhängend (sonst BFS-Wald)



Die orangen Kanten (diejenigen Kanten (u,v), die zum ersten Mal v besuchen) bilden einen Baum: den BFS-Baum

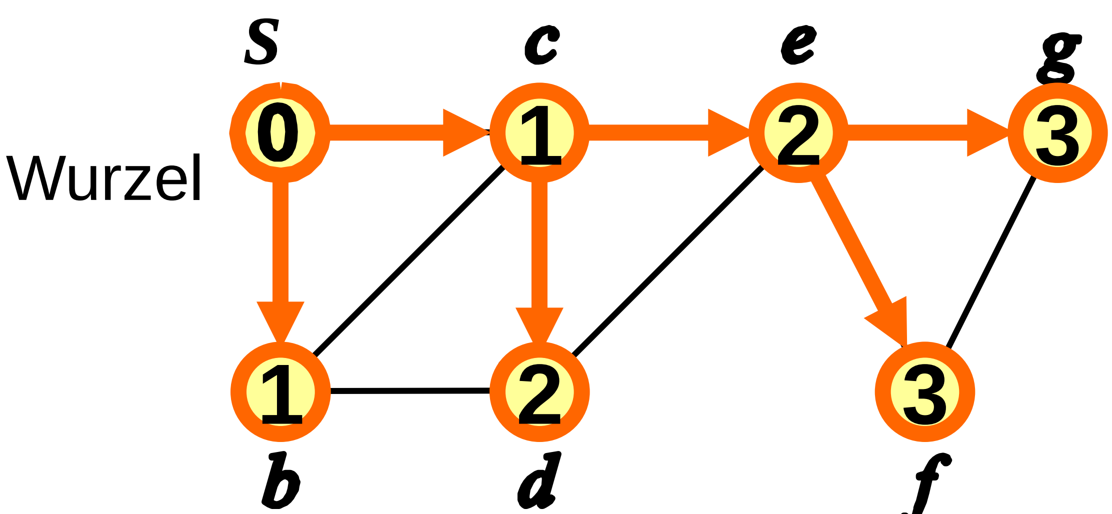
Algorithmus BFS(s)

Sei G=(V,E) ungerichteter Graph, s,u,v: Knoten

```
for all v∈V \ {s} do { marked[v]:=false; dist[v]:=∞}
Q.PUT(s); marked[s]:=true; dist[s]:=π[s]:=0
while not Q.ISEMPTY() do {
    u:=Q.GET()
    for all v∈N(u) do {
        if not marked[v] then {
            Q.PUT(v)
            marked[v]:=true
            dist[v]:=dist[u]+1; π[v]:=u
} } }
```

BFS-Baum

- Im Feld $\pi[v]$ werden die Vorgänger von v gespeichert.
- Die Kanten $(\pi(v),v)$ bilden einen Baum mit Wurzel s.
- Die Höhe des BFS-Baumes mit Startknoten s ist eindeutig bestimmt.
- Die Tiefe eines Blattes v entspricht dem graphentheoretischen Abstand von v zu s



Analyse von BFS

- Initialisierung: $\Theta(|V|)$
- Die **while**-Schleife wird für jeden von s aus erreichbaren Knoten genau einmal durchlaufen, da jeder Knoten höchstens einmal in die Queue kommt.
- Die **for all**-Schleife durchläuft für jeden Knoten die Liste seiner Nachbarn $N(v)$, das ist also jeweils $\Theta(d(v))$ Aufwand
- Gesamtaufwand: $\Theta(|V|)+\sum_{v \in V} \Theta(d(v))= \Theta(|V|+|E|)$,

  da im Worst Case alle Knoten von s aus erreichbar sein können

Problem USSSP

Unweighted Single-Source Shortest Path (USSSP):

- **Gegeben:** ungerichteter Graph $G=(V,E)$
- **Gesucht:** kürzester Weg von s zu jedem Knoten in G.

Breitensuche (BFS)

Theorem: Sei $G=(V,E)$ ein ungerichteter Graph und $s \in V$. Dann löst der Algorithmus $BFS(s)$ das Unweighted Single-Source Shortest Path (USSSP) für Startknoten s in Zeit $O(|V|+|E|)$.