14. Elementare Graphalgorithmen

- Graphen sind eine der wichtigsten Modellierungskonzepte der Informatik
- Graphalgorithmen bilden die Grundlage vieler Algorithmen in der Praxis

Übersicht:

- Zunächst eine kurze Einführung in Graphen.
- Dann Darstellungen von Graphen
- Schließlich einfache Graphalgorithmen:
  Breiten- und Tiefensuche, Zusammenhangskomponenten, Minimalspannende Bäume

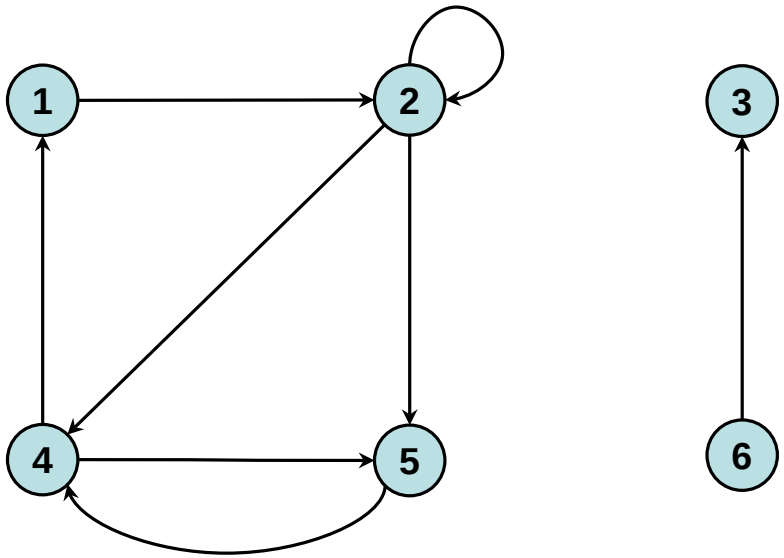
Wiederholung Graphen – Gerichtete Graphen

- Ein gerichteter Graph G ist ein Paar (V,E), wobei V eine endliche Menge ist und $E \subseteq V \times V$.
- Elemente aus V heißen Knoten, Elemente aus E heißen Kanten. Entsprechend heißt V Knotenmenge und E heißt Kantenmenge von G.
- Kanten sind geordnete Paare von Knoten. Kanten der Form (u,u), $u \in V$, sind zugelassen und heißen Schleifen.
- Ist $(u,v) \in E$, so sagen wir, dass die Kante von u nach v führt. Sagen auch, dass u und v adjazent sind. Müssen dann aber noch Richtung berücksichtigen.
- Der Grad eines Knotens v ist Anzahl Kanten (v,w) in E.

Illustration gerichteter Graph

$$V = \{1,2,3,4,5,6\}$$

$$E = \{(1,2),(2,2),(2,4),(2,5),(4,1),(4,5),(5,4),(6,3)\}$$



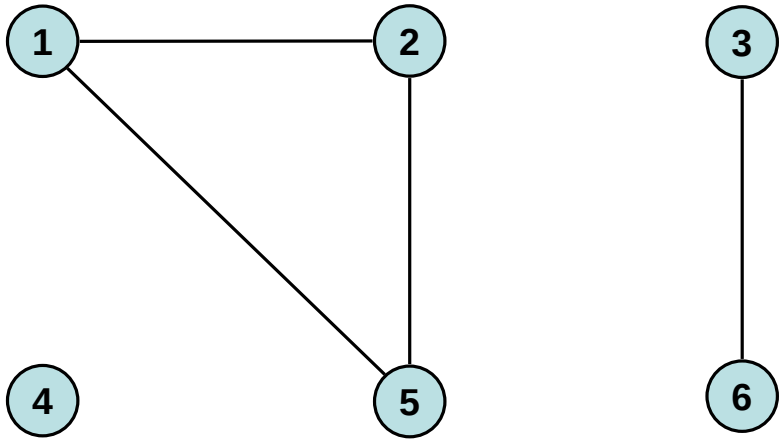
Wiederholung Graphen – Ungerichtete Graphen

- Ein ungerichteter Graph G ist ein Paar (V,E), wobei V eine endliche Menge ist und E eine Menge von 2-elementigen Teilmengen von V ist.
- Elemente aus V heißen Knoten, Elemente aus E heißen Kanten. Entsprechend heißt V Knotenmenge und E heißt Kantenmenge von G.
- Kanten haben die Form {u,v}. Kanten der Form {u,u} sind nicht zugelassen.
- Ist $\{u,v\} \in E$, so sagen wir, u und v adjazent sind. Der Grad von u ist die Anzahl der Kanten {u,v} in E.

Illustration ungerichteter Graph

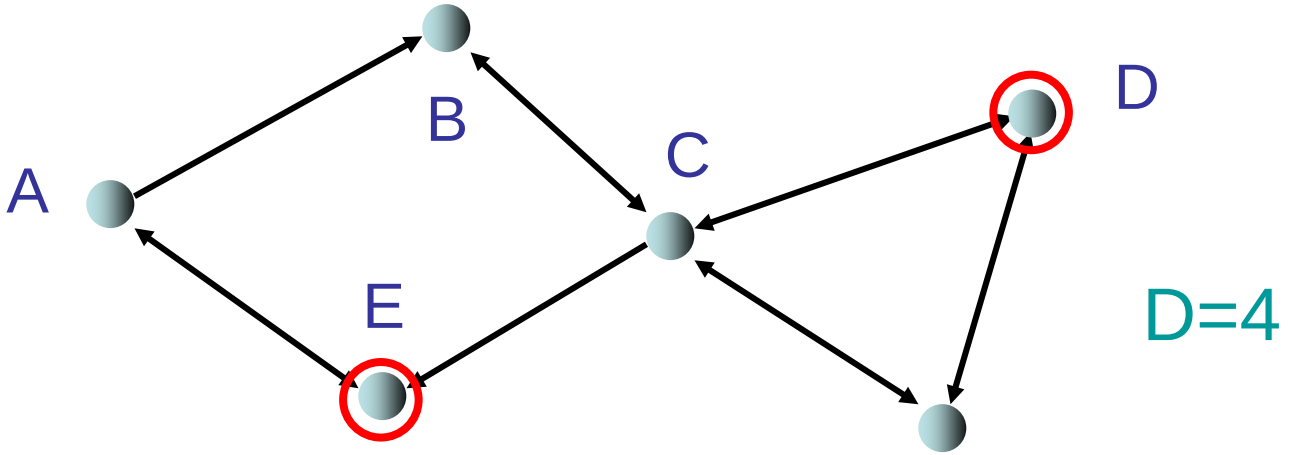
V={1,2,3,4,5,6}

E={ {1,2}, {1,5}, {2,5}, {3,6} }



Graphentheorie

- n: Anzahl Knoten, m: Anzahl Kanten
- $\delta(v,w)$: Distanz von w zu v in G
  - gerichteter Graph: Anzahl Kanten eines kürzesten gerichteten Weges von v nach w
  - ungerichteter Graph: Anzahl Kanten eines kürzesten Weges von v nach w
- $D=\max_{v,w} \delta(v,w)$: Durchmesser von G

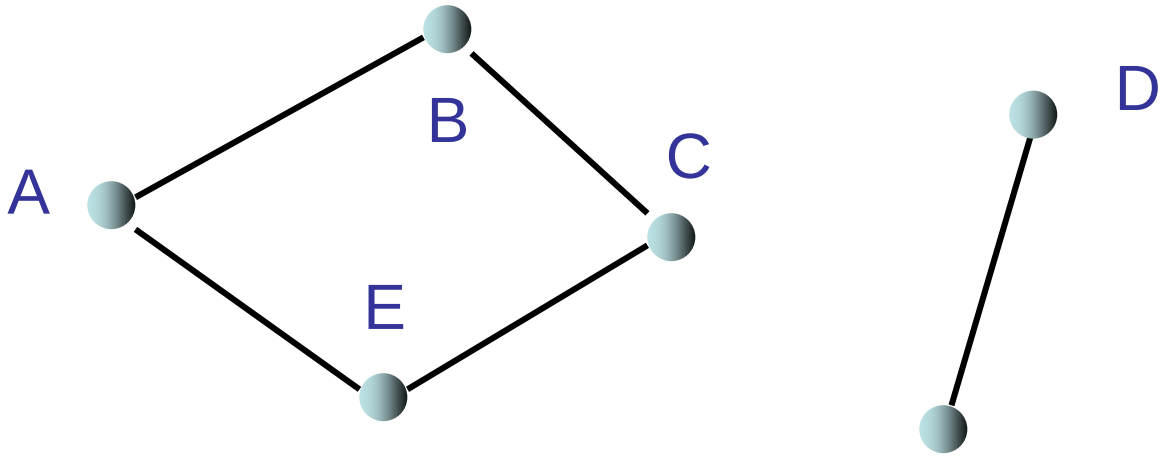


Graphentheorie

G ungerichtet:

- G ist zusammenhängend: Durchmesser D endlich (d.h. es gibt einen Weg von jedem Knoten zu jedem anderen Knoten in G)

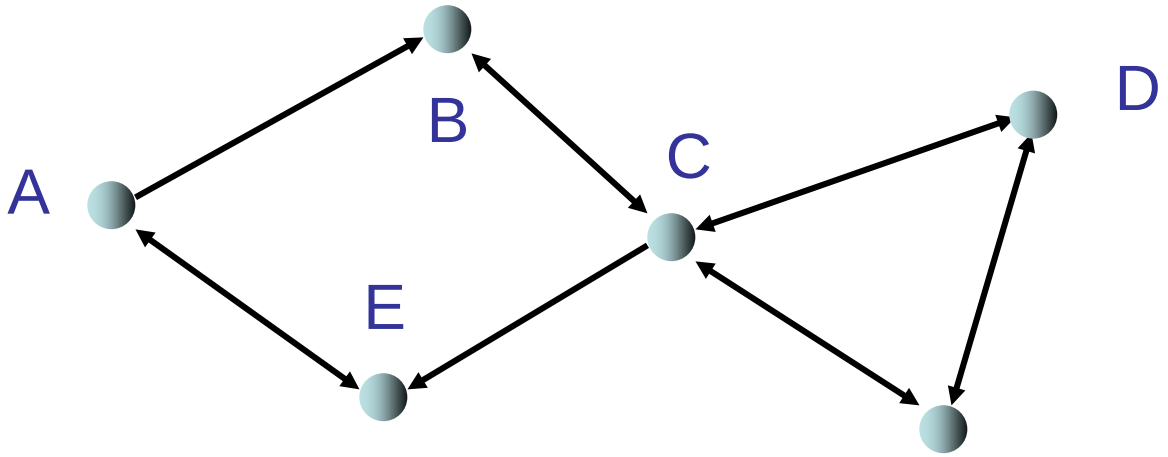
Beispiel: nicht zusammenhängender Graph



Graphentheorie

G gerichtet:

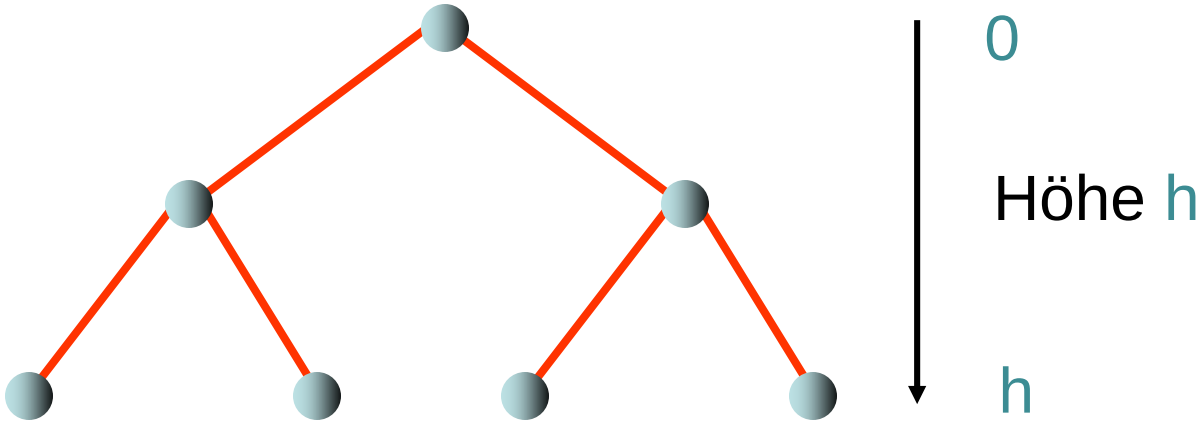
- G ist schwach zusammenhängend: Durchmesser D endlich, wenn alle Kanten als ungerichtet angesehen werden
- G ist stark zusammenhängend: D endlich



Lineare Liste:

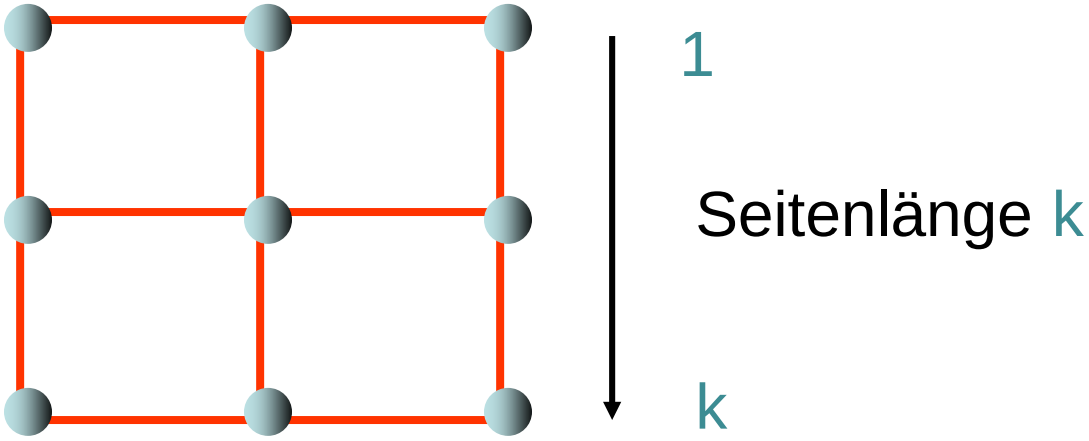
- Grad 2
- Hoher Durchmesser (n-1 für n Knoten)

Vollständiger binärer Baum:



- $n=2^{h+1}-1$ Knoten, Grad 3
- Durchmesser ist $2h \sim 2 \log_2 n$

2-dimensionales Gitter:

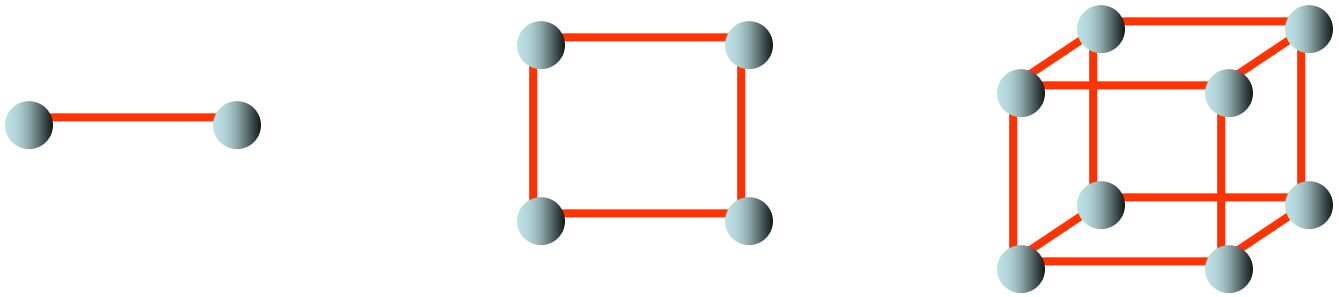


- $n = k^2$ Knoten, maximaler Grad 4
- Durchmesser ist $2(k-1) \sim 2\sqrt{n}$

Graphentheorie

Hypercube:

- Knoten: $(x_1,\ldots,x_d)\in\{0,1\}^d$
- Kanten: $\forall i: (x_1,\ldots,x_d)\rightarrow(x_1,..,x_{i-1},1\text{-}x_i,x_{i+1},..,x_d)$



d=1 d=2 d=3

Grad d, Durchmesser d

Operationen auf Graphen

G=(V,E): gerichteter Graph

Operationen:

- Insert(G,e): $E:=E\cup\{e\}$
- Remove(G,i,j): $E:=E\setminus\{e\}$ für die Kante $e=(v,w)$ mit key[v]=i und key[w]=j
- Insert(G,v): $V:=V\cup\{v\}$
- Remove(G,i): sei $v\in V$ der Knoten mit key[v]=i. $V:=V\setminus\{v\}$, $E:=E\setminus\{(x,y) \mid x=v \vee y=v\}$
- Search(G,i): gib Knoten v aus mit key[v]=i
- Search(G,i,j): gib Kante (v,w) aus mit key[v]=i und key[w]=j

Operationen auf Graphen

Anzahl der Knoten oft fest. In diesem Fall:

- $V=\{1,\ldots,n\}$ (Knoten fortlaufend numeriert, identifiziert durch ihre Keys)

Relevante Operationen:

- Insert(G,e): $E:=E\cup\{e\}$
- Remove(G,i,j): $E:=E\setminus\{e\}$ für die Kante $e=(i,j)$
- Search(G,i,j): gib Kante $e=(i,j)$ aus

Operationen auf Graphen

Menge und Schlüssel der Knoten variabel,

aber Anzahl nach oben begrenzt durch n:

- Hashing (z.B. Kuckuckshashing) kann verwendet werden, um Keys von n Knoten in Bereich $\{1,\ldots,O(n)\}$ 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.

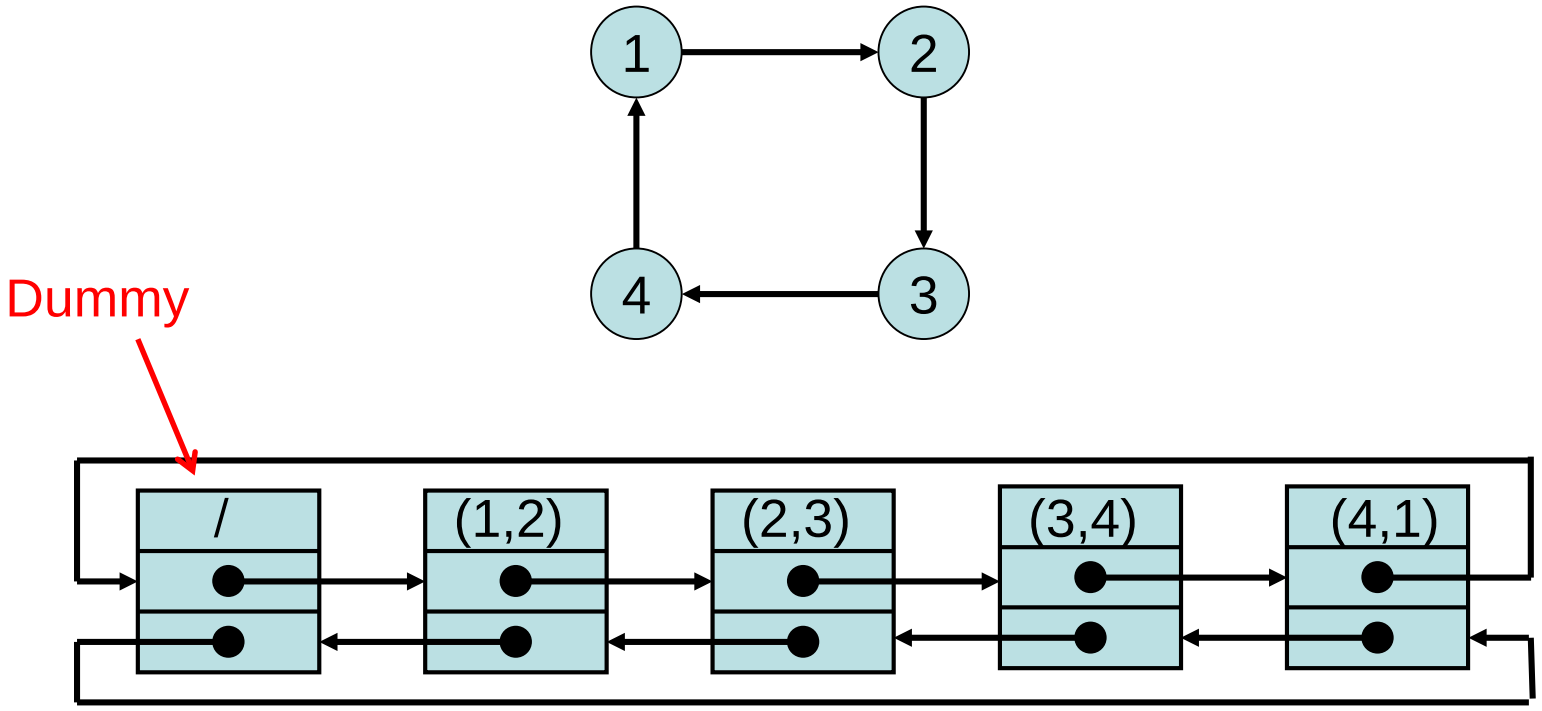
Parameter für Laufzeitanalyse:

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

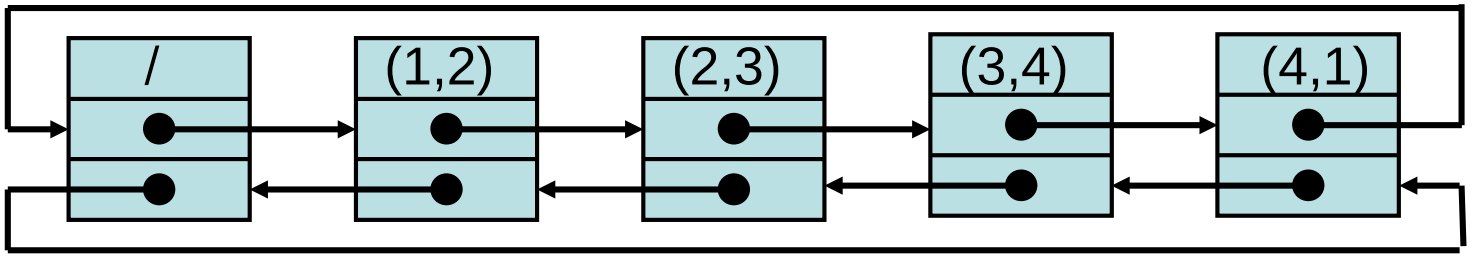
Graphrepräsentationen

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

1: Sequenz von Kanten



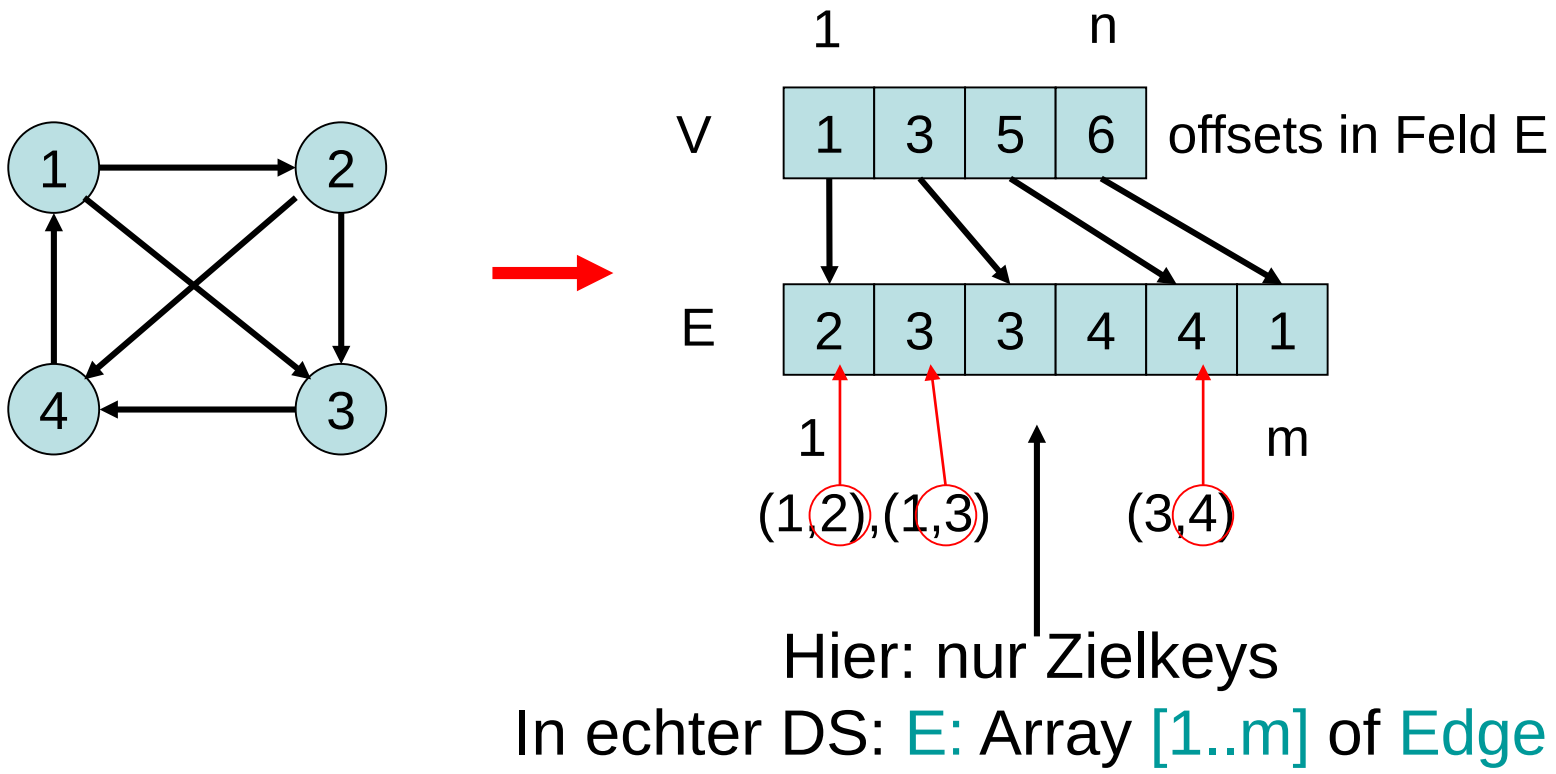
Graphrepräsentationen



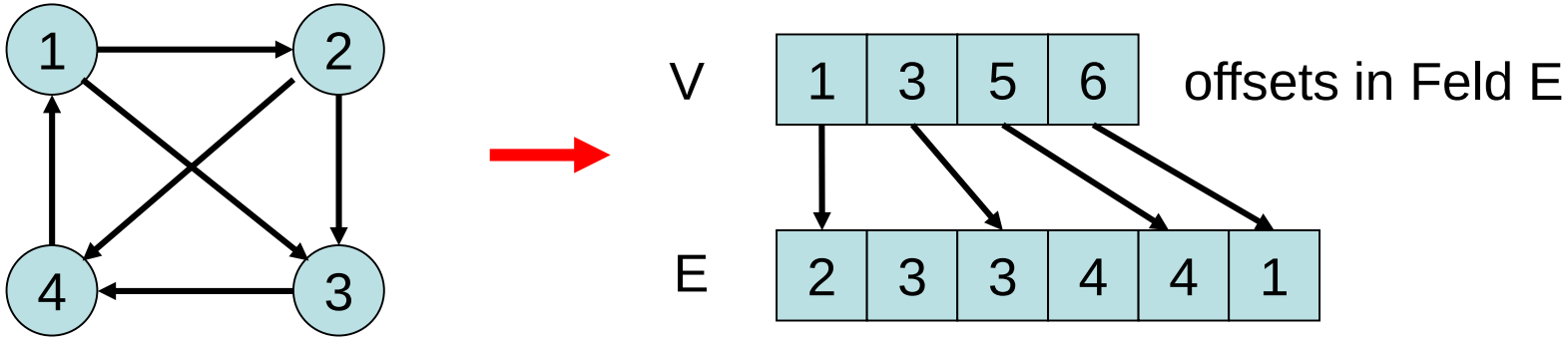
Zeitaufwand:

- Search(G,i,j): $\Theta(m)$ im worst case
- Insert(G,e): $O(1)$ (sofern e nicht in E)
- Remove(G,i,j): $\Theta(m)$ im worst case

2: Adjazenzfeld



Graphrepräsentationen

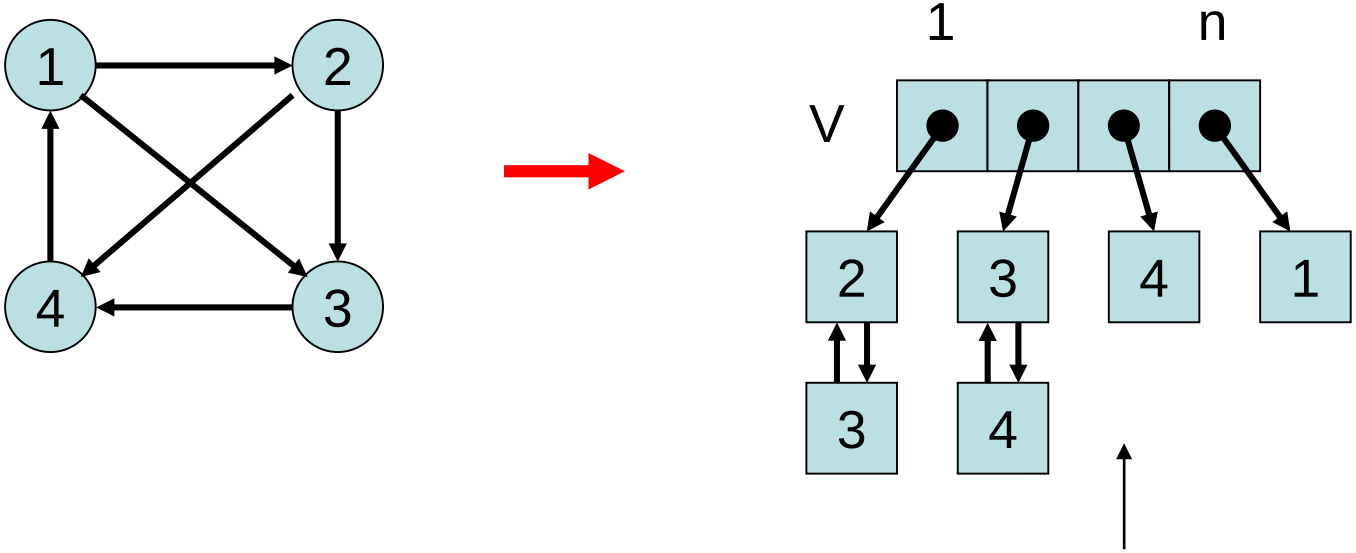


Zeitaufwand:

- Search(G,i,j): Zeit O(d)
- Insert(G,e): Zeit O(n+m) (worst case)
- Remove(G,i,j): Zeit O(n+m) (worst case)

wegen Verschiebungen in E

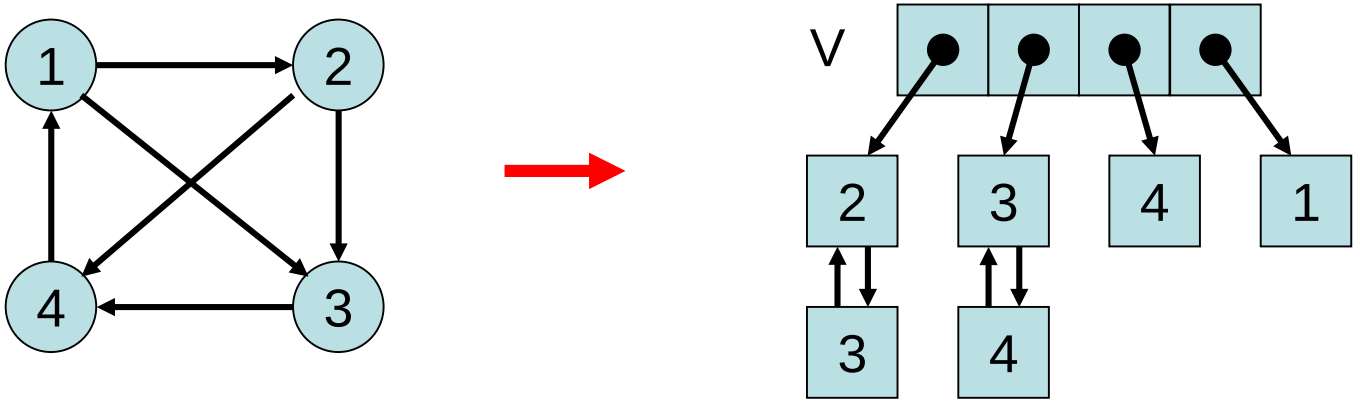
3: Adjazenzliste



Hier: nur Zielkeys

In echter DS: V: Array [1..n] of List of Edge

Graphrepräsentationen

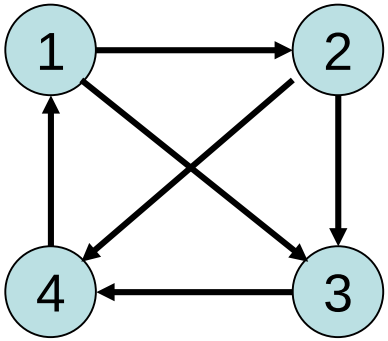


Zeitaufwand:

- Search(G,i,j): Zeit O(d)
- Insert(G,e): Zeit O(d)
- Remove(G,i,j): Zeit O(d)

Problem: d kann groß sein!

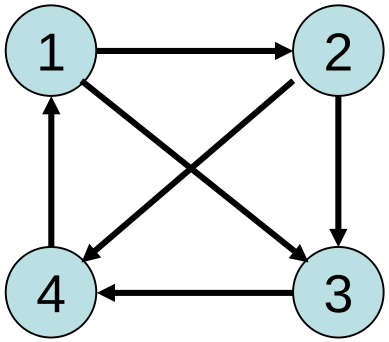
4: Adjazenzmatrix



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

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

Graphrepräsentationen



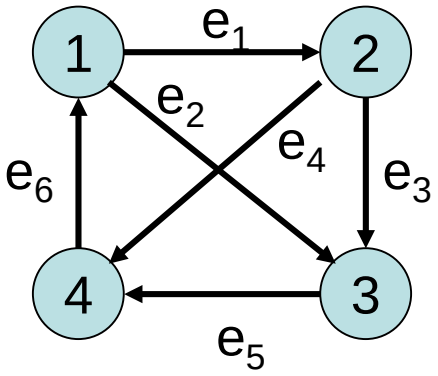
$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

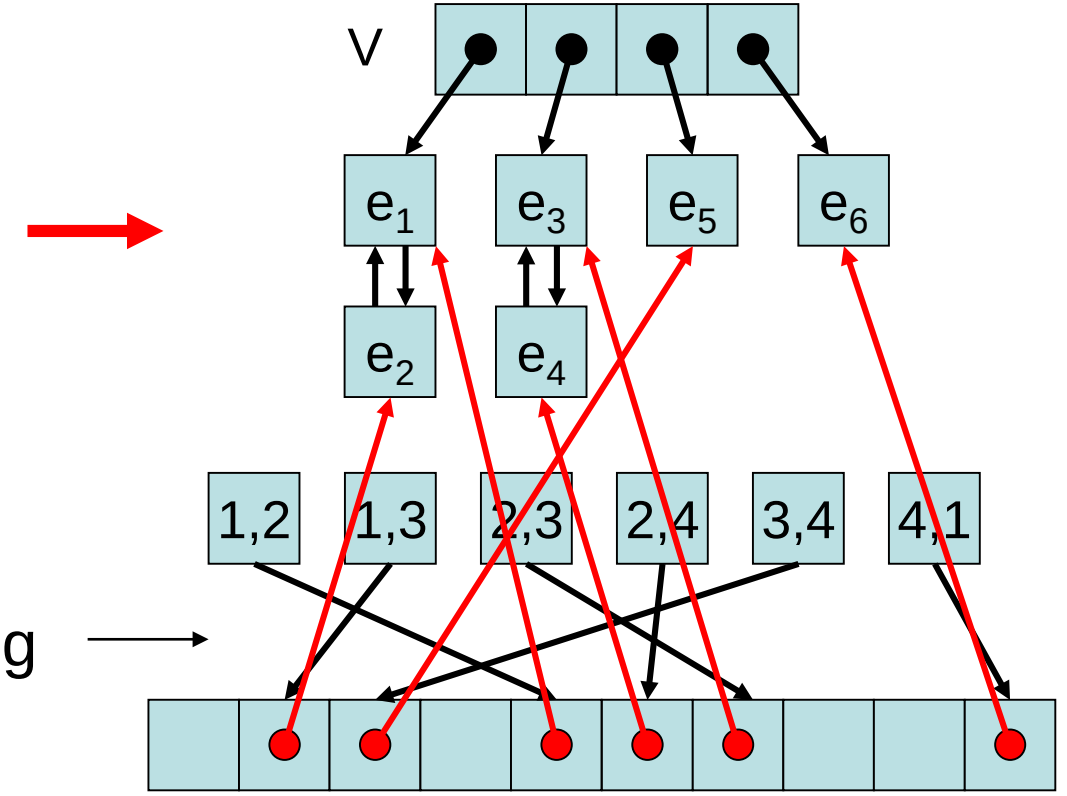
Zeitaufwand:

- Search(G,i,j): Zeit $O(1)$
- Insert(G,e): Zeit $O(1)$
- Remove(G,i,j): Zeit $O(1)$

Aber: Speicher-aufwand $\Theta(n^2)$

5: Adjazenzliste + Hashtabelle



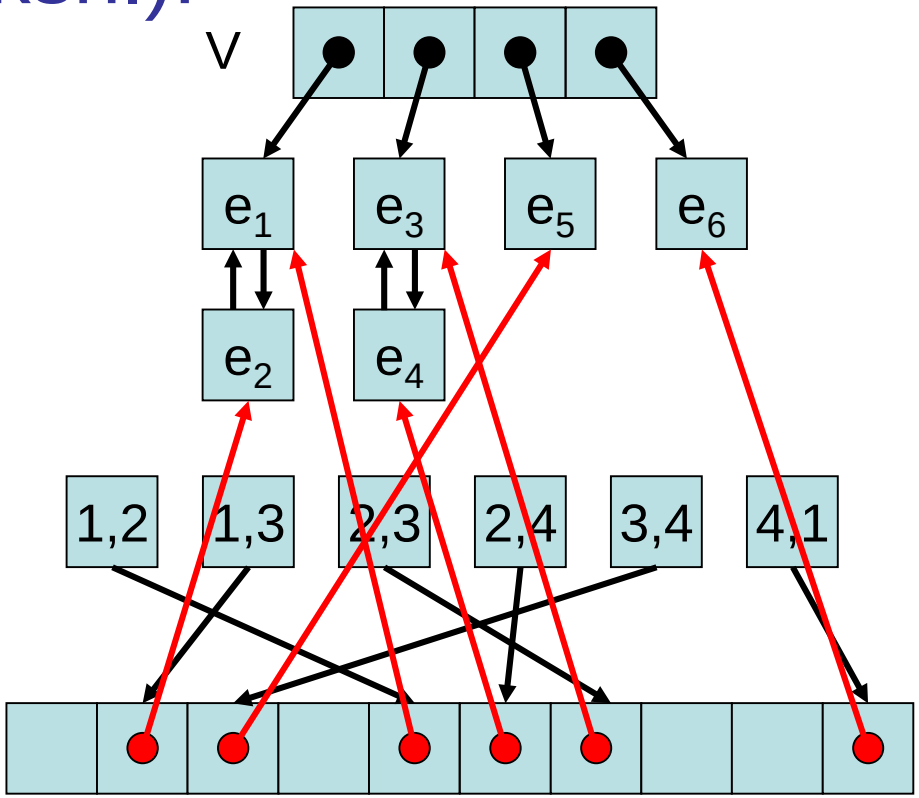


z.B. Kuckuckshashing

Graphrepräsentationen

Zeitaufwand (Kuckucksh.):

- Search(G,i,j):
  O(1) (worst case)
- Insert(G,e):
  O(1) (erwartet)
- Remove(G,i,j):
  O(1) (worst case)
- Speicher: O(n+m)

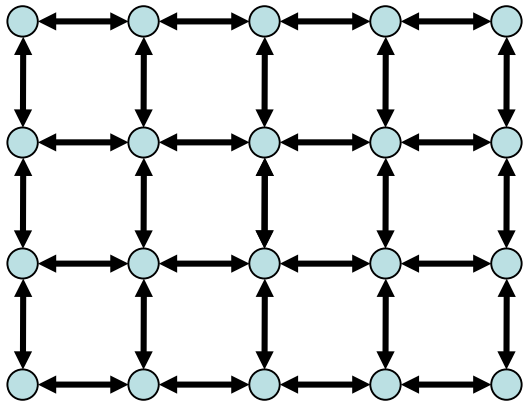


6: Implizite Repräsentationen

(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

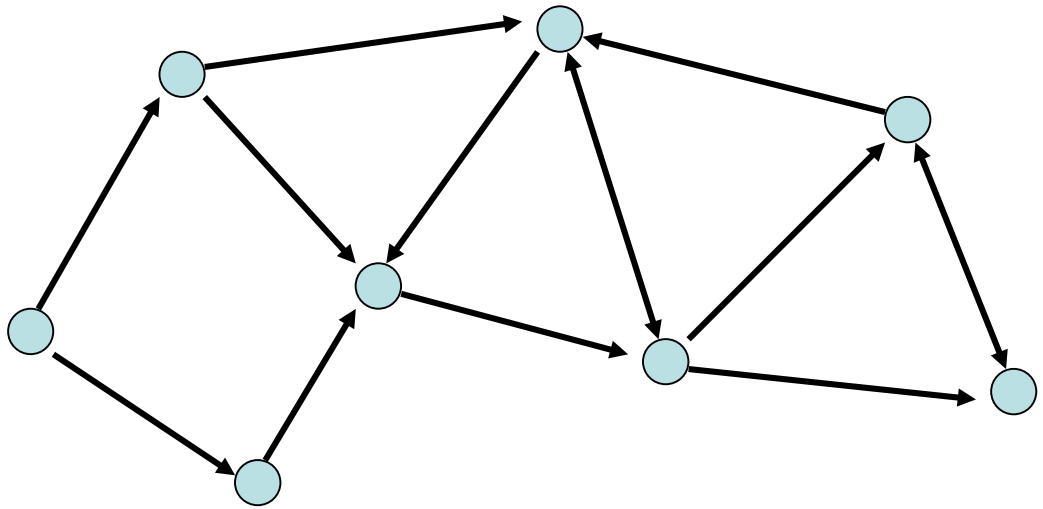
(k,l)-Gitter G=(V,E):

- $V=[k]\times[l]$ ($[a]=\{0,\ldots,a-1\}$ 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)
- Search-Operation: $O(1)$ Zeit (reine Rechnung)

Graphdurchlauf

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



Graphdurchlauf

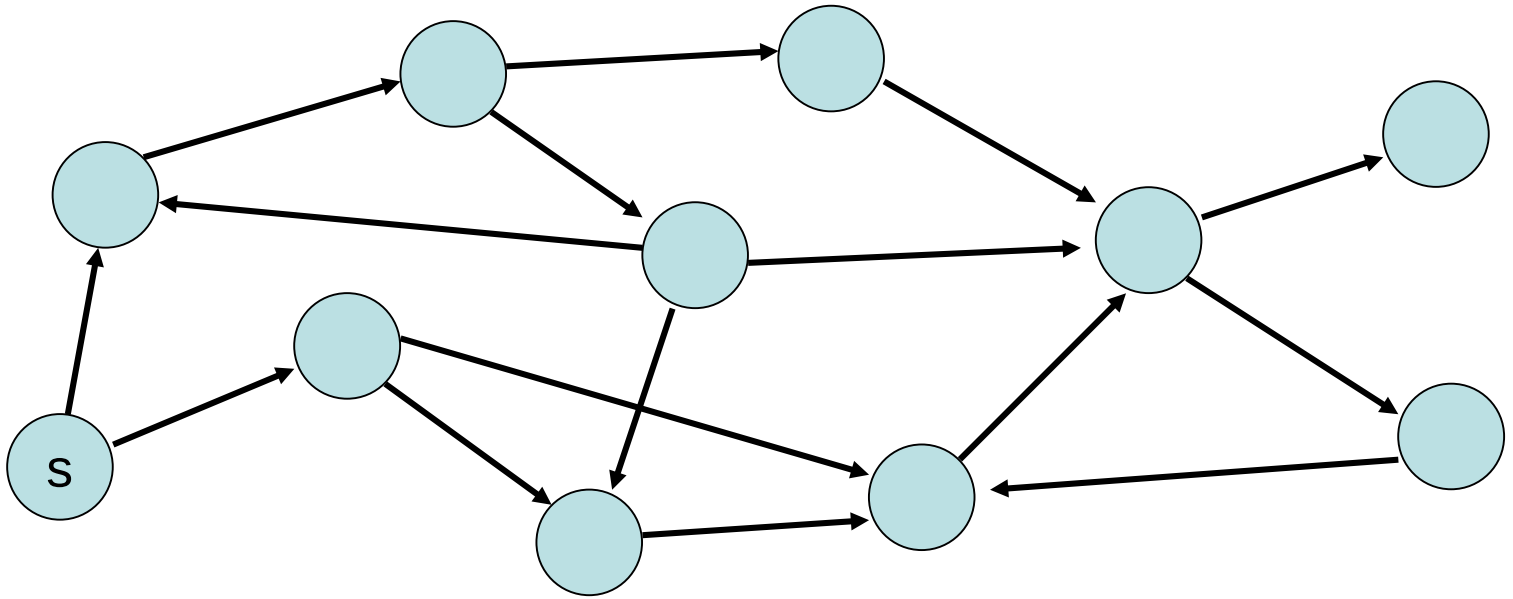
Zentrale Frage: Wie können wir die Knoten eines Graphen durchlaufen, so dass jeder Knoten mindestens einmal besucht wird?

Grundlegende Strategien:

- Breitensuche
- Tiefensuche

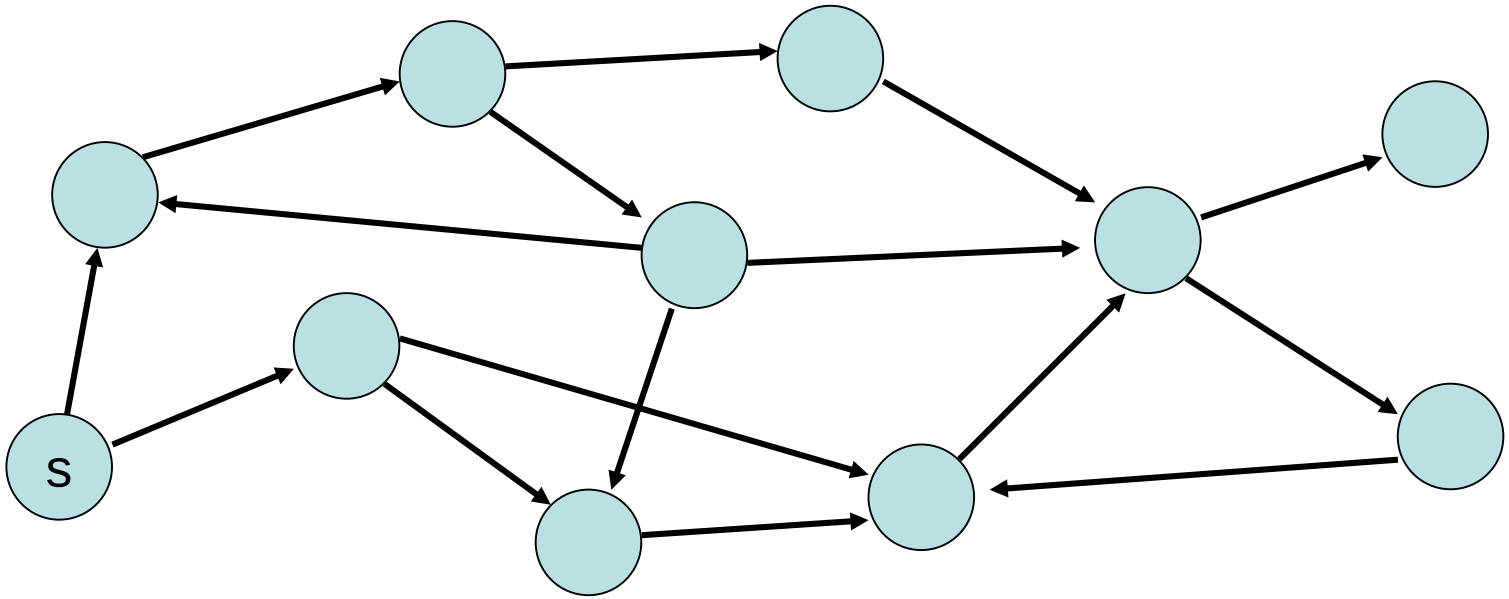
Breitensuche

- 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 – Überblick (1)

- Die Breitensuche ist ein Algorithmus, der die Grundlage vieler Graphenalgorithmen bildet.
- Ziel der Breitensuche ist es, bei einem Graphen $G=(V,E)$ und einer Quelle $s \in V$ alle Knoten $v \in V$ zu finden, die von s aus erreichbar sind. Dabei ist ein Knoten v von s aus erreichbar, wenn es in G einen (gerichteten) Pfad von s nach v gibt.
- Die Breitensuche berechnet auch für alle Knoten v die Distanz $\delta(s,v)$ von v zu s. Dabei ist die Distanz (oder der Abstand) von v zu s die minimale Anzahl Kanten auf einem (gerichteten) Pfad von s nach v.

Breitensuche - Überblick (2)

- Wird v entdeckt während Adj[u] nach neuen Knoten durchsucht wird, so heißt u Vorgänger von v.
  (Adj[u]: Menge der Nachbarn oder Adjazenzliste von u, d.h. der Knoten $v \in V$ mit $\{u,v\} \in E$ bzw. $(u,v) \in E$)
- Im Algorithmus: Knoten sind entweder weiß, grau oder schwarz.
- Weiß sind alle noch nicht entdeckten Knoten.
- Grau sind alle entdeckten Knoten, deren Adjazenzliste noch nicht vollständig nach neuen Knoten durchsucht wurde.
- Schwarz sind alle anderen Knoten, d.h. schwarze Knoten wurden bereits entdeckt und ihre Adjazenzliste wurde vollständig durchsucht.

Pseudocode für Breitensuche

```
BFS(G,s)
 1 for jeden Knoten u in V \ {s}
 2     do color[u] ← WHITE
 3        d[u] ← ∞
 4        π[u] ← NIL
 5 color[s] ← GRAY
 6 d[s] ← 0
 7 π[s] ← NIL
 8 Q ← { }
 9 Enqueue(Q,s)
10 while Q ≠ { }
11       do u ← Dequeue(Q)
12          for v ∈ Adj[u]
13              do if color[v] = WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u] + 1
16                         π[v] ← u
17                         Enqueue(Q,v)
18          color[u] ← BLACK
```

- BFS benutzt Queue für graue Knoten.
- color[u] := Feld für Farbe von v. Initial WHITE.
- d[u] := Feld für bislang berechneten Abstand zu s. Initial ∞.
- π[u] := Feld für Vorgänger. Initial NIL.

Pseudocode für Breitensuche

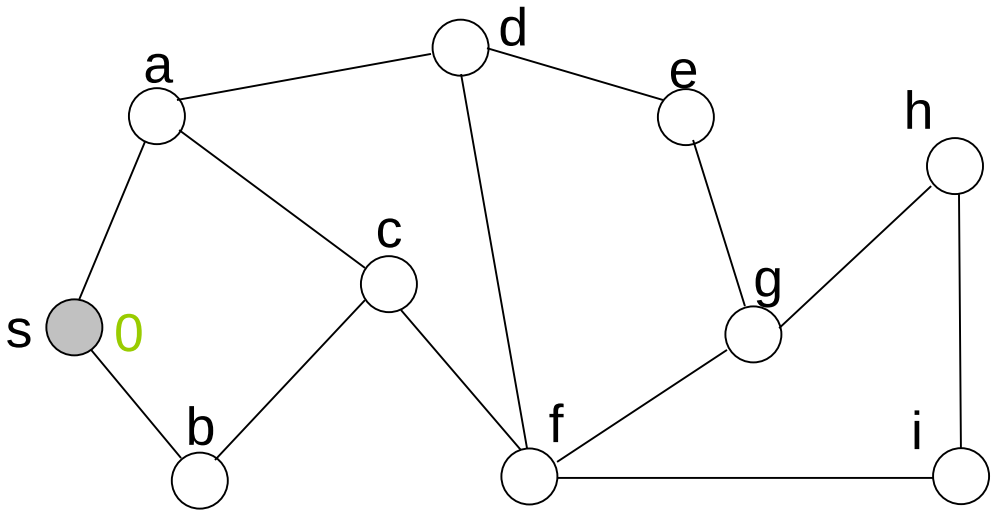
```
BFS(G,s)
 1 for jeden Knoten u in V \ {s}
 2     do color[u] ← WHITE
 3        d[u] ← ∞
 4        π[u] ← NIL
 5 color[s] ← GRAY
 6 d[s] ← 0
 7 π[s] ← NIL
 8 Q ← { }
 9 Enqueue(Q,s)
10 while Q ≠ { }
11       do u ← Dequeue(Q)
12          for v ∈ Adj[u]
13              do if color[v] = WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u] + 1
16                         π[v] ← u
17                         Enqueue(Q,v)
18          color[u] ← BLACK
```

- BFS benutzt Queue für graue Knoten.
- color[u] := Feld für Farbe von v. Initial WHITE.
- d[u] := Feld für bislang berechneten Abstand zu s. Initial ∞.
- π[u] := Feld für Vorgänger. Initial NIL.

initialisiere BFS

Breitensuche - Beispiel

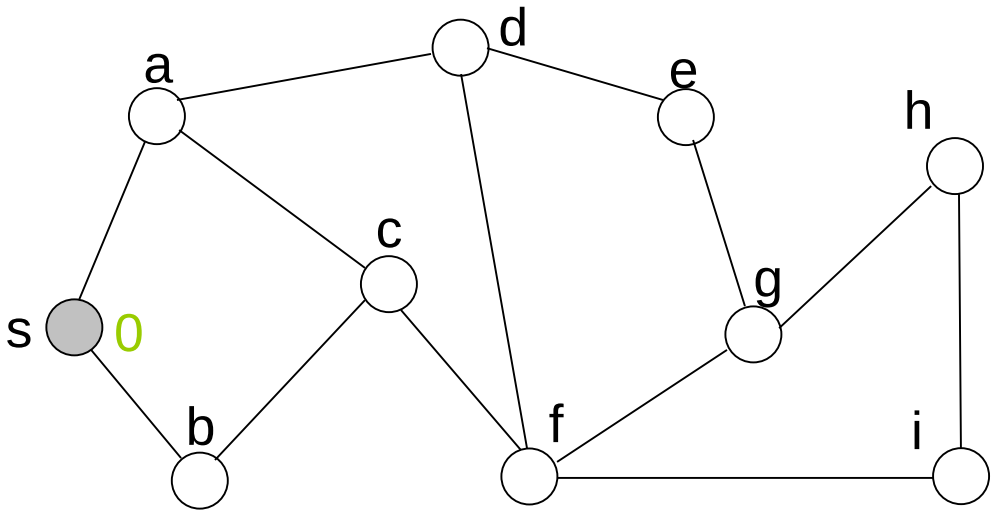
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: s

Breitensuche - Beispiel

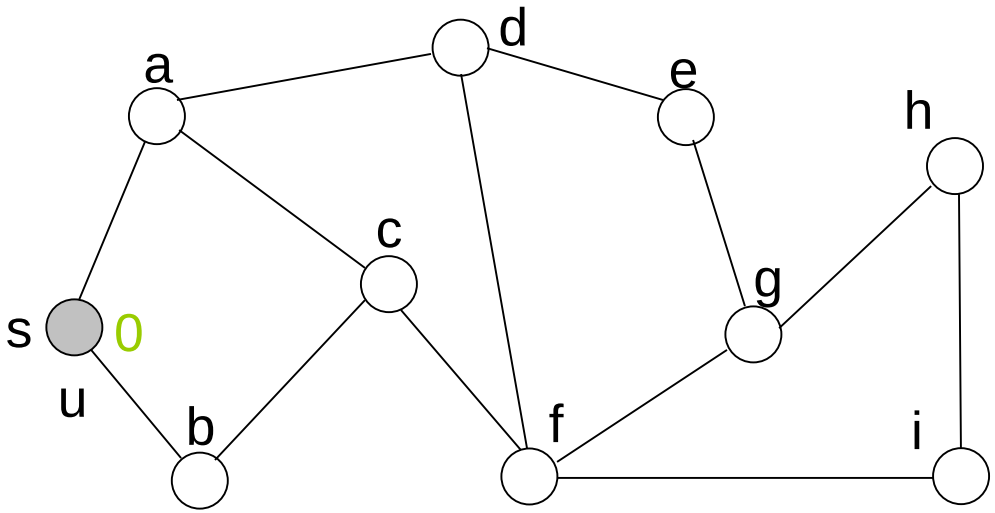
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: s

Breitensuche - Beispiel

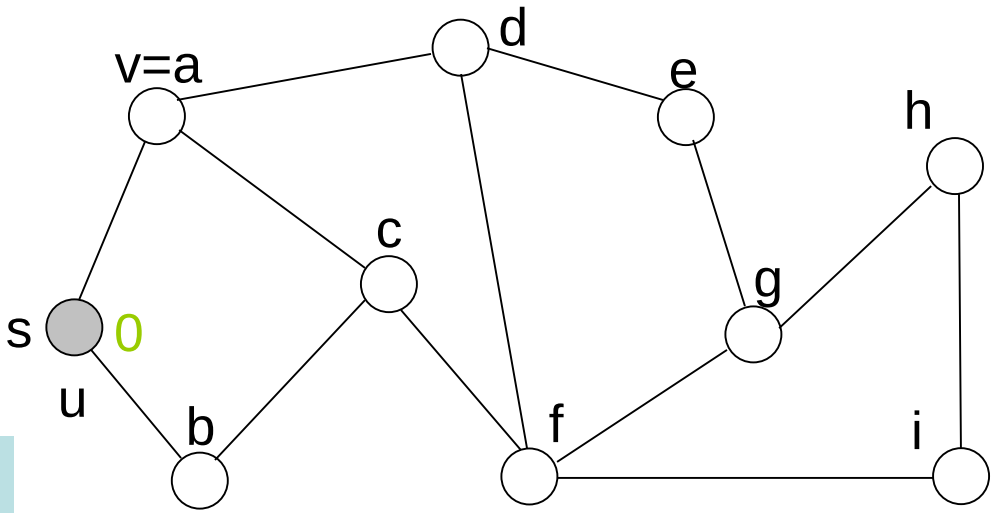
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche - Beispiel

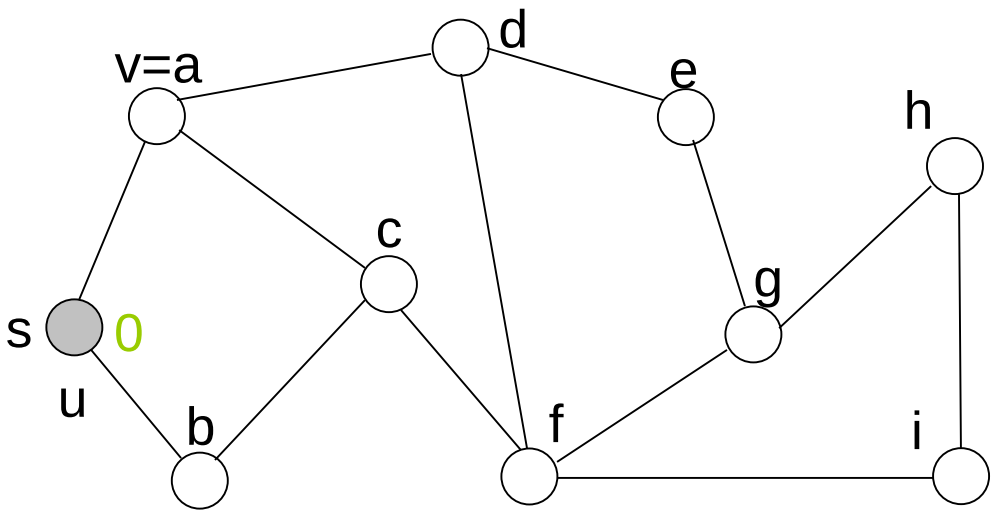
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche - Beispiel

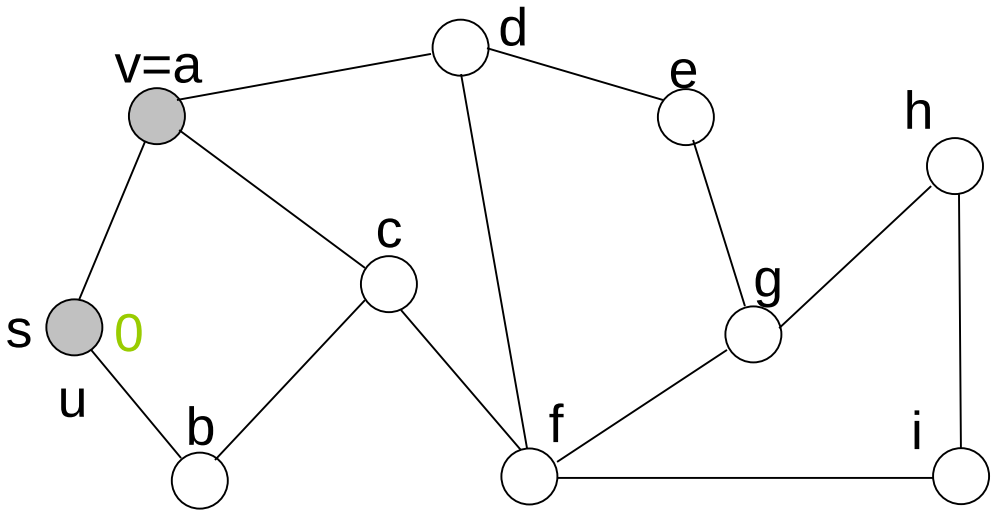
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche - Beispiel

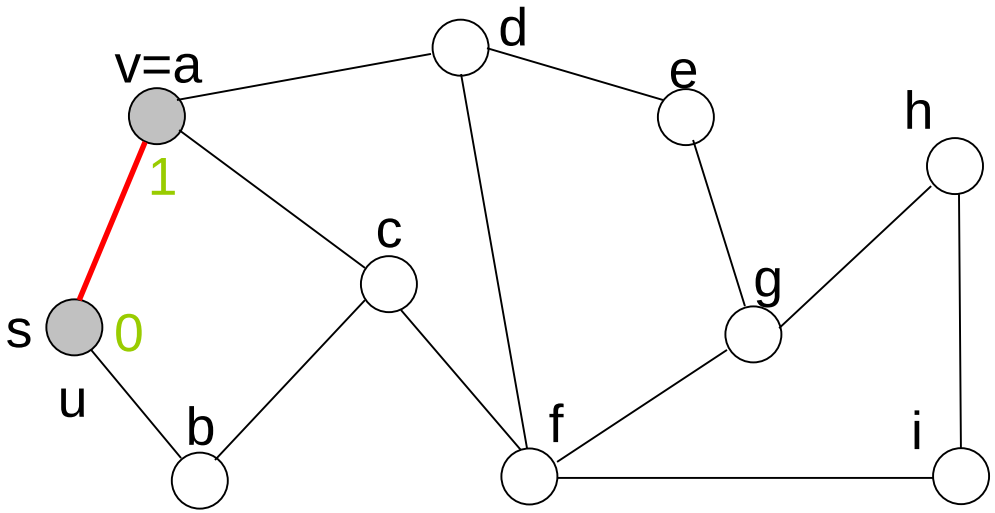
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche - Beispiel

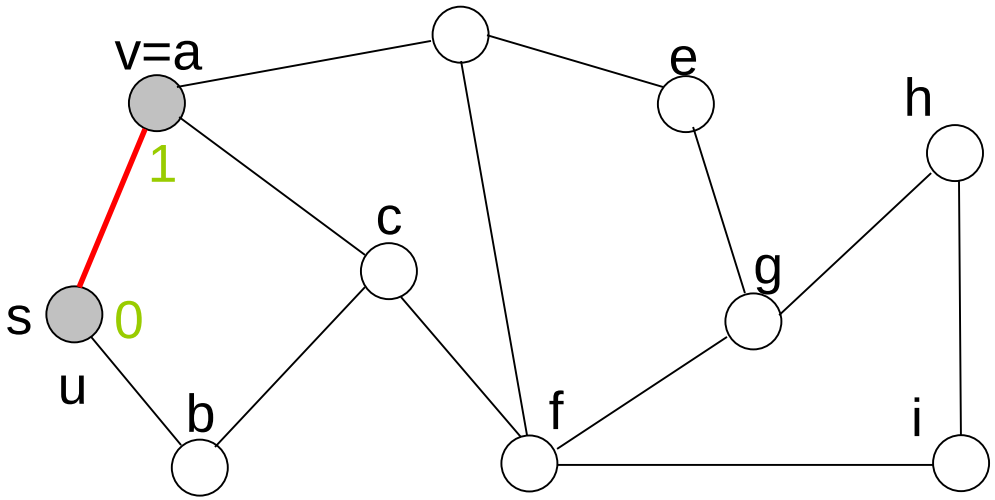
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche - Beispiel

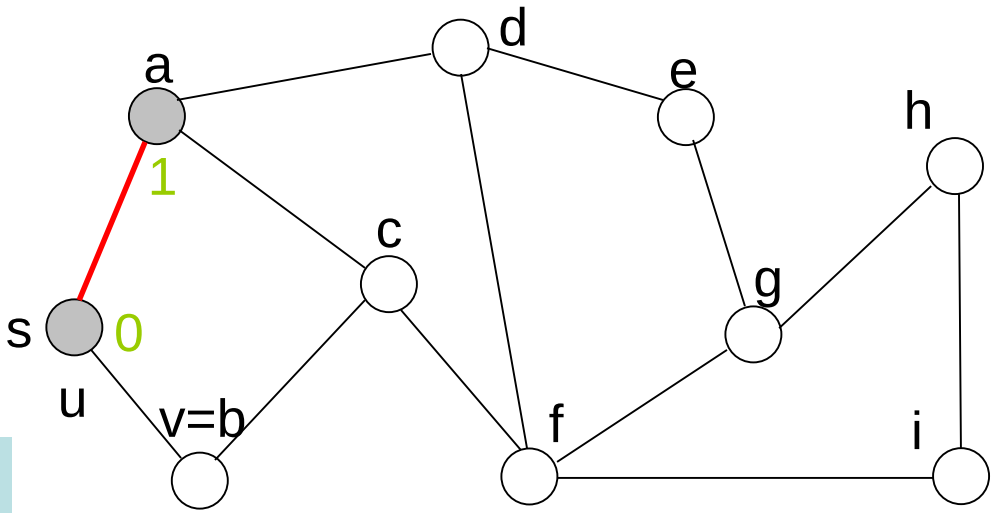
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breitensuche - Beispiel

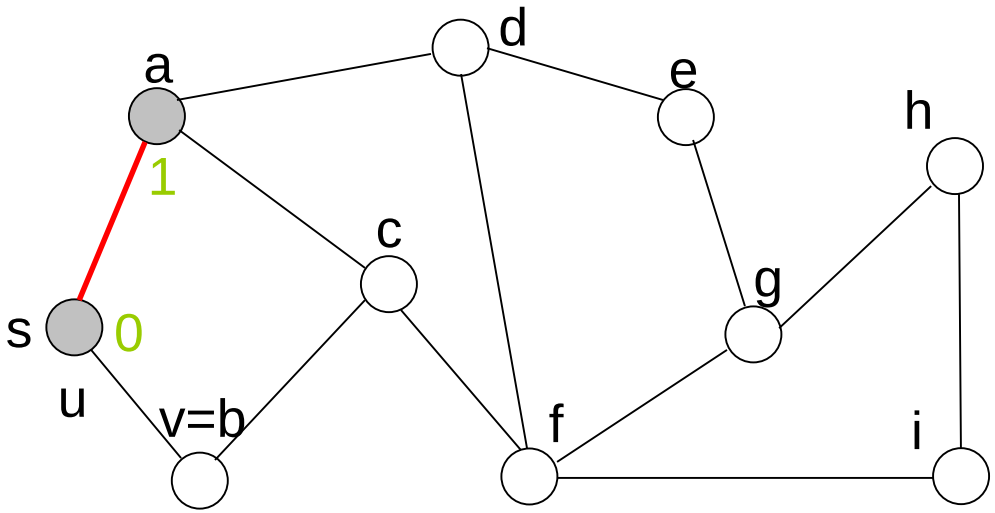
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breitensuche - Beispiel

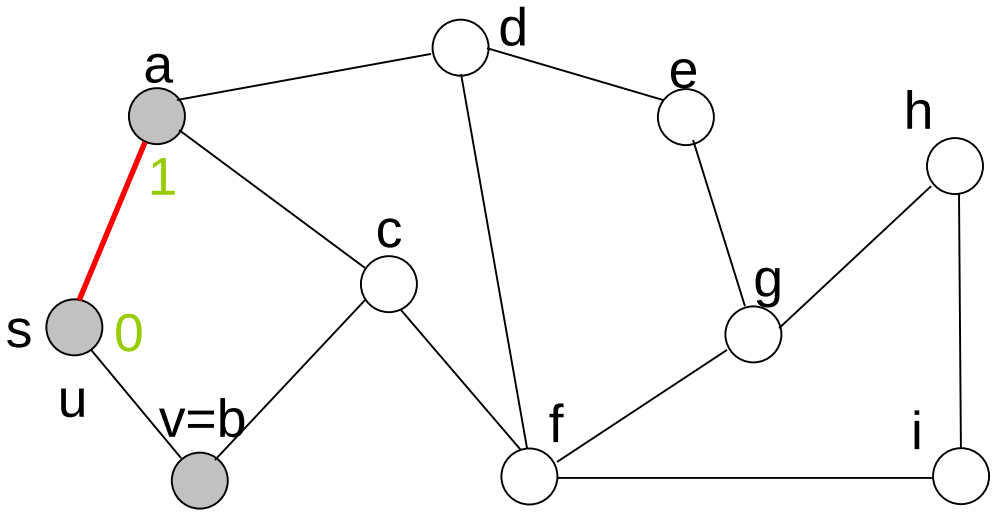
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breitensuche - Beispiel

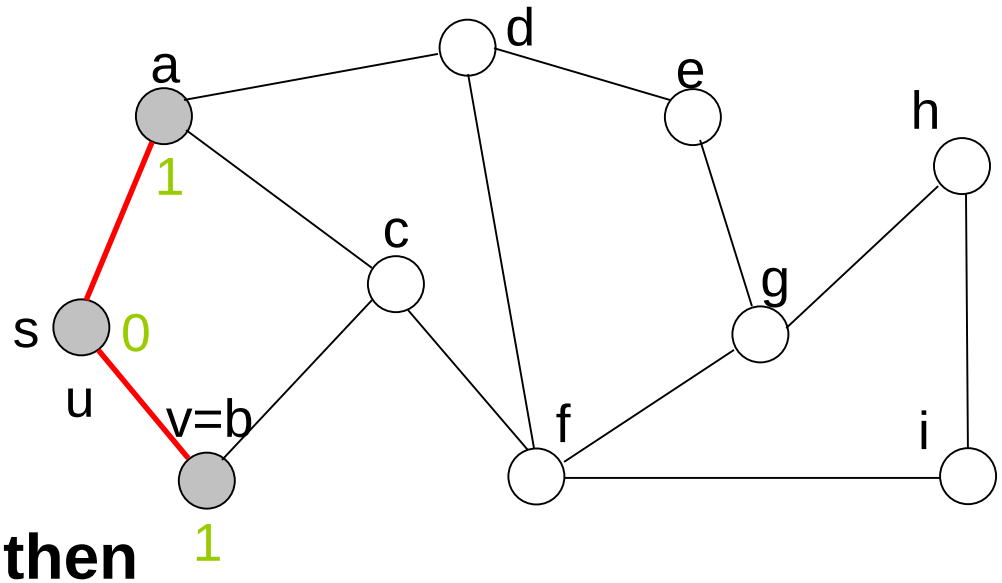
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breitensuche - Beispiel

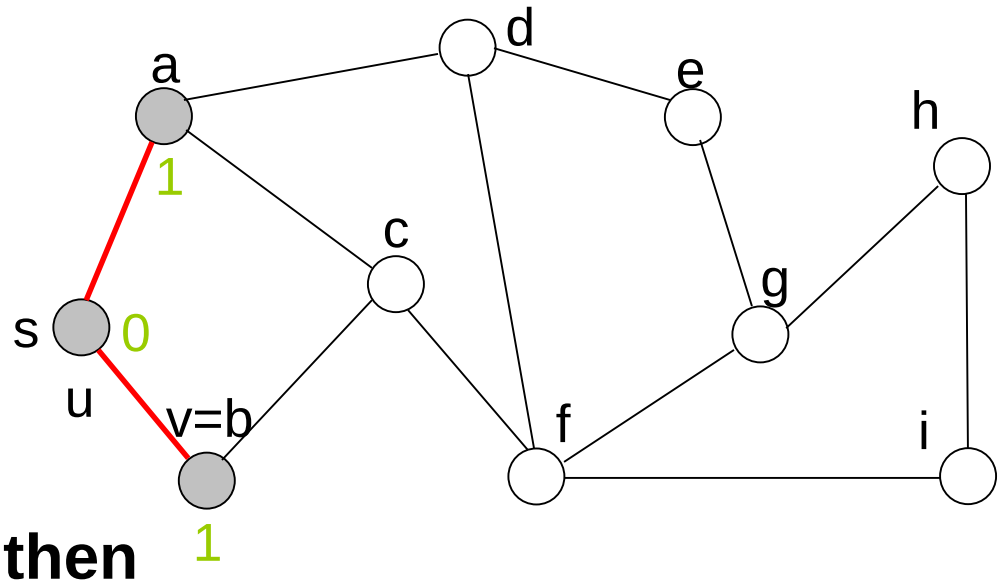
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a

Breitensuche - Beispiel

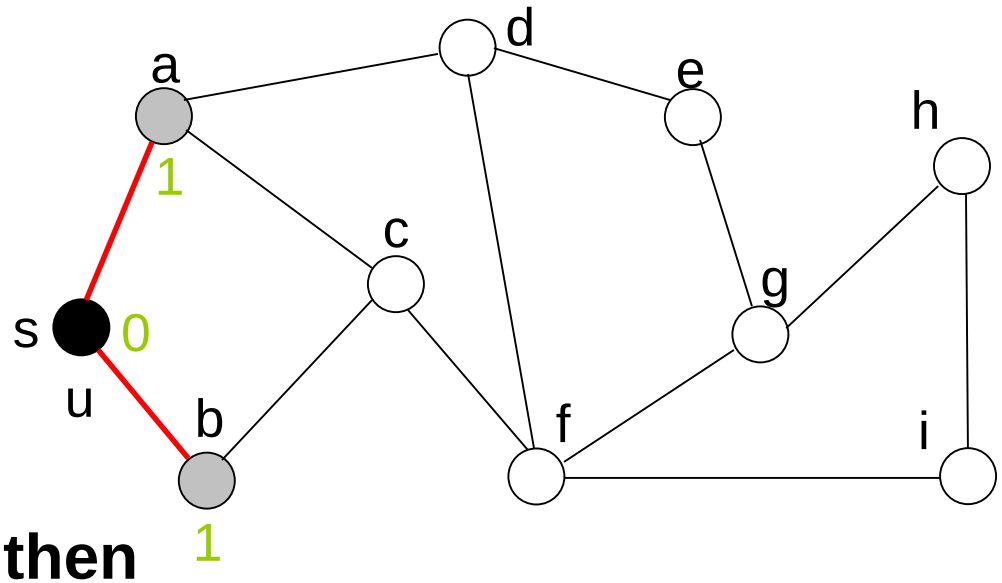
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a, b

Breitensuche - Beispiel

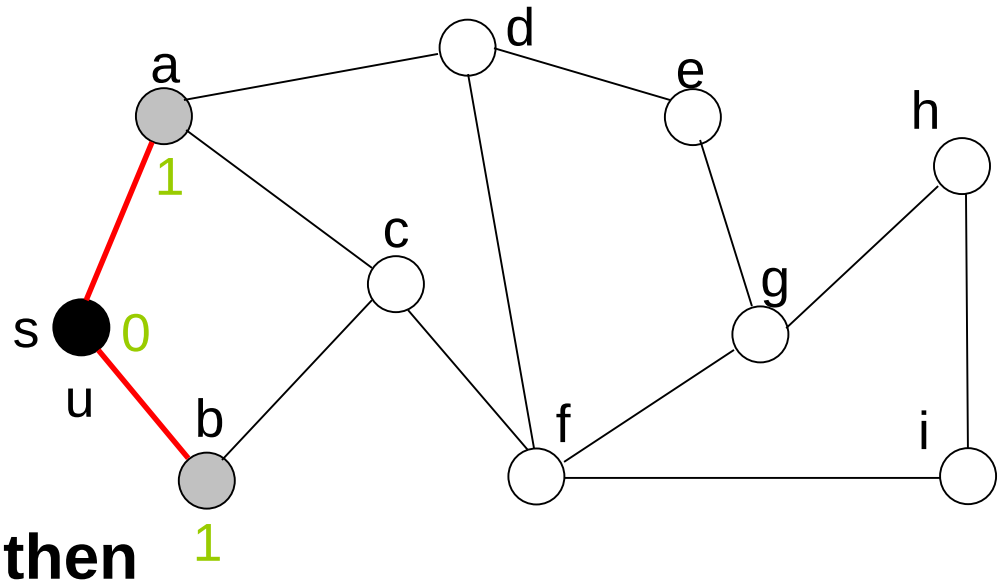
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a, b

Breitensuche - Beispiel

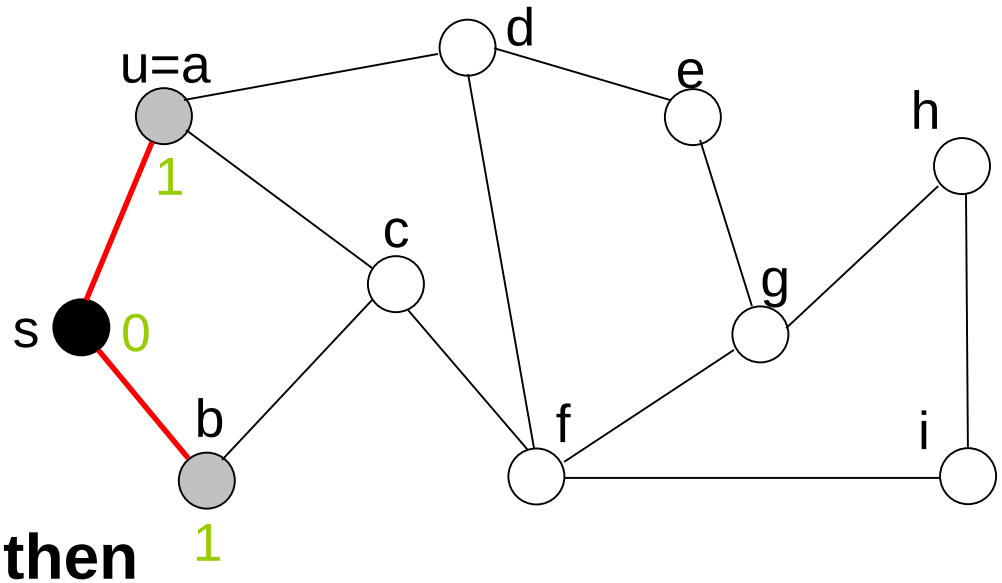
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: a, b

Breitensuche - Beispiel

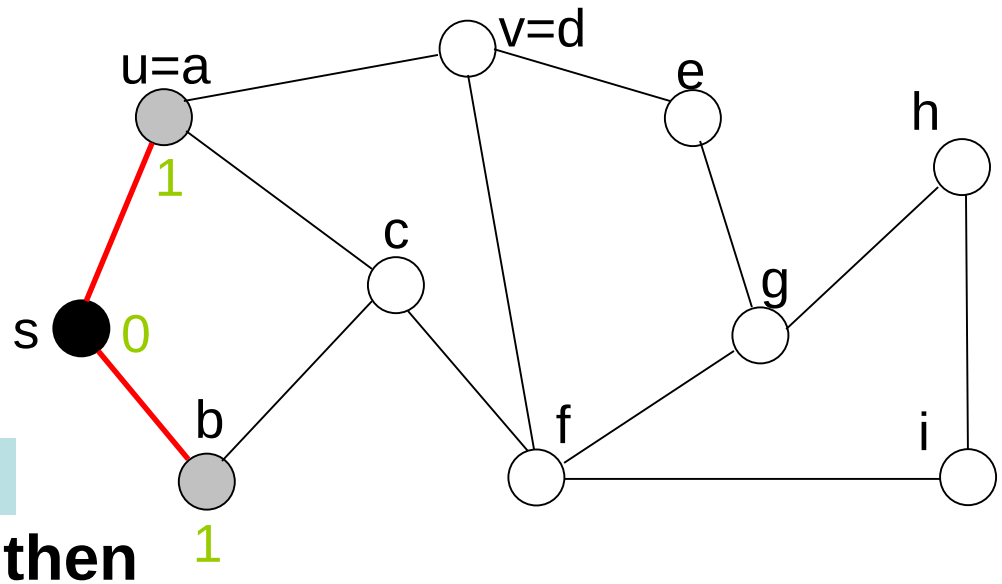
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breitensuche - Beispiel

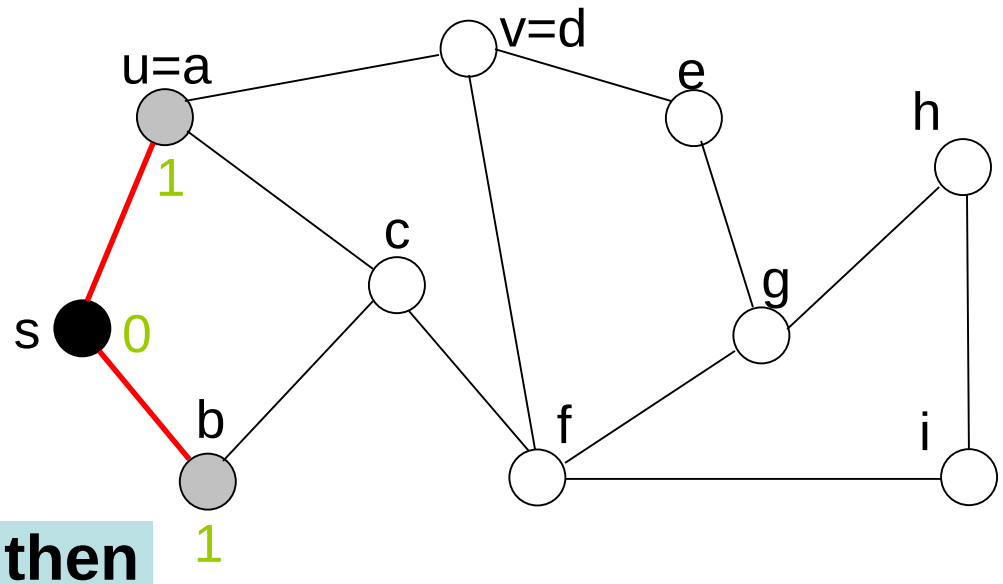
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breitensuche - Beispiel

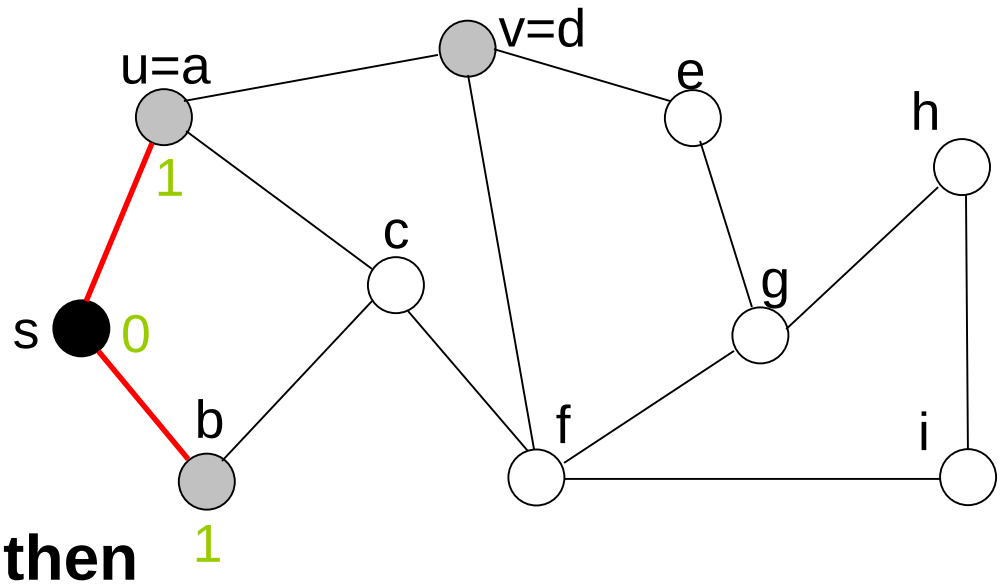
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breitensuche - Beispiel

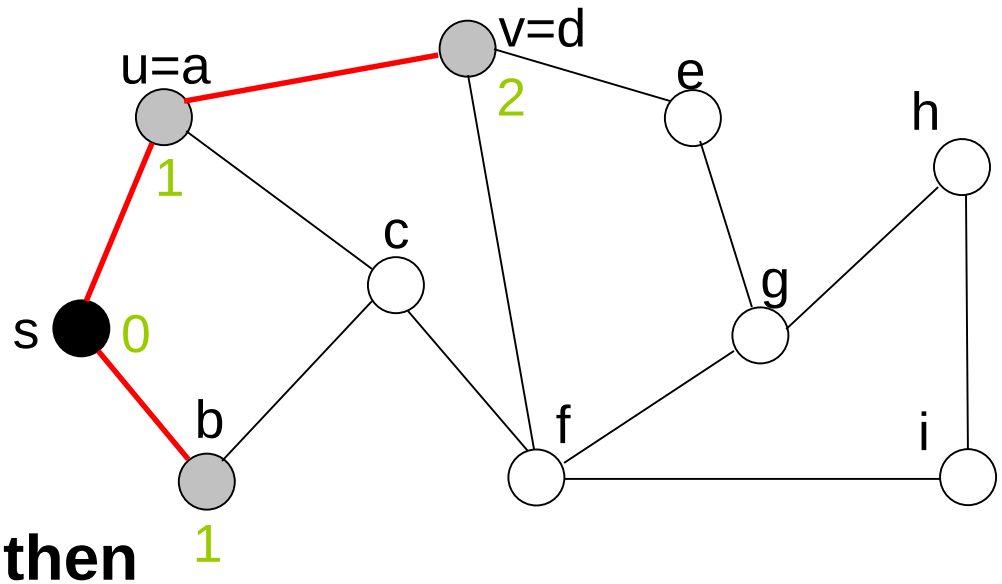
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breitensuche - Beispiel

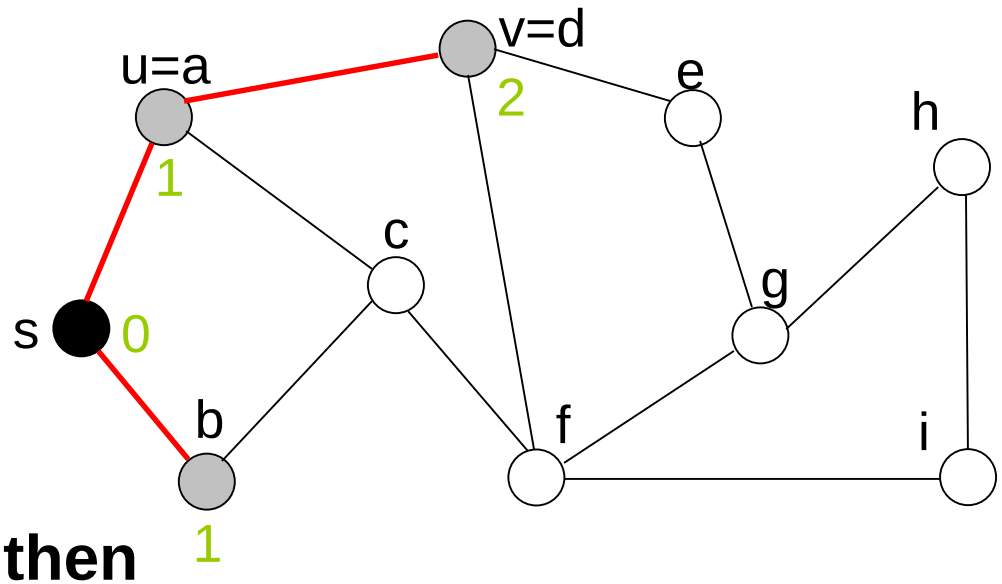
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b

Breitensuche - Beispiel

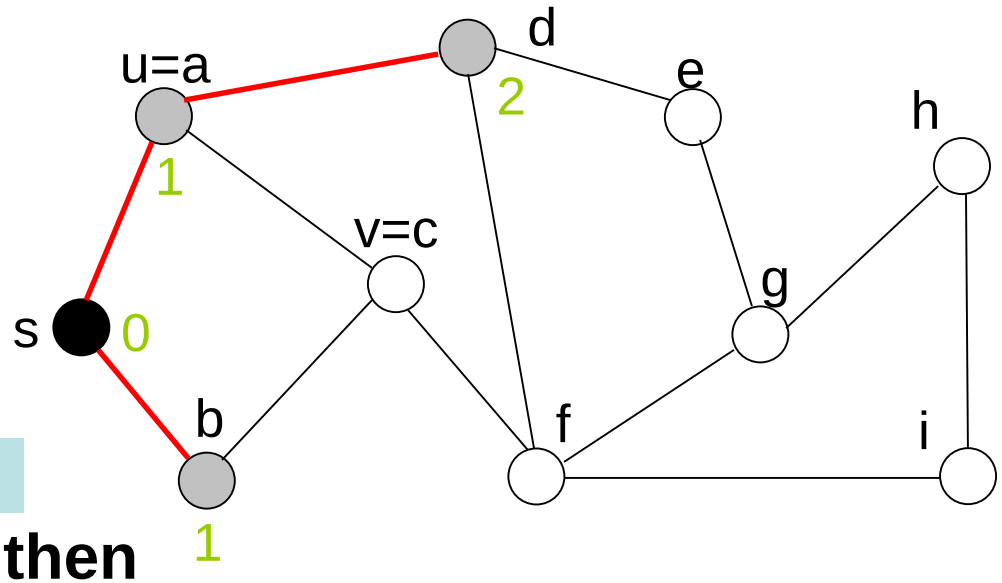
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breitensuche - Beispiel

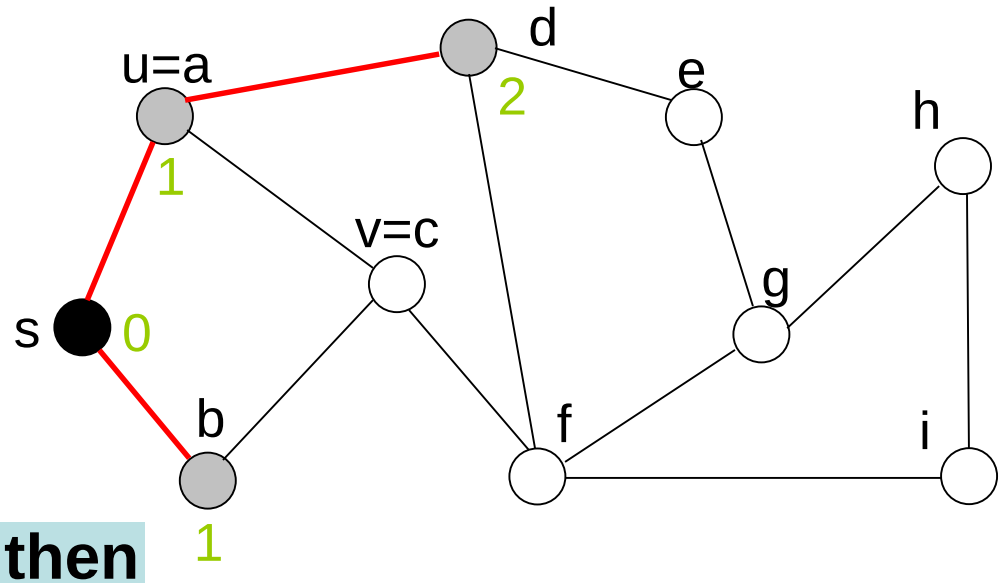
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breitensuche - Beispiel

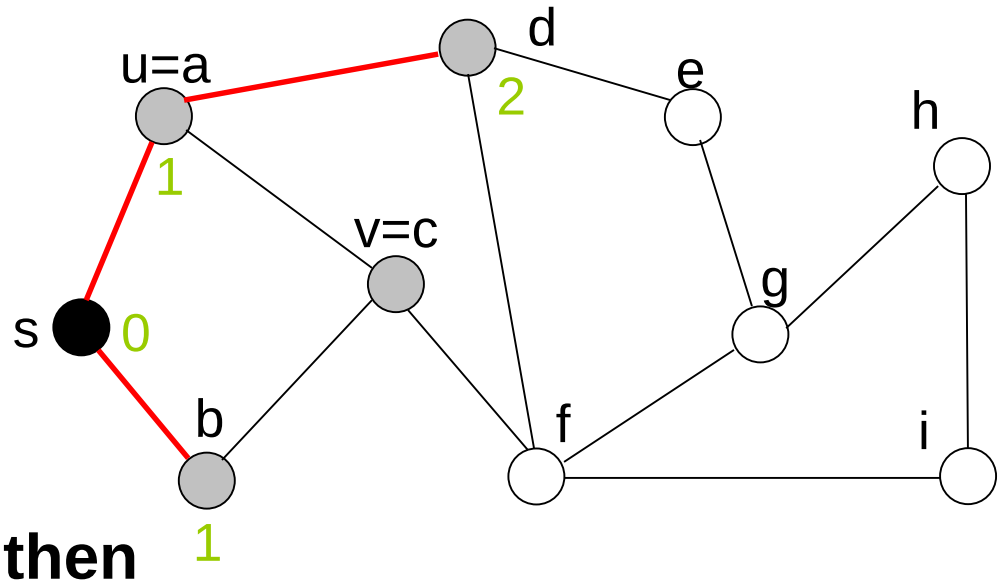
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breitensuche - Beispiel

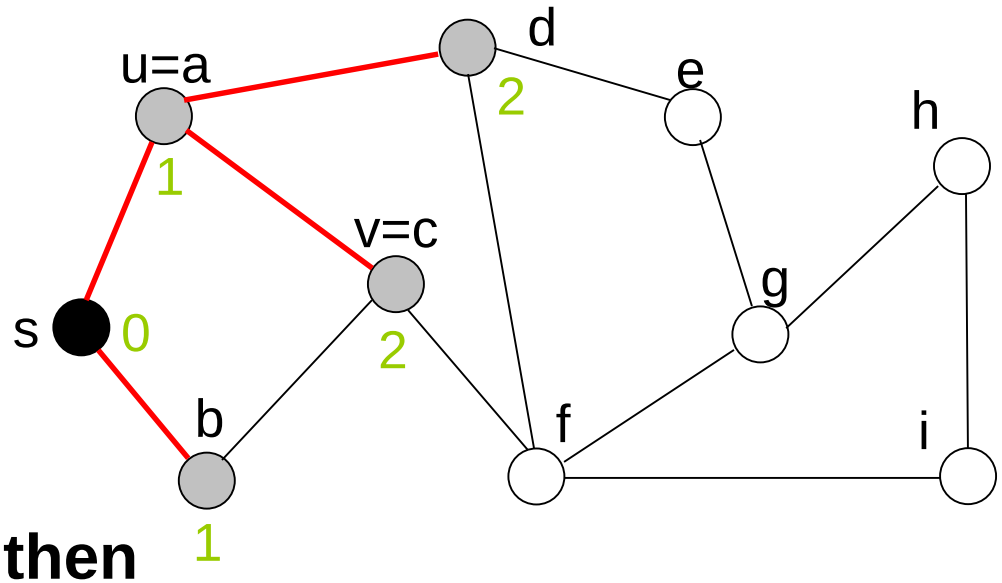
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breitensuche - Beispiel

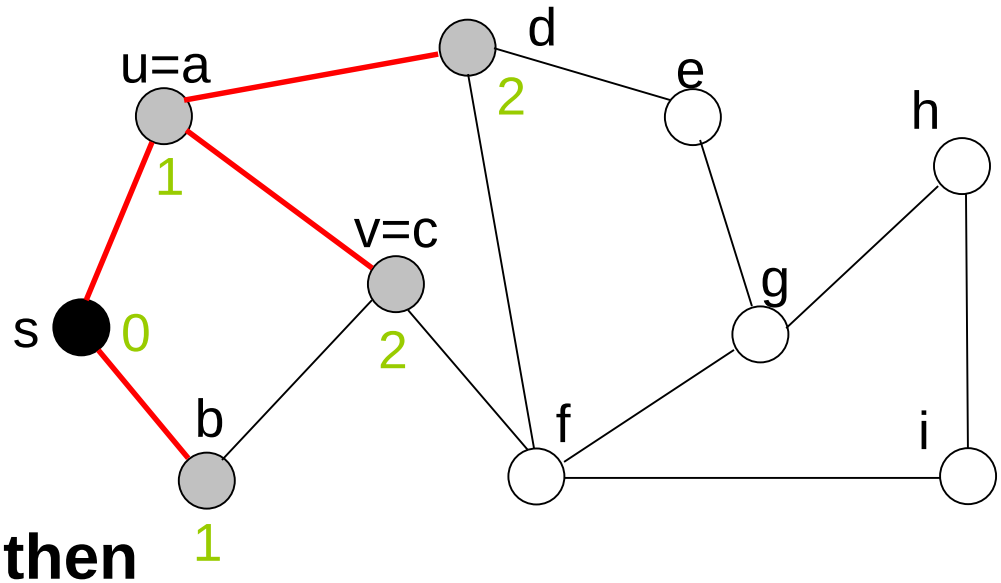
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d

Breitensuche - Beispiel

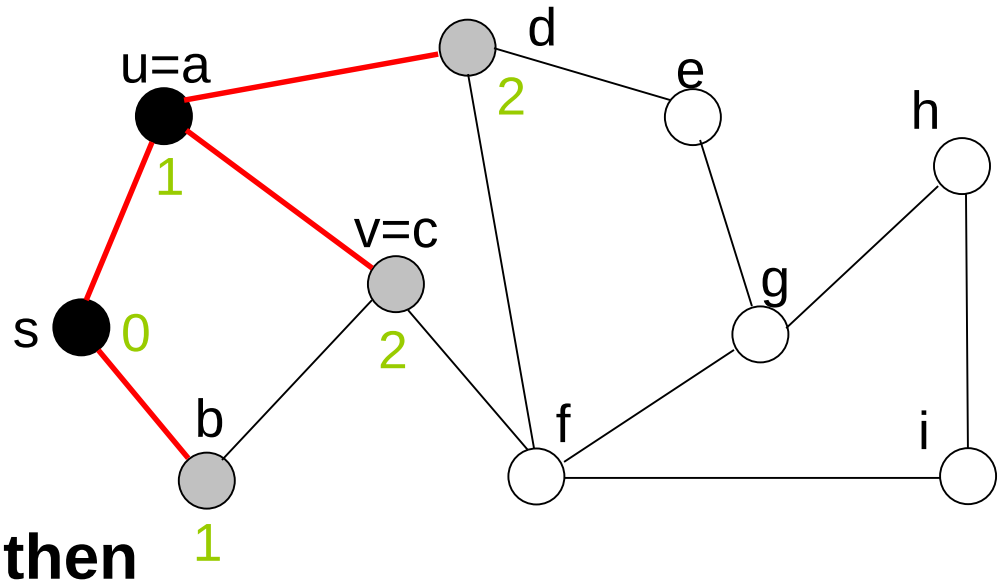
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d, c

Breitensuche - Beispiel

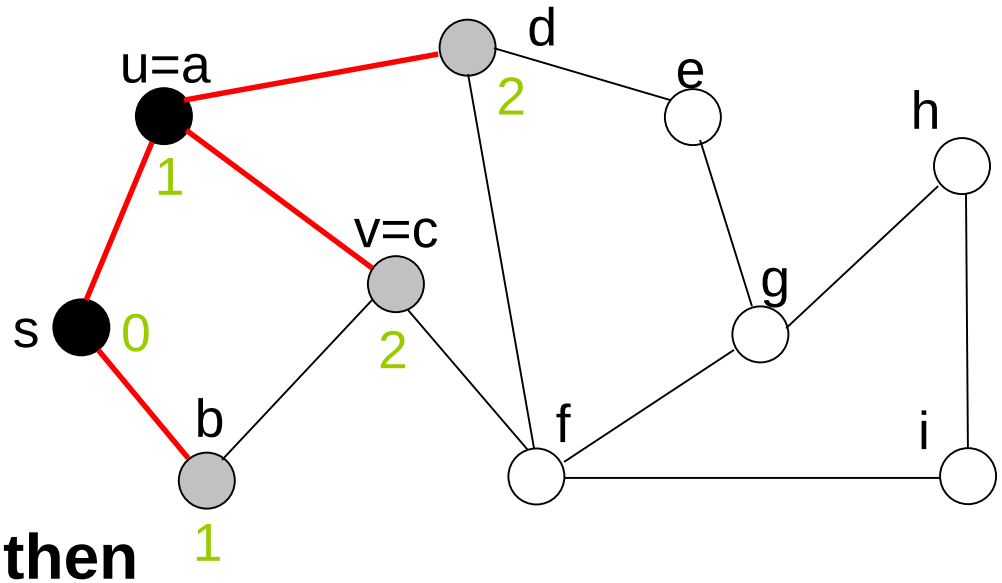
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d, c

Breitensuche - Beispiel

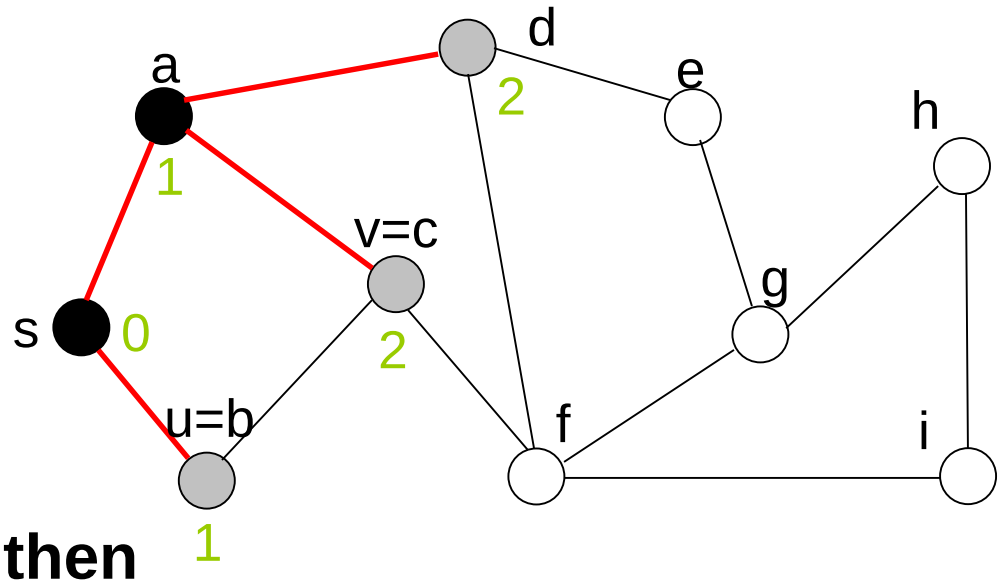
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: b, d, c

Breitensuche - Beispiel

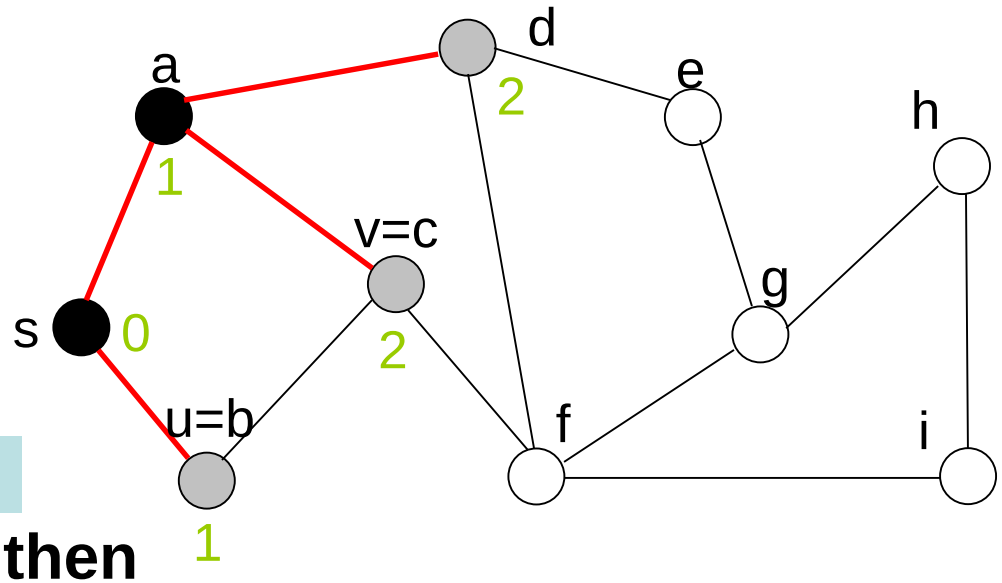
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breitensuche - Beispiel

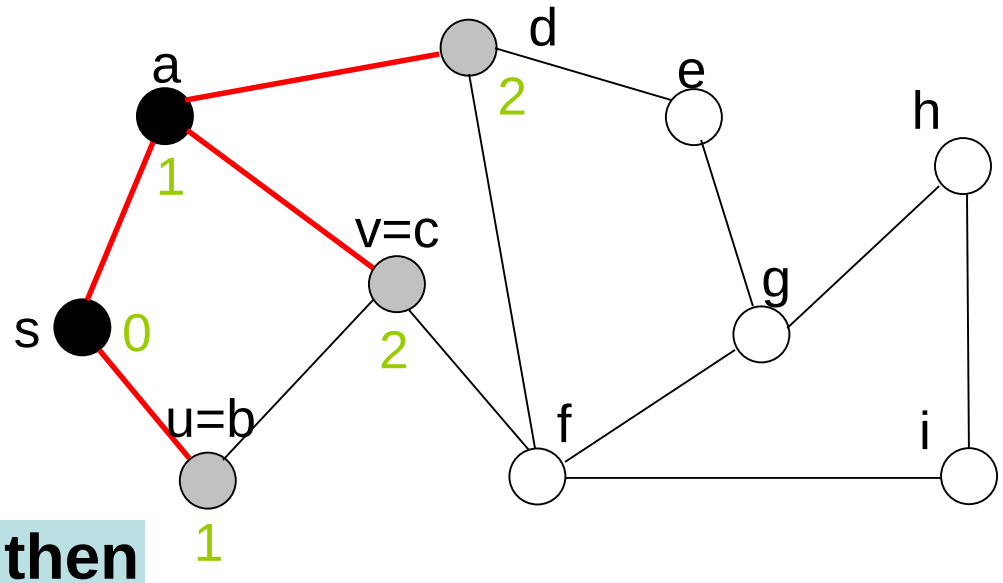
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breitensuche - Beispiel

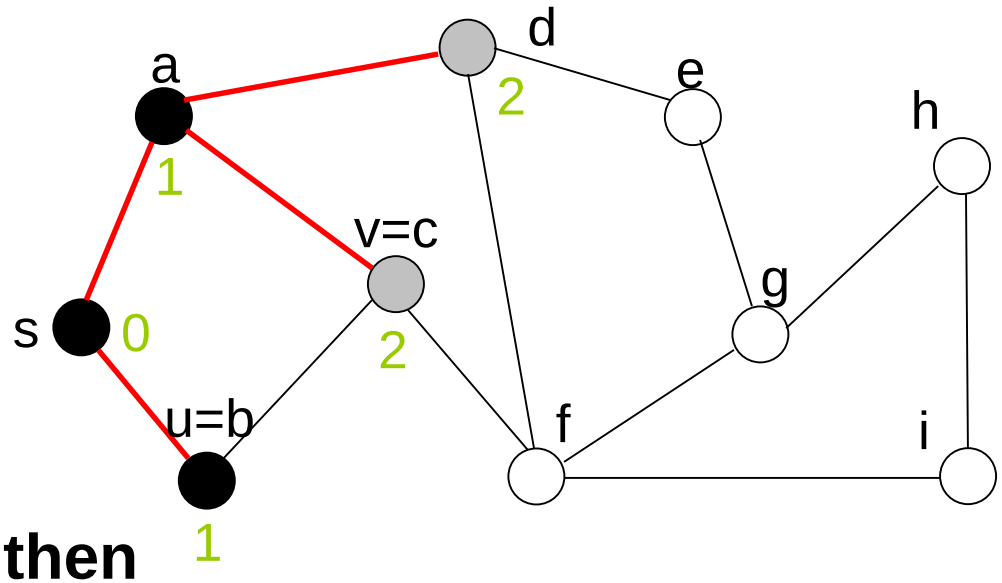
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breitensuche - Beispiel

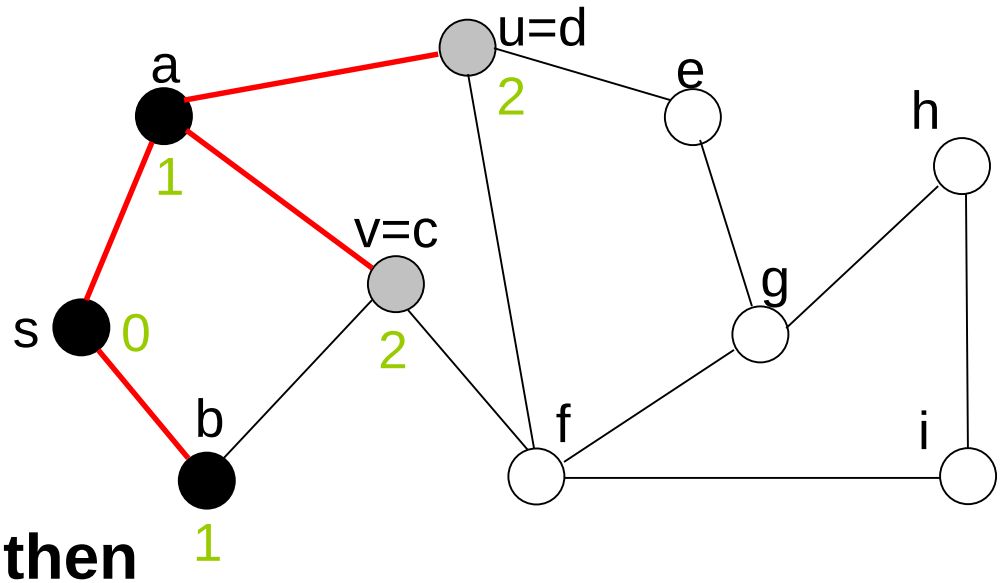
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: d, c

Breitensuche - Beispiel

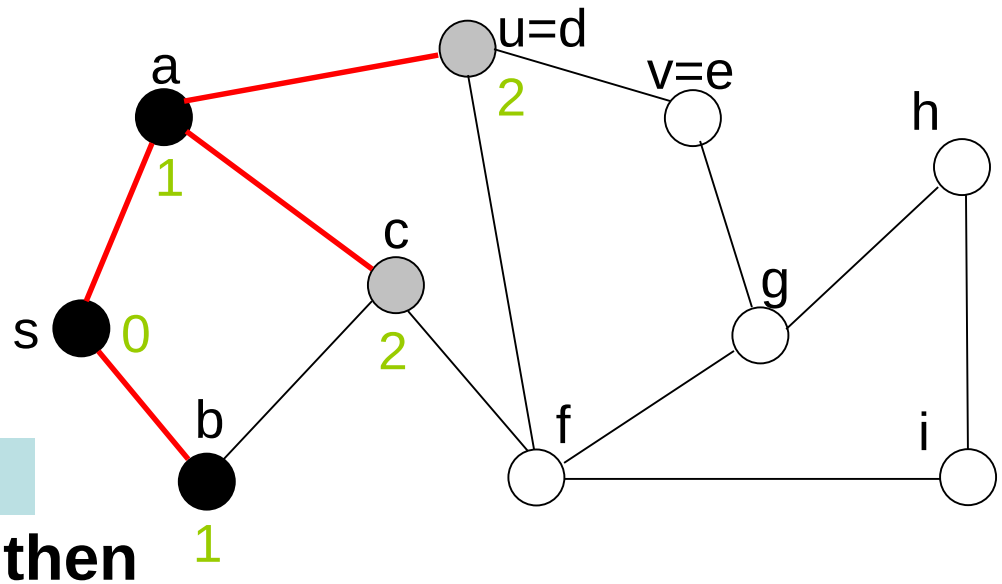
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c

Breitensuche - Beispiel

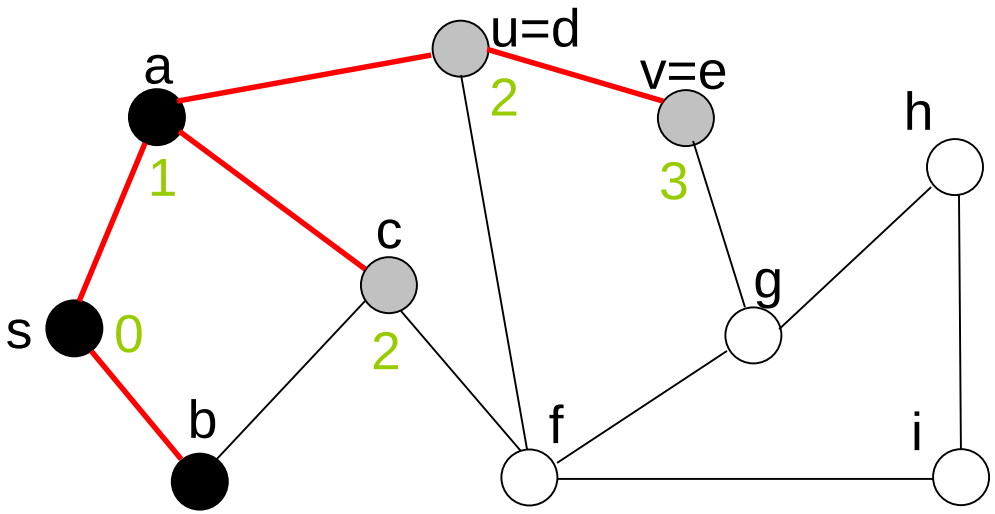
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c

Breitensuche - Beispiel

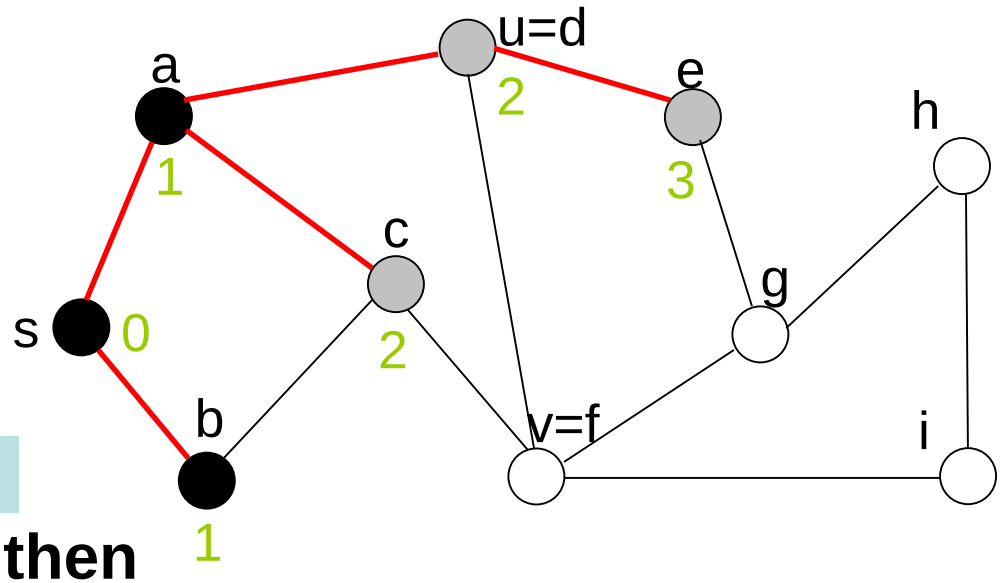
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e

Breitensuche - Beispiel

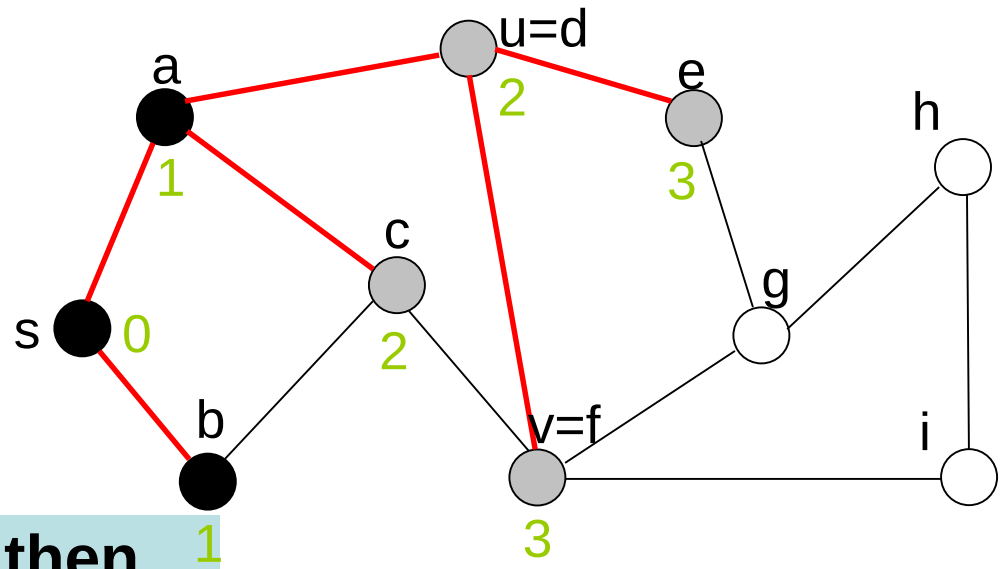
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e

Breitensuche - Beispiel

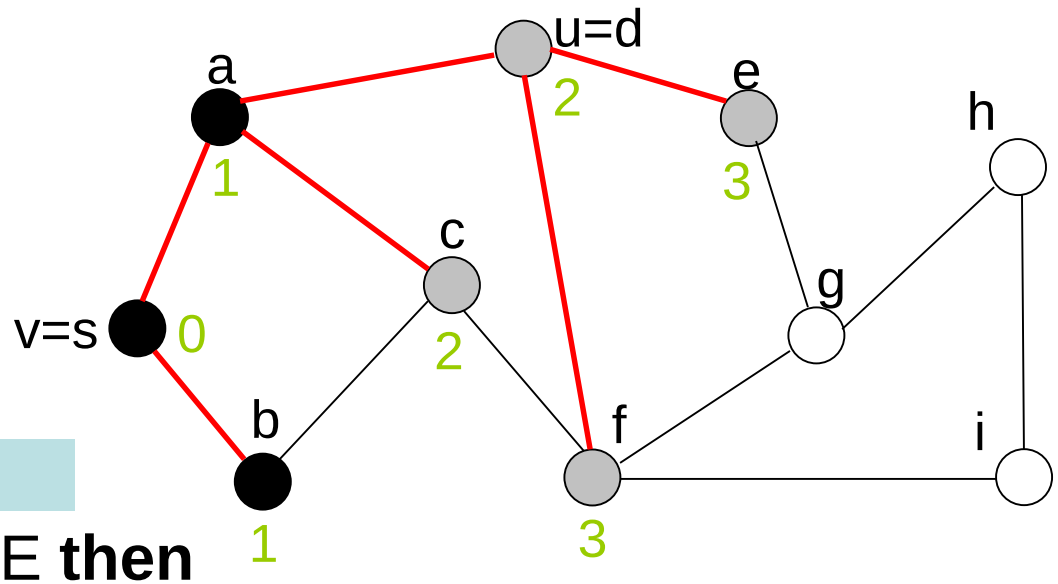
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breitensuche - Beispiel

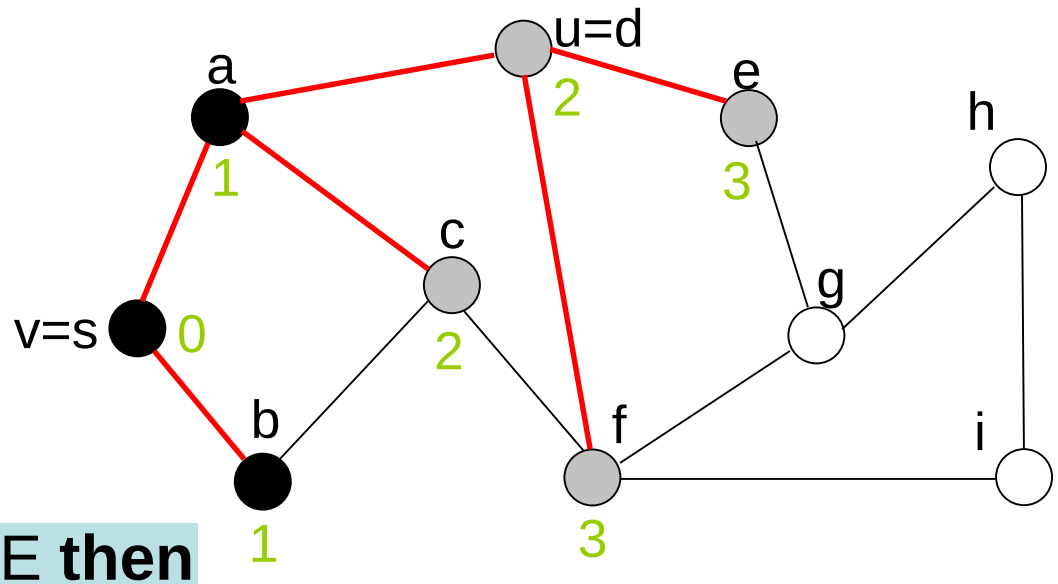
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breitensuche - Beispiel

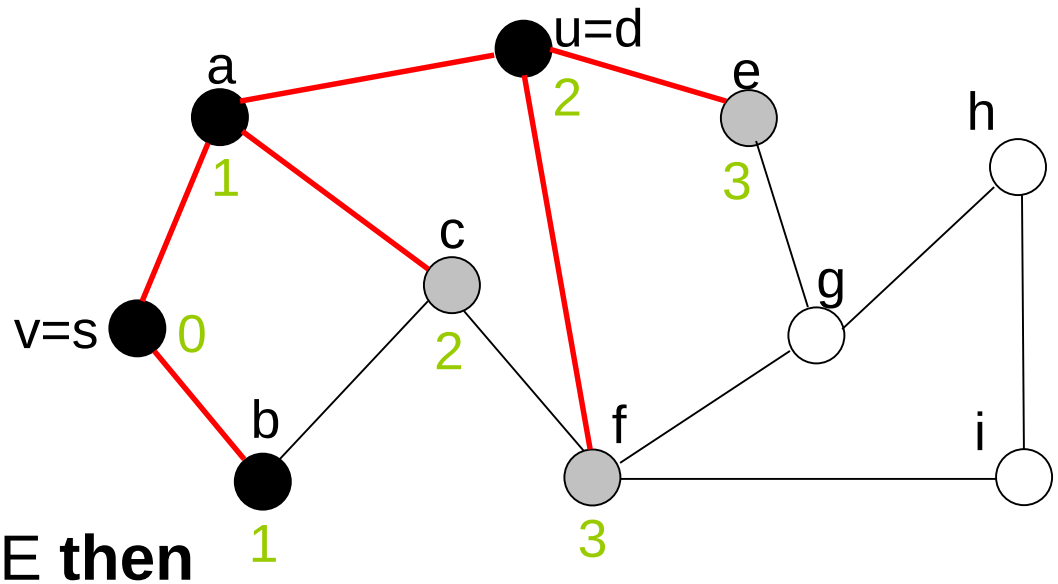
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breitensuche - Beispiel

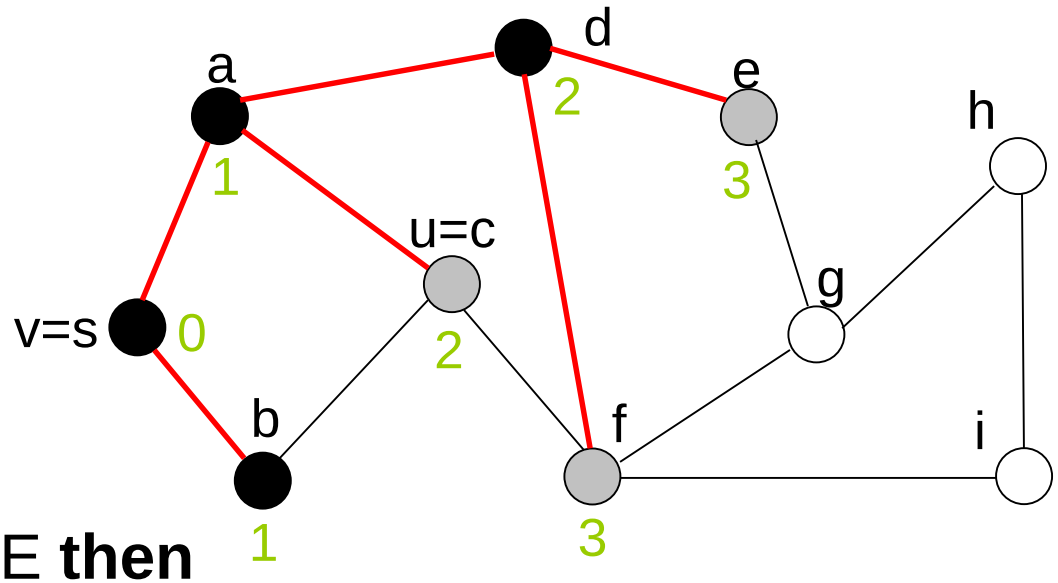
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: c, e, f

Breitensuche - Beispiel

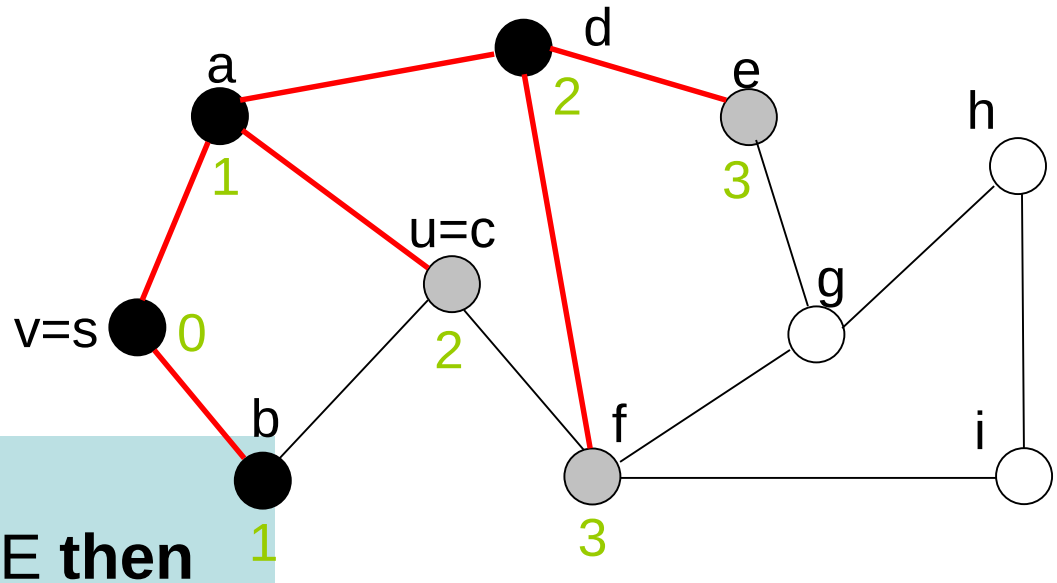
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: e, f

Breitensuche - Beispiel

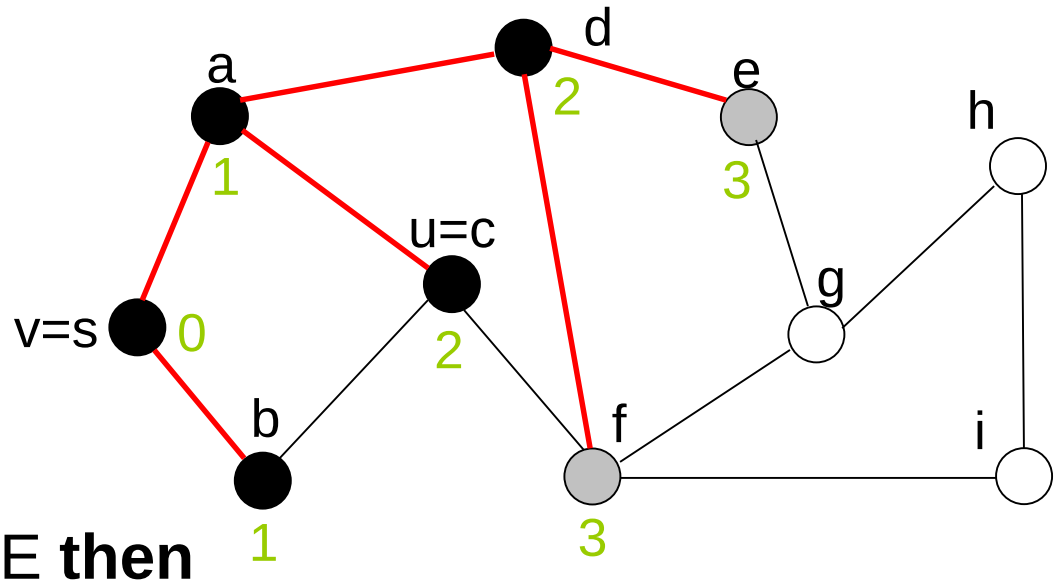
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: e, f

Breitensuche - Beispiel

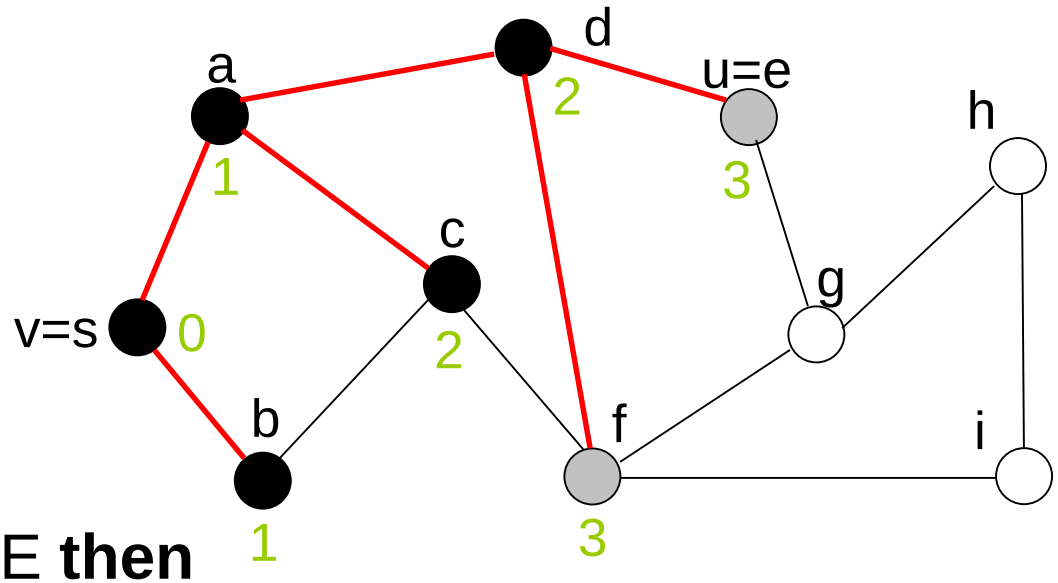
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: e, f

Breitensuche - Beispiel

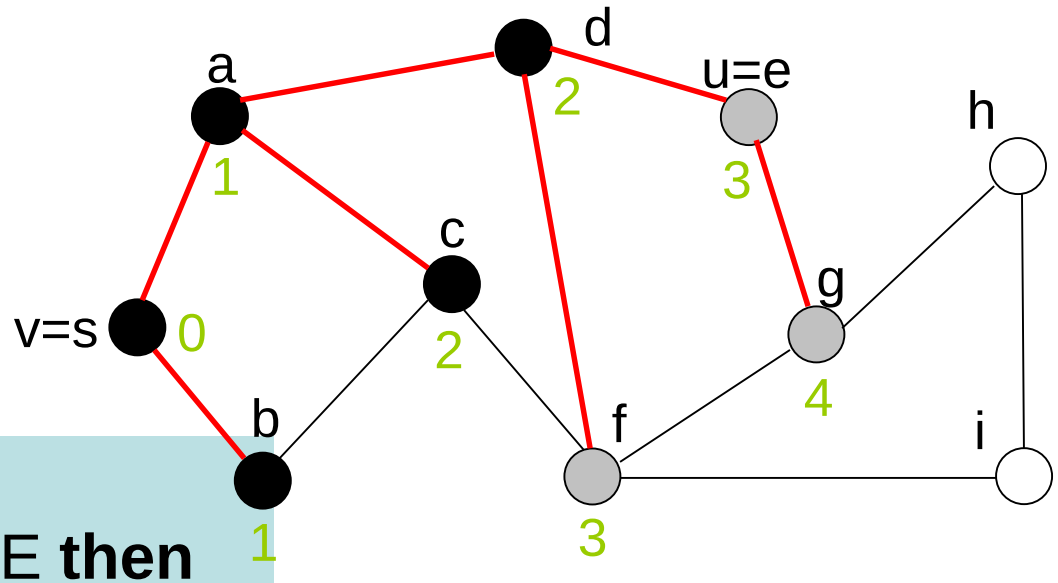
```
BFS(G,s)
1.  „initialisiere BFS“
2.  while Q≠∅ do
3.      u ← Dequeue(Q)
4.      for  v∈Adj[u] do
5.          if color[v]=WHITE then
6.              color[v] ← GRAY
7.              d[v] ← d[u]+1; π[v] ← u
8.              Enqueue(Q,v)
9.      color[u] ← BLACK
```



Q: f

Breitensuche - Beispiel

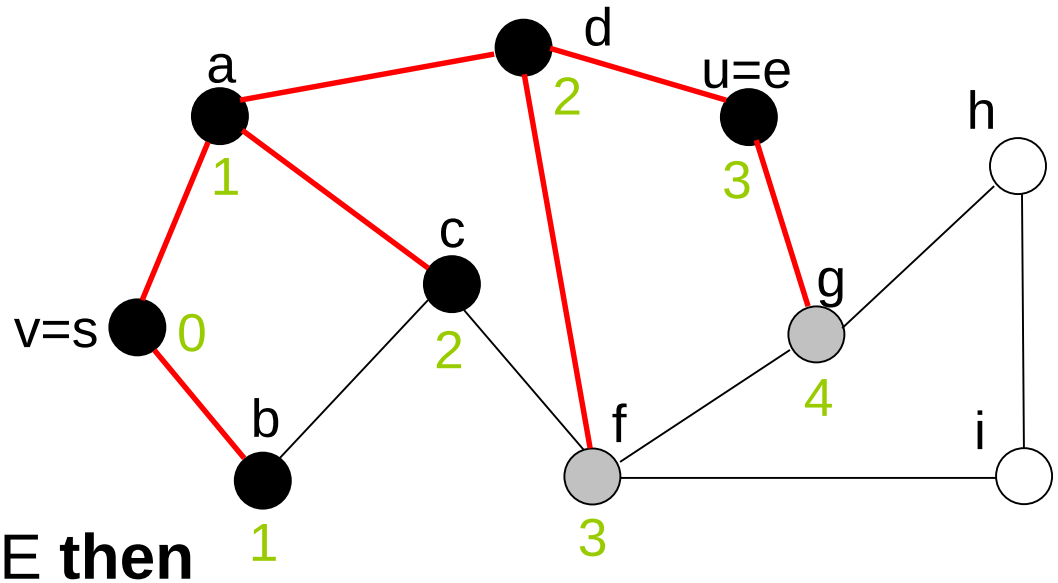
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: f, g

Breitensuche - Beispiel

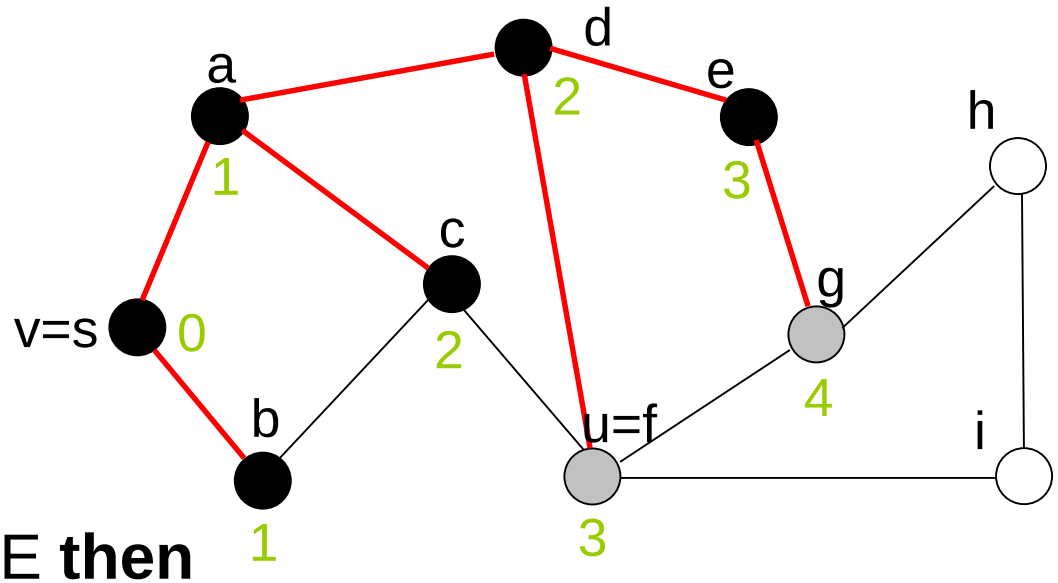
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: f, g

Breitensuche - Beispiel

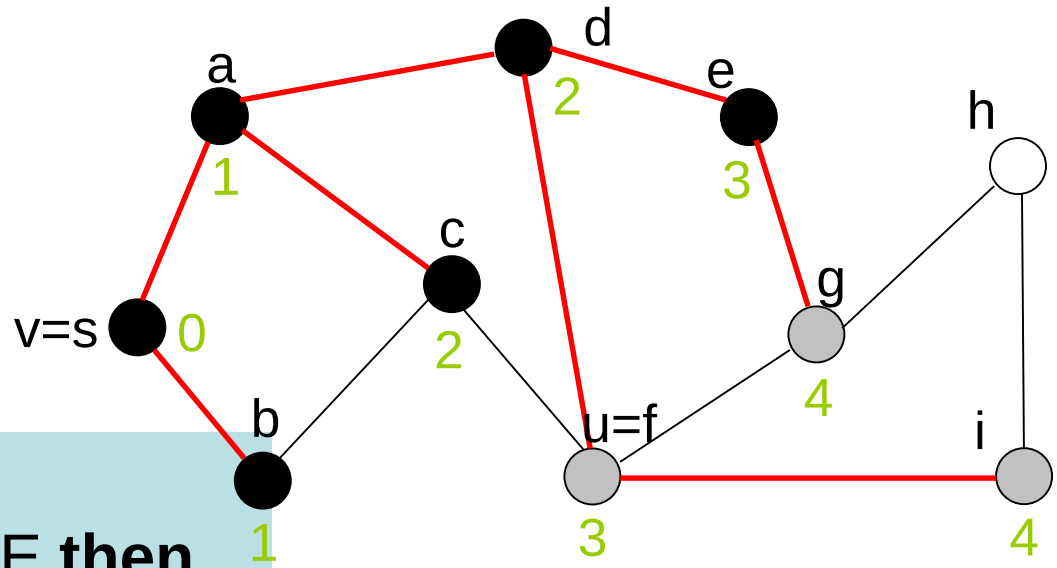
```
BFS(G,s)
1.  „initialisiere BFS“
2.  while Q≠∅ do
3.      u ← Dequeue(Q)
4.      for  v∈Adj[u] do
5.          if color[v]=WHITE then
6.              color[v] ← GRAY
7.              d[v] ← d[u]+1; π[v] ← u
8.              Enqueue(Q,v)
9.      color[u] ← BLACK
```



Q: g

Breitensuche - Beispiel

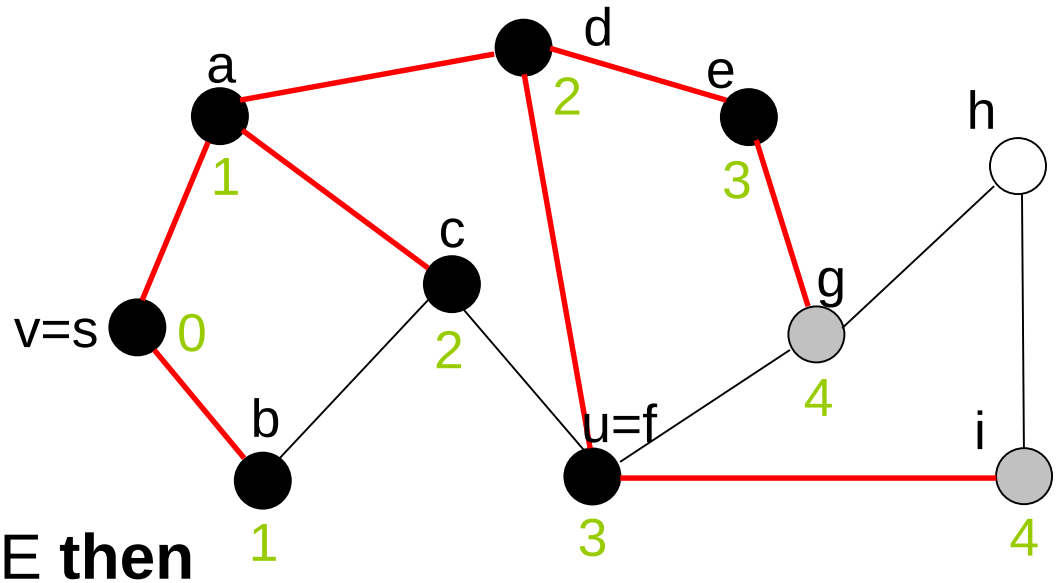
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: g, i

Breitensuche - Beispiel

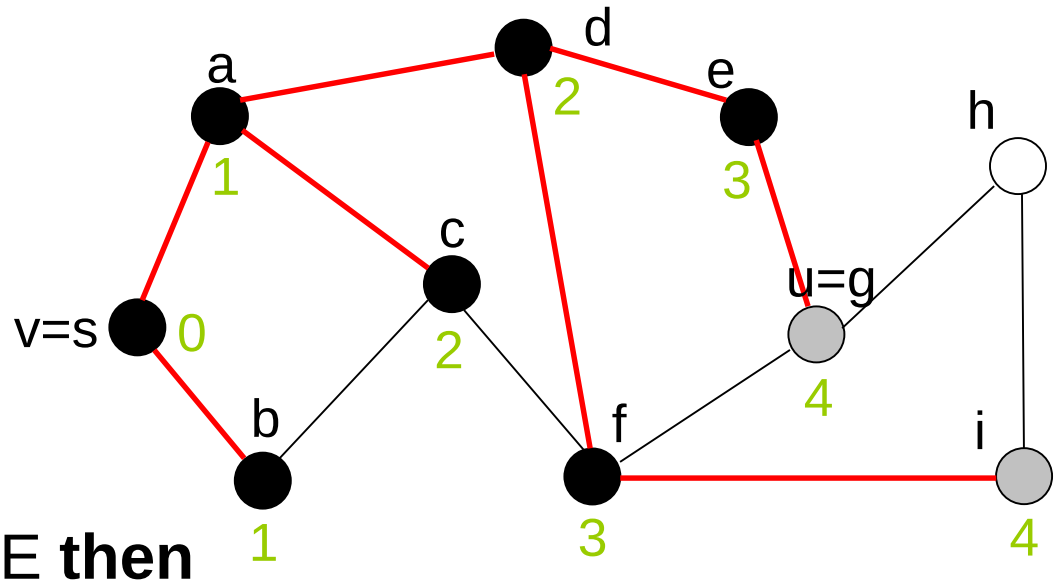
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: g, i

Breitensuche - Beispiel

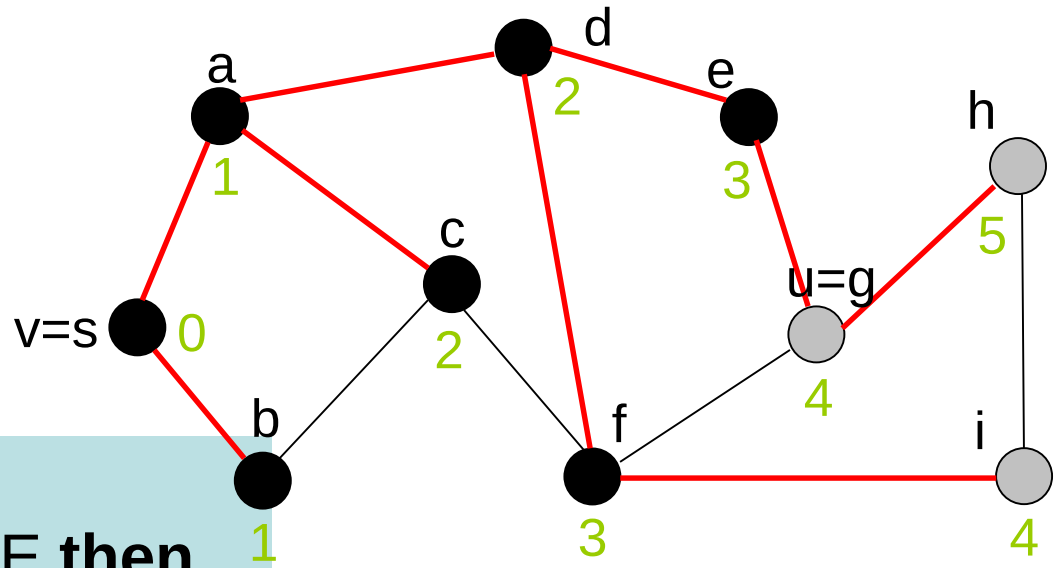
```
BFS(G,s)
1.  „initialisiere BFS“
2.  while Q≠∅ do
3.      u ← Dequeue(Q)
4.      for  v∈Adj[u] do
5.          if color[v]=WHITE then
6.              color[v] ← GRAY
7.              d[v] ← d[u]+1; π[v] ← u
8.              Enqueue(Q,v)
9.      color[u] ← BLACK
```



Q: i

Breitensuche - Beispiel

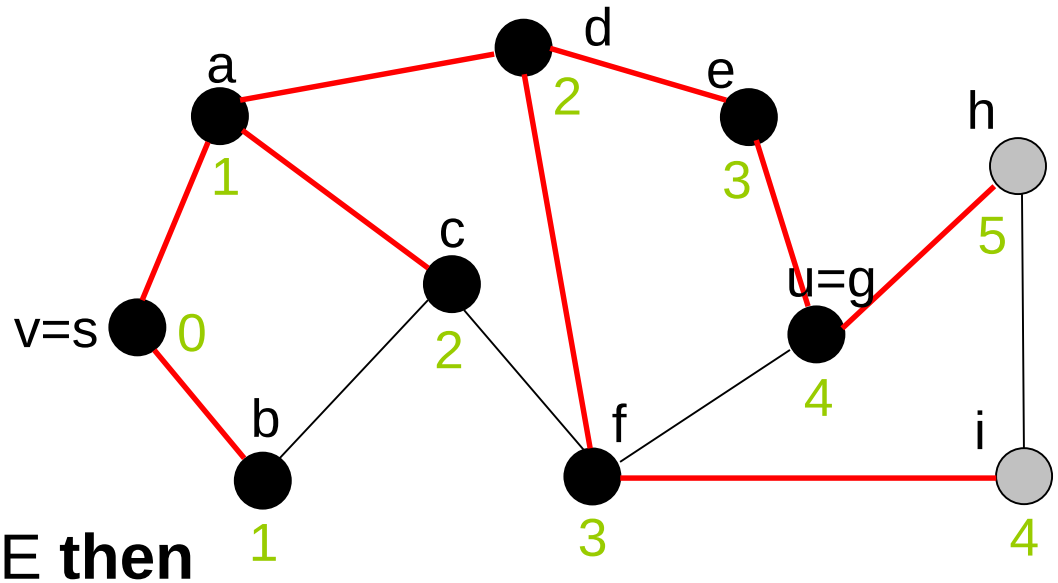
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: i, h

Breitensuche - Beispiel

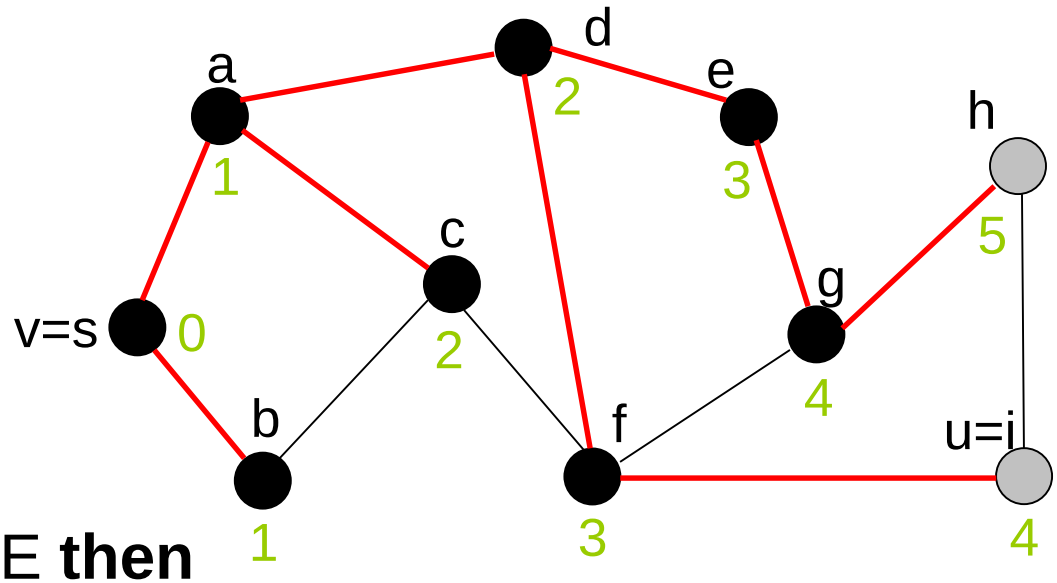
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: i, h

Breitensuche - Beispiel

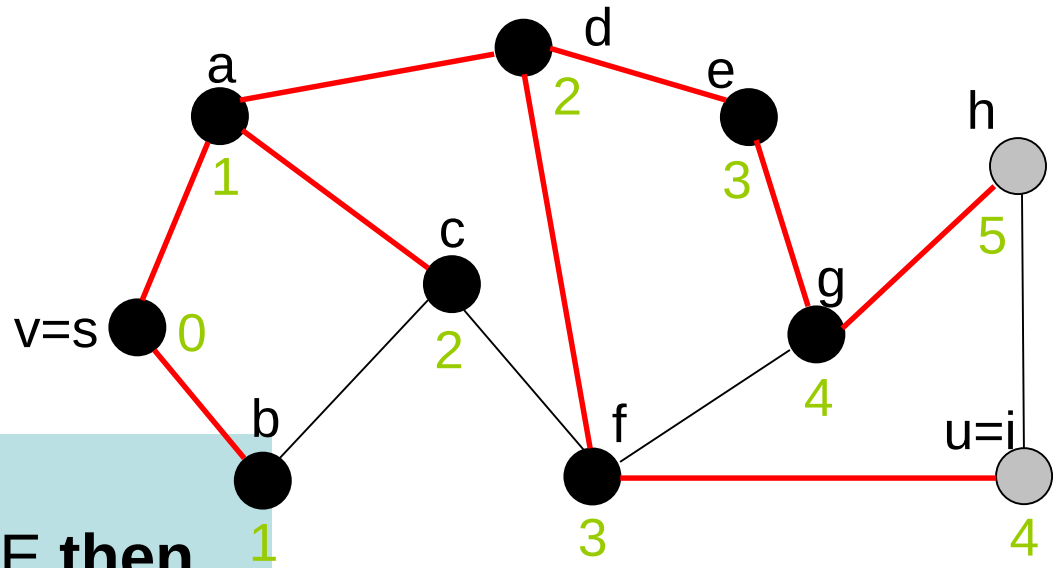
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: h

Breitensuche - Beispiel

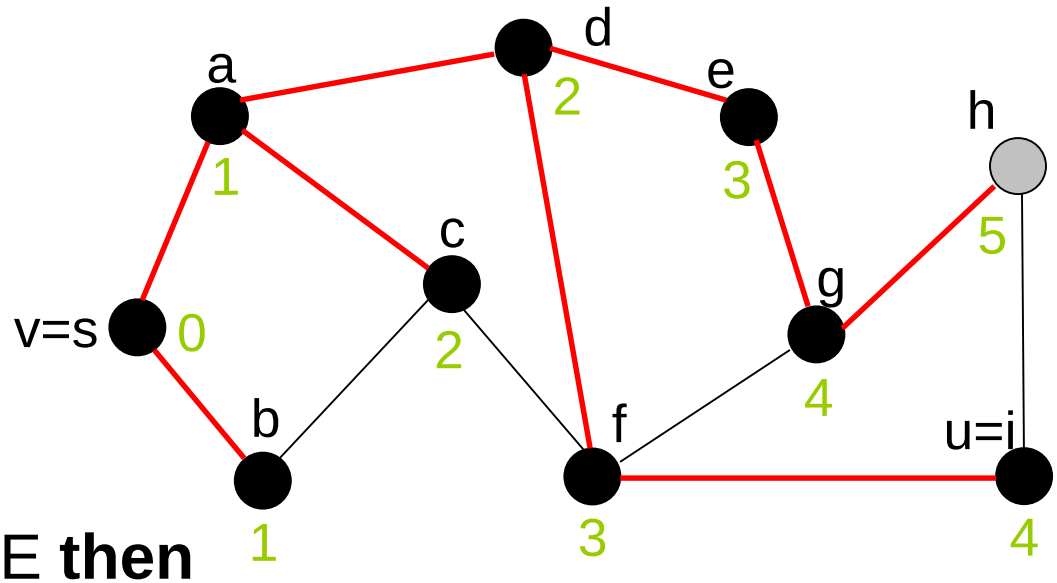
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: h

Breitensuche - Beispiel

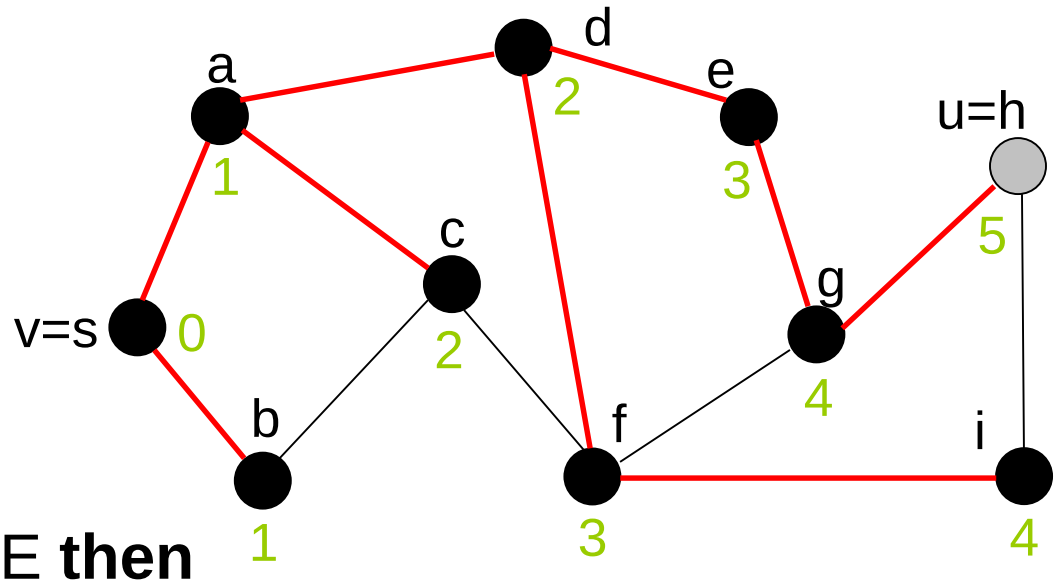
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q: h

Breitensuche - Beispiel

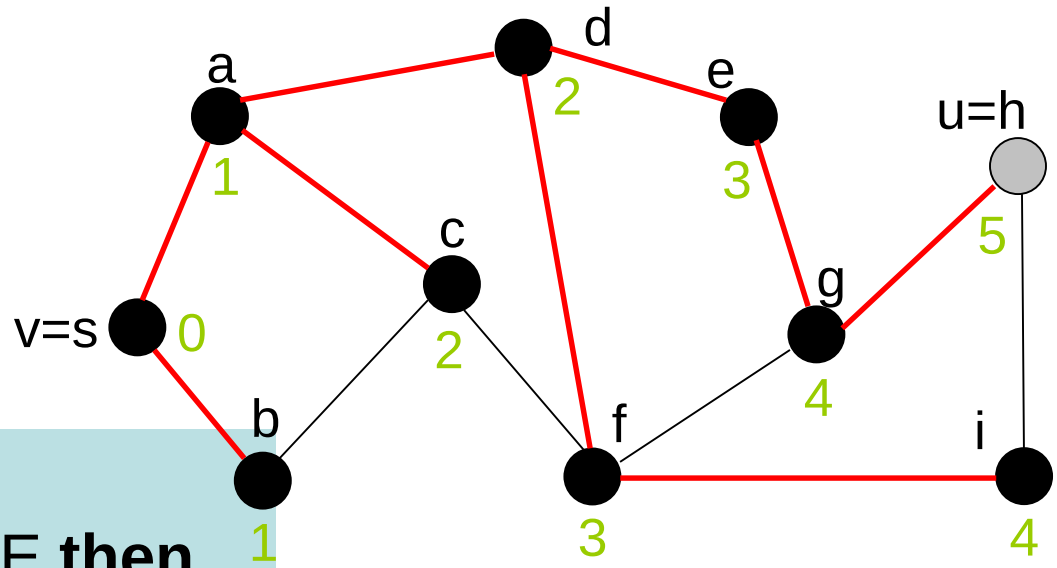
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche - Beispiel

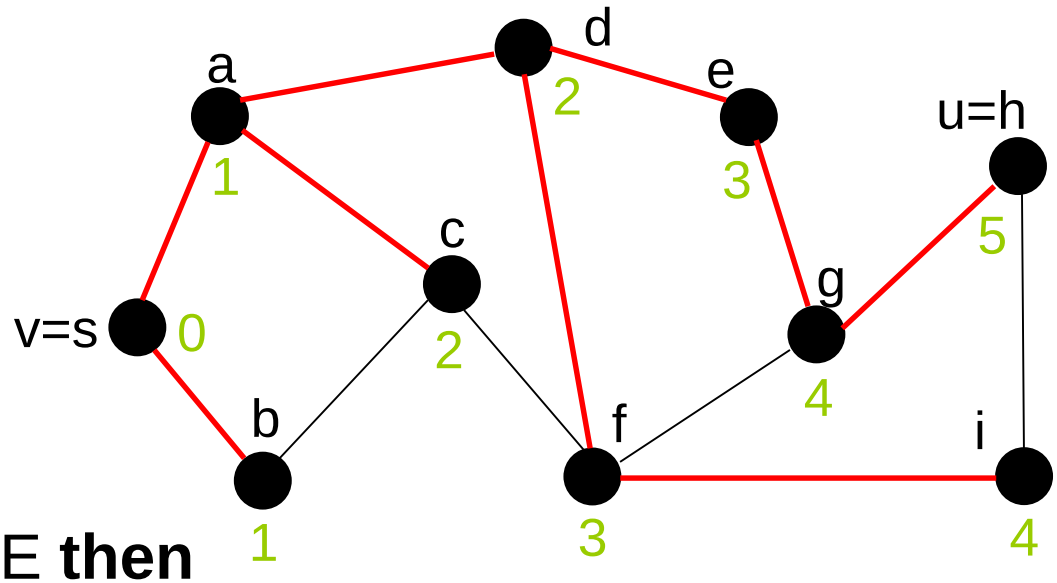
BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche - Beispiel

BFS(G,s)
1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u ← Dequeue(Q)
4. **for** v∈Adj[u] **do**
5. **if** color[v]=WHITE **then**
6. color[v] ← GRAY
7. d[v] ← d[u]+1; π[v] ← u
8. Enqueue(Q,v)
9. color[u] ← BLACK



Q:

Breitensuche – Laufzeitanalyse

- Zeilen 2-4 jeweils konstante Zeit. Werden $|V|-1$ mal durchlaufen. Damit Zeit $O(|V|)$.
- Zeilen 5-9 insgesamt konstante Zeit.
- Jeder Knoten wird nur einmal in Queue eingefügt und gelöscht.
- Schleife in Zeilen 12 -17 wird für jeden Eintrag v in Adjazenzlisten nur einmal durchlaufen.
- Zeilen 12-17 pro Durchlauf Zeit $O(1)$.

Breitensuche – Laufzeitanalyse

- Gesamtzeit für Durchläufe der Schleife in Zeilen 11-18 insgesamt $O(|E|)$. Denn Gesamtgröße aller Adjazenzlisten $\Theta(|E|)$.

Satz 14.1: Bei Eingabe von Graph G=(V,E) und Quelle s besitzt Algorithmus BFS Laufzeit

$$O(|V| + |E|)$$

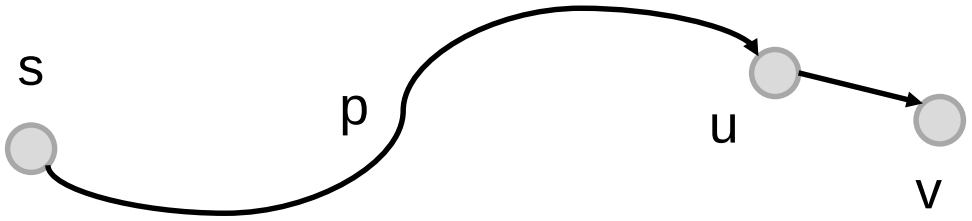
Breitensuche – Erreichbarkeit, Kürzeste Pfade (1)

Lemma 14.2: Sei $G=(V,E)$ ein gerichteter Graph. Sei $s \in V$ beliebig. Für jede Kante $(u,v) \in E$ gilt

$$\delta(s,v) \leq \delta(s,u)+1$$

Beweis:

- Sei p ein kürzester Weg von s nach u.



- Sei $p' = p \circ (u,v)$. Offensichtlich ist

$$\delta(s,v) \leq |p'| = |p|+1 = \delta(s,u)+1$$

Breitensuche – Erreichbarkeit, Kürzeste Pfade (2)

Lemma 14.3: Sei $G=(V,E)$ ein gerichteter Graph. Sei $s \in V$ beliebig und (s,G) die Eingabe für BFS. Nach Beendigung von BFS gilt für jeden Knoten $v \in V$, dass

$$d[v] \geq \delta(s,v)$$

Beweis:

- Sei S die Menge der Knoten, die bereits in der Queue Q waren oder sind.
- Schleifeninvariante: Für alle $v \in S$ gibt es einen Weg der Länge $d[v]$ von s nach v.
- Die Terminierung ergibt dann Lemma 14.3.

Breitensuche – Erreichbarkeit, Kürzeste Pfade (3)

Lemma 14.4: Die Queue Q enthalte zu einem beliebigen Zeitpunkt des Ablaufs von BFS die Knoten $(v_1,\ldots,v_r)$, wobei v_1 der Kopf head[Q] und v_r das Ende tail[Q] sei. Dann gilt

$$d[v_i] \leq d[v_{i+1}] \text{ für alle } i \in \{1,\ldots,r-1\} \text{ und } d[v_r] \leq d[v_1]+1$$

Beweis:

- Verwende die Aussage als Schleifeninvariante.

Korollar 14.5: Knoten v_i werde während des Ablaufs von BFS vor Knoten v_j in die Queue Q eingefügt. Zum Zeitpunkt, an dem v_j in die Queue Q eingefügt wird, gilt

$$d[v_i] \leq d[v_j]$$

Breitensuche – Erreichbarkeit, Kürzeste Pfade (4)

Satz 14.6: Sei G=(V,E) ein gerichteter oder ungerichteter Graph. Sei $s \in V$ beliebig und s,G Eingabe für BFS. Nach Beendigung von BFS gilt für jeden Knoten $v \in V$

$$d[v] = \delta(s,v).$$

Insbesondere sind die von s aus erreichbaren Knoten v die Knoten mit $d[v] < \infty$.

Weiter ist für jeden von s aus erreichbaren Knoten $v \neq s$ ein kürzester Pfad von s zu v gegeben durch einen kürzesten Pfad von s zum Vorgänger $\pi[v]$ von v erweitert um die Kante $(\pi[v],v)$.

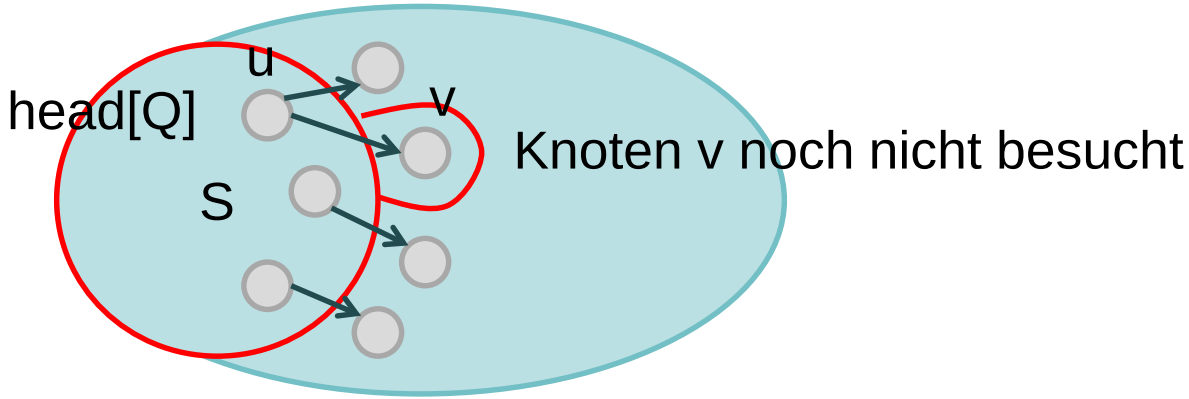
Breitensuche – Erreichbarkeit, Kürzeste Pfade (5)

Beweis von Satz 14.6:

- S: Menge der Knoten, die in Q waren oder sind

Invariante:

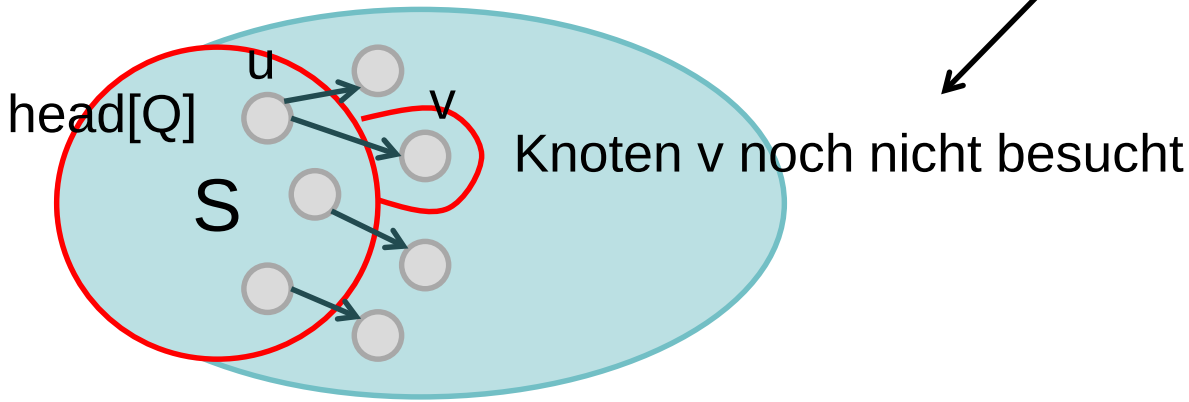
1. Für jedes $u \in S$: $d[u]=\delta(s,u)$ und $d[u] \leq \min_{v \in V \setminus S} \delta(s,v)$
2. Für jedes $v \in V$: $d[v] \geq \delta(s,v)$ (Lemma 14.3)
3. Korollar 14.5



Breitensuche – Erreichbarkeit, Kürzeste Pfade (6)

Beweis von Satz 14.6:

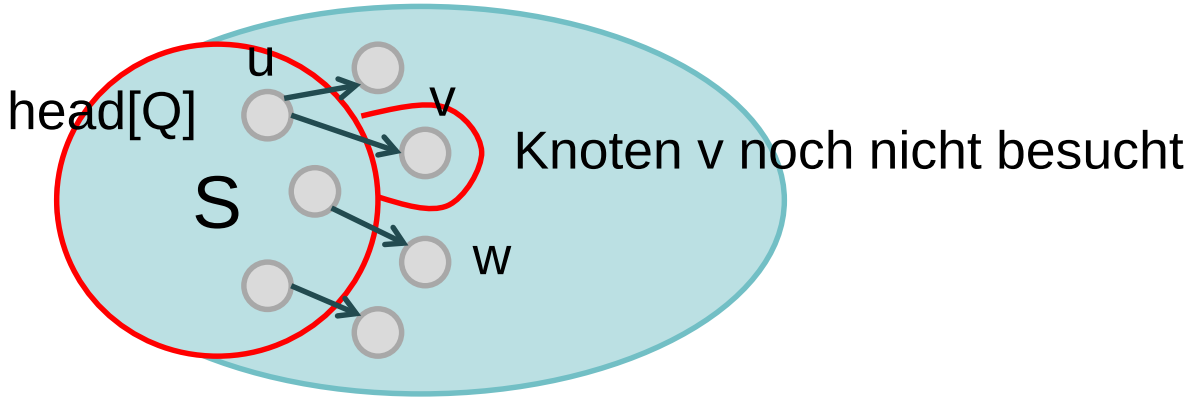
- Angenommen, $d[v](:=d[u]+1) > \delta(s,v)$.
- Dann existiert ein $u' \in V$ mit $\delta(s,u') \leq \delta(s,u)$ und $(u',v) \in E$.
- Laut Inv. 1 ist dann aber $u' \in S$, und laut Inv. 3 ist u' vor u im BFS betrachtet worden.
- Da v bei Betrachtung von u' bereits in Q eingefügt worden wäre, kann v bei Betrachtung von u nicht mehr in $V \backslash S$ sein. Widerspruch!



Breitensuche – Erreichbarkeit, Kürzeste Pfade (7)

Beweis von Satz 14.6:

- Also muss $d[v] \leq \delta(s,v)$ sein.
- Zusammen mit Inv. 2 ergibt sich daraus, dass $d[v] = \delta(s,v)$.
- Weiterhin kann es kein $w \in V \setminus S$ geben mit $\delta(s,w) < d[v]$.
- Sonst können wir wiederum einen Widerspruch erzeugen.
- Daher gilt auch nach Einfügung von v in Q Inv. 1.
- Wir haben schon gezeigt, dass Inv. 2 und 3 gelten.



Breitensuchbäume

Betrachten nach BFS mit Eingabe G,s den Graphen $G_\pi = (V_\pi, E_\pi)$ mit

$$V_\pi = \{v \in V \mid \pi[v] \neq \text{NIL}\} \cup \{s\}.$$

und

$$E_\pi := \{(\pi[v], v) \mid v \in V_\pi \setminus \{s\}\}.$$

Dann gilt

Satz 14.7: $G_\pi = (V_\pi, E_\pi)$ ist ein Baum. V_π enthält genau die von s aus erreichbaren Knoten in G. Für jeden Knoten $v \in V_\pi$ ist der eindeutige Pfad von s zu v in G_π ein kürzester Pfad von s zu v in G.

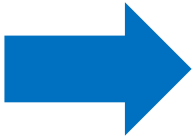
Tiefensuche

Erinnerung:

- Suche zunächst „tiefer“ im Graph
- Neue Knoten werden immer vom zuletzt gefundenen Knoten entdeckt
- Sind alle adjazenten Knoten des zuletzt gefundenen Knoten v bereits entdeckt, springe zurück zum Knoten, von dem aus v entdeckt wurde
- Wenn irgendwelche unentdeckten Knoten übrigbleiben, starte Tiefensuche von einem dieser Knoten

Tiefensuche

```
BFS(G,S)
1    for jeden Knoten u∈V\{s}
2        do color[u]←WHITE
3           d[u] ← ∞
4           p[u] ← NIL
5    color[s] ← GRAY
6    d[s] ← 0
7    π[s] ← NIL
8    Q ← { }
9    Enqueue(Q,s)
10   while Q ≠ { }
11       do u ← Dequeue(Q)
12          for v∈Adj[u]
13              do if color[v]=WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u]+1
16                         π[v] ← u
17                         Enqueue(Q,v)
18          color[u] ← BLACK
```



```
DFS(G,S)
1    for jeden Knoten u∈V\{s}
2        do color[u]←WHITE
3           d[u] ← ∞
4           p[u] ← NIL
5    color[s] ← GRAY
6    d[s] ← 0
7    π[s] ← NIL
8    Q ← { }
9    Push(Q,s)
10   while Q ≠ { }
11       do u ← Pop(Q)
12          for v∈Adj[u]
13              do if color[v]=WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u]+1
16                         π[v] ← u
17                         Push(Q,v)
18          color[u] ← BLACK
```

Tiefensuche

Statt iterativer Umsetzung gibt es auch eine rekursive Umsetzung der Tiefensuche.

Rekursive Umsetzung:

- Zu Beginn: alle Knoten weiß
- Entdeckte Knoten werden grau
- Abgearbeitete Knoten werden schwarz
- Zwei Zeitstempel: d[v] und f[v] (liegen zwischen 1 und 2|V|)
- d[v]: Zeit, zu der v entdeckt wird
- f[v]: Zeit, zu der v abgearbeitet ist

Tiefensuche

DFS(G)

1. **for each** vertex $u \in V$ **do** color[u] $\leftarrow$ weiß ; π[u] $\leftarrow$ nil
2. time $\leftarrow$ 0
3. **for each** vertex $u \in V$ **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

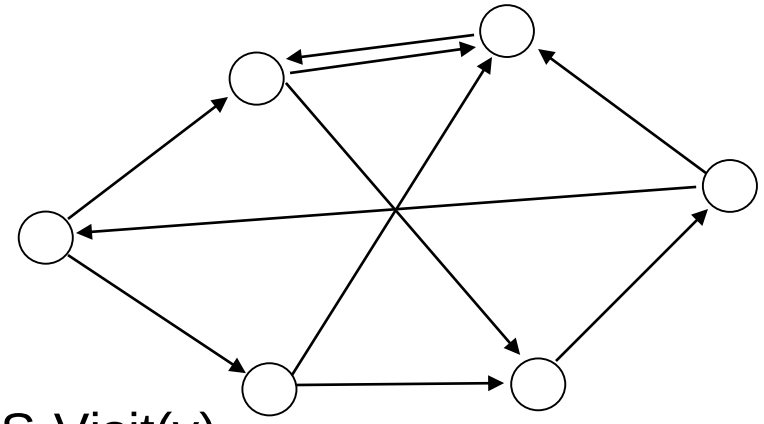
DFS-Visit(u)

1. color[u] $\leftarrow$ grau
2. time $\leftarrow$ time +1; d[u] $\leftarrow$ time
4. **for each** $v \in$ Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] $\leftarrow$ u ; DFS-Visit(v)
6. color[u] $\leftarrow$ schwarz
7. time $\leftarrow$ time+1; f[u] $\leftarrow$ time

Tiefensuche

DFS(G)

1. **for each** vertex u$\in$V **do** color[u] $\leftarrow$ weiß ; π[u] $\leftarrow$ nil
2. time $\leftarrow$ 0
3. **for each** vertex u$\in$V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)



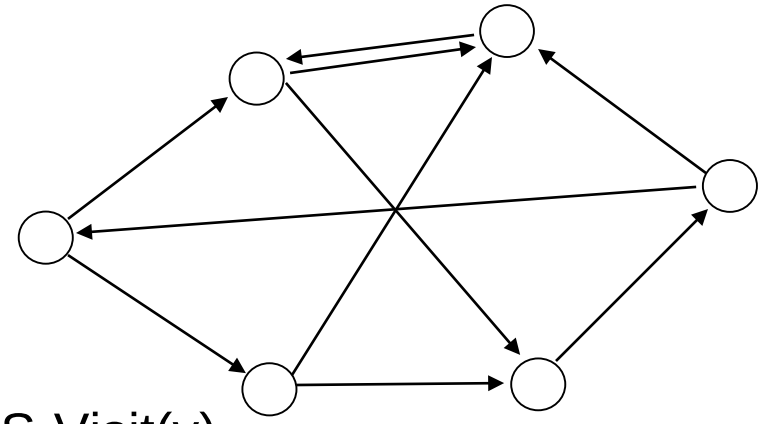
DFS-Visit(u)

1. color[u] $\leftarrow$ grau
2. time $\leftarrow$ time +1; d[u] $\leftarrow$ time
4. **for each** v$\in$Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] $\leftarrow$ u ; DFS-Visit(v)
6. color[u] $\leftarrow$ schwarz
7. time $\leftarrow$ time+1; f[u] $\leftarrow$ time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)



DFS-Visit(u)

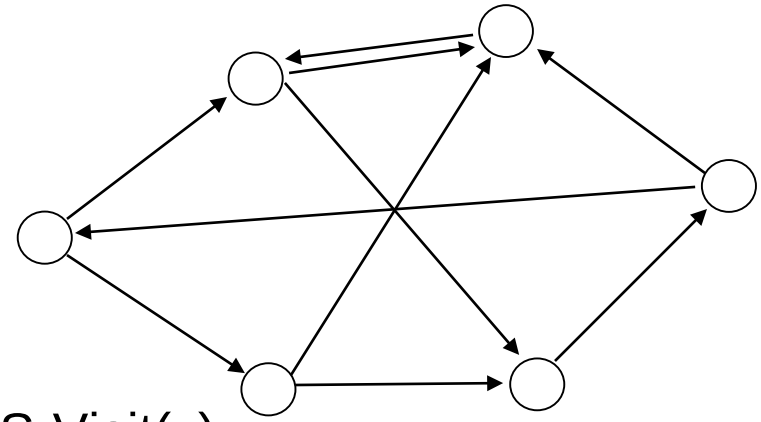
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

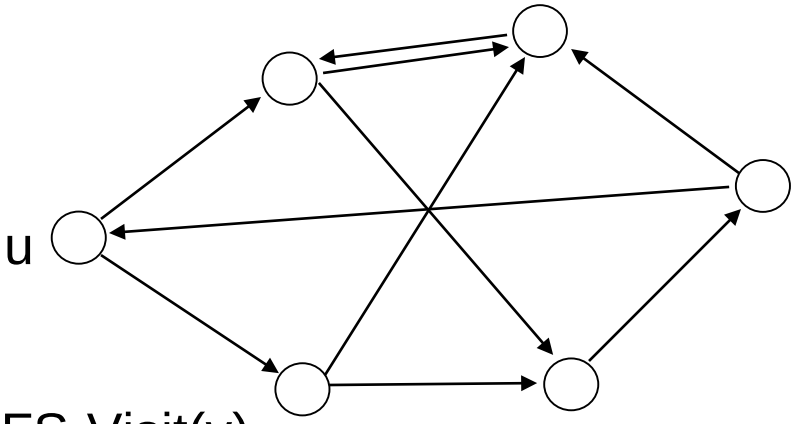
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time



Tiefensuche

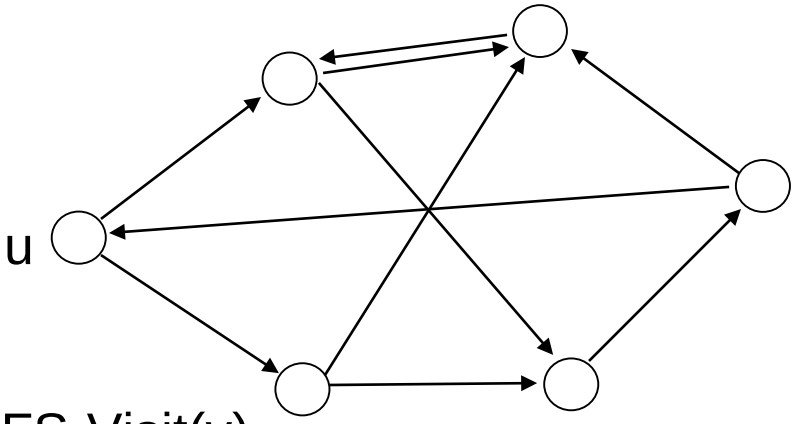
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time



Tiefensuche

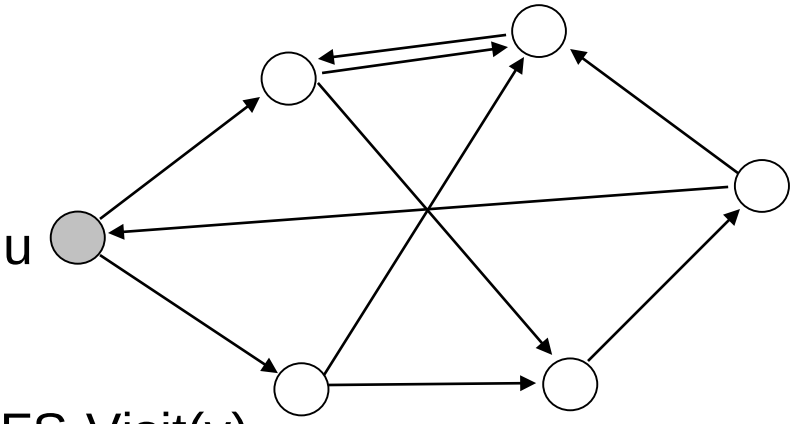
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=0

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

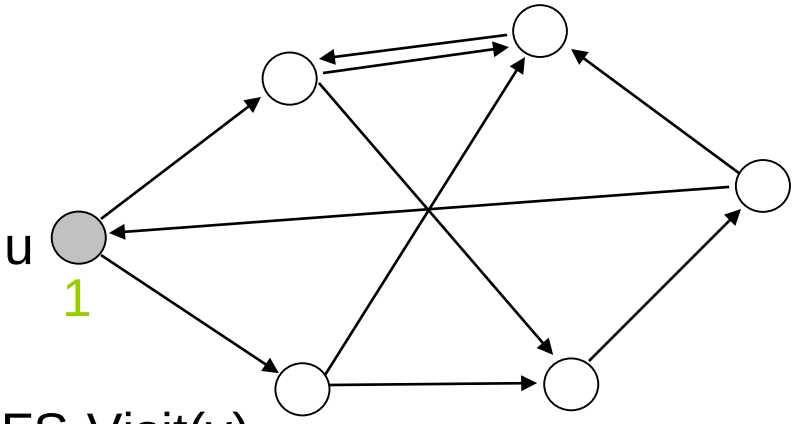


Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

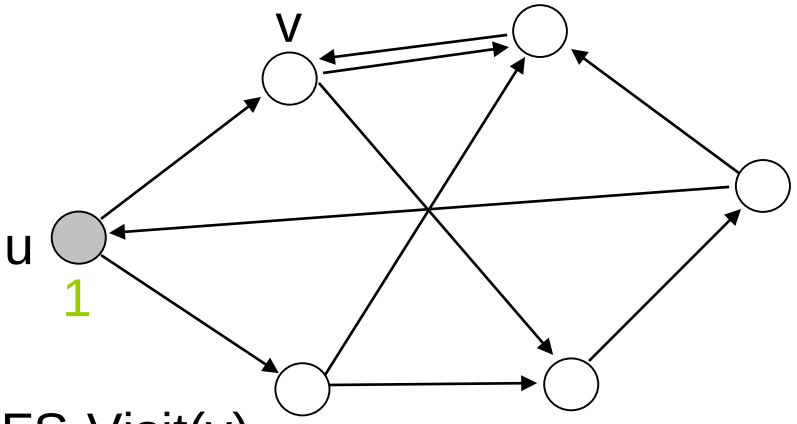
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

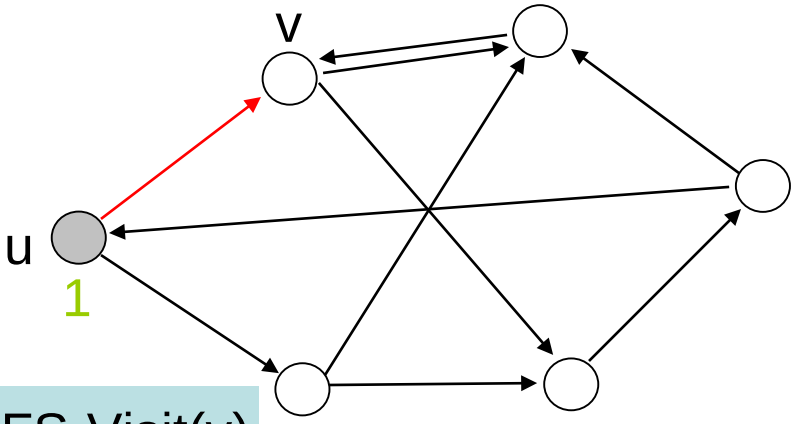
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

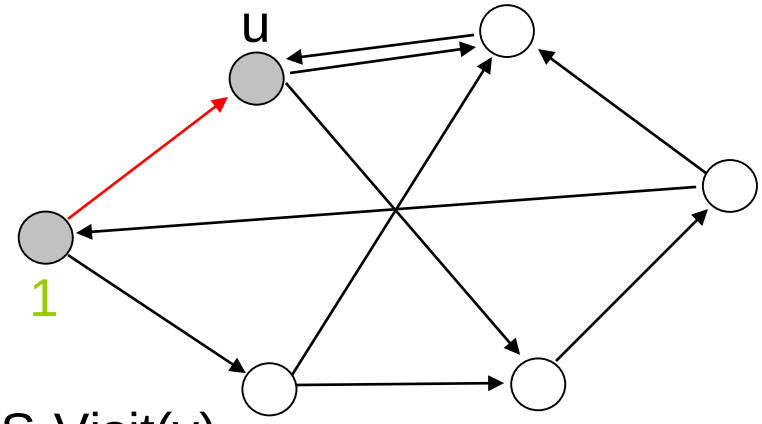
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=1



DFS-Visit(u)

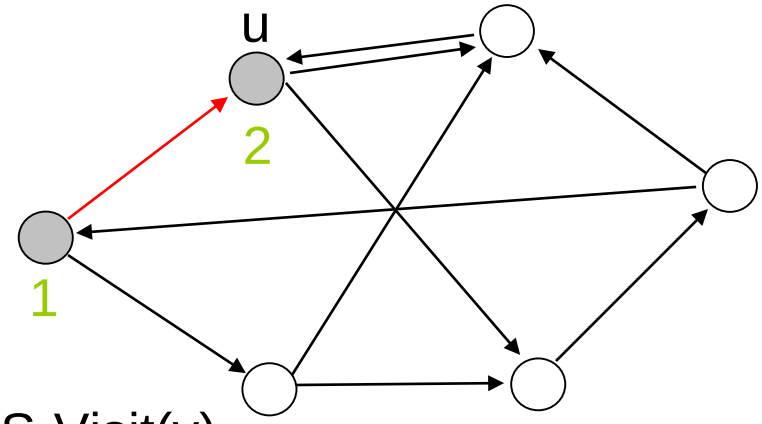
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

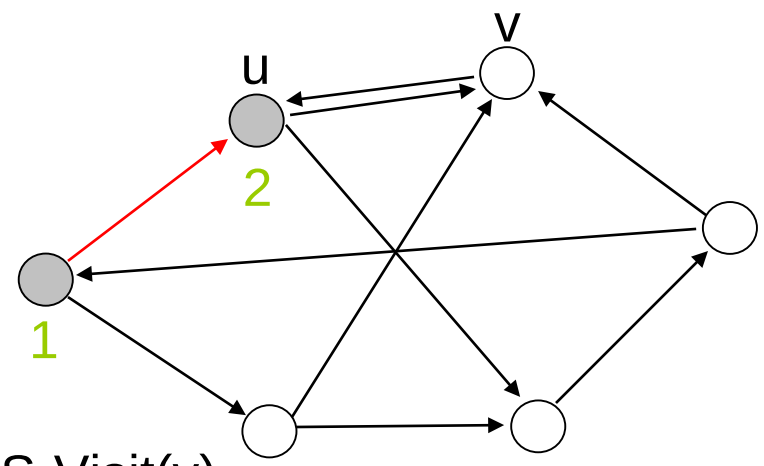
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

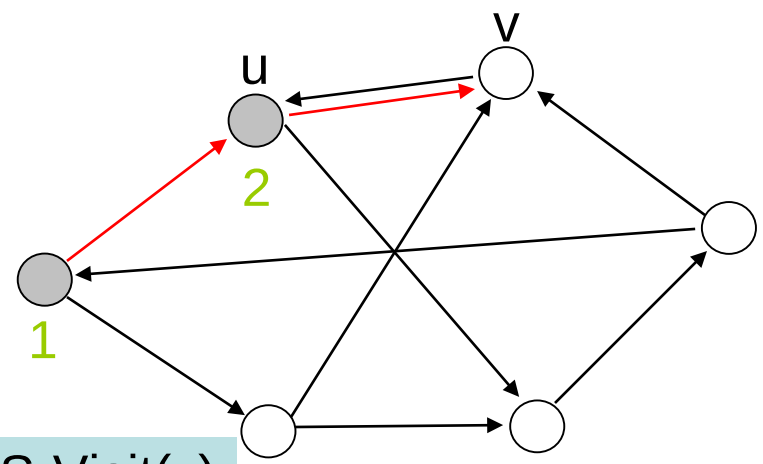
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

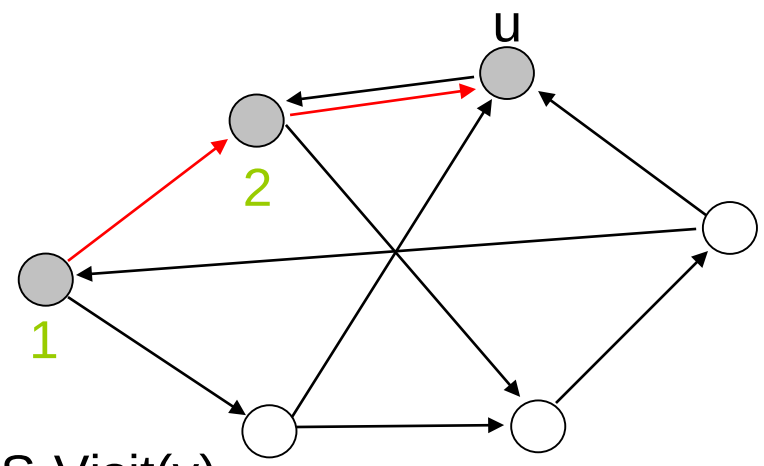
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=2



DFS-Visit(u)

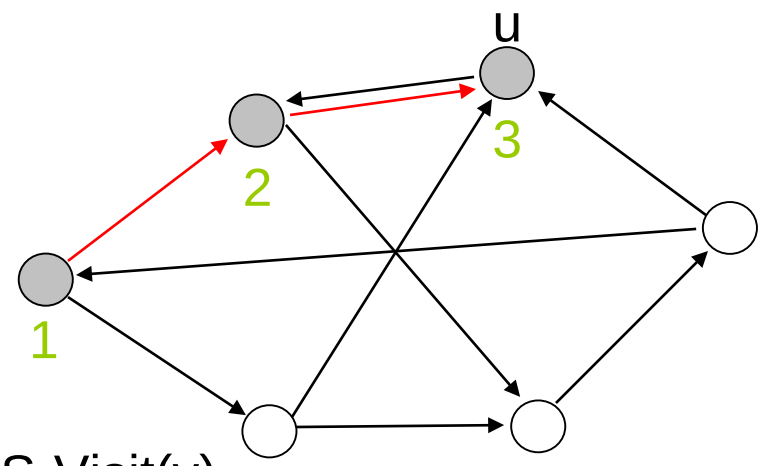
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=3



DFS-Visit(u)

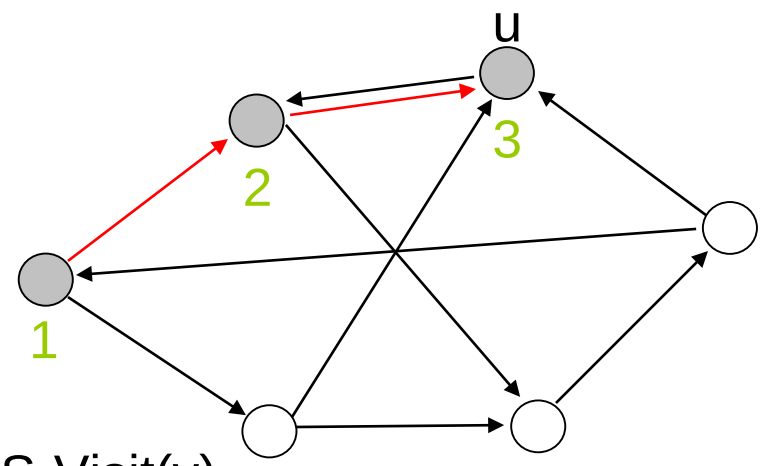
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=3



DFS-Visit(u)

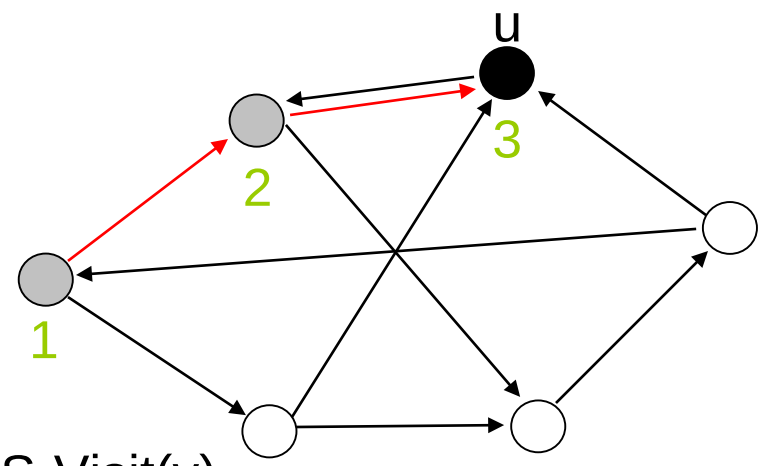
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=3



DFS-Visit(u)

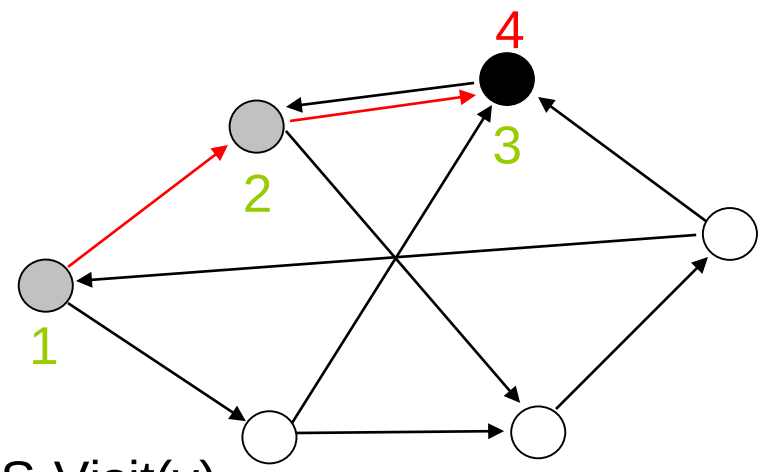
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=4



DFS-Visit(u)

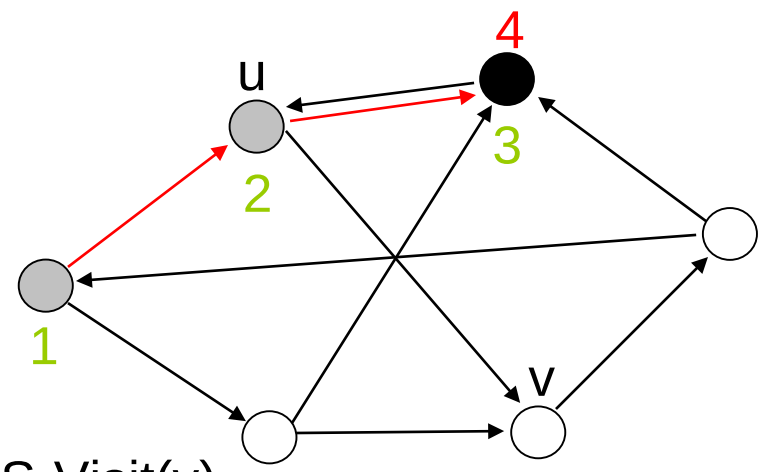
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=4



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

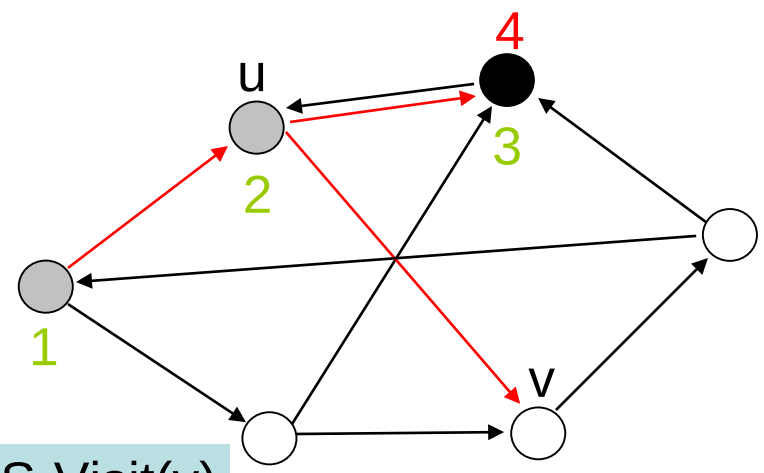
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=4

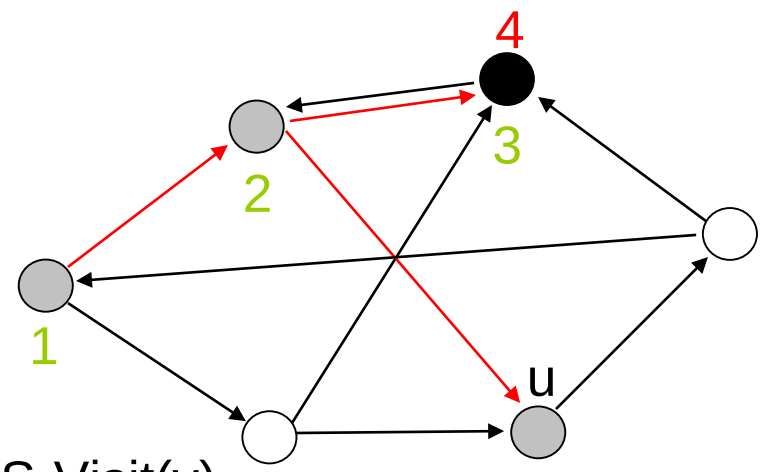


Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=4



DFS-Visit(u)

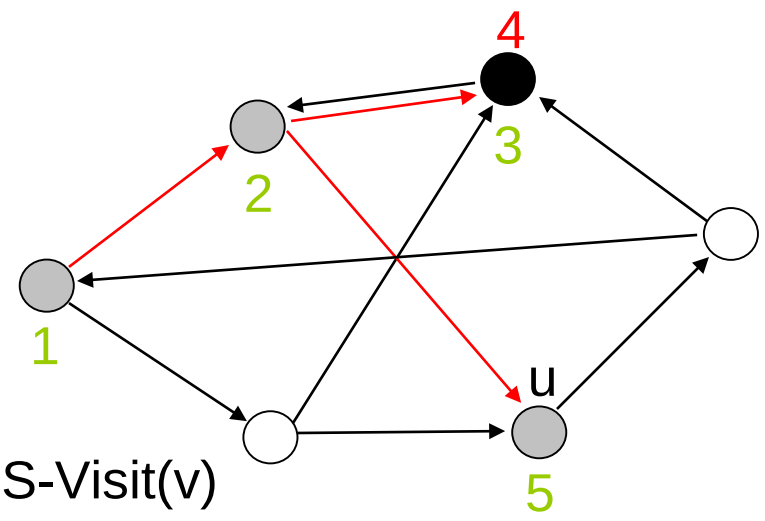
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=5



DFS-Visit(u)

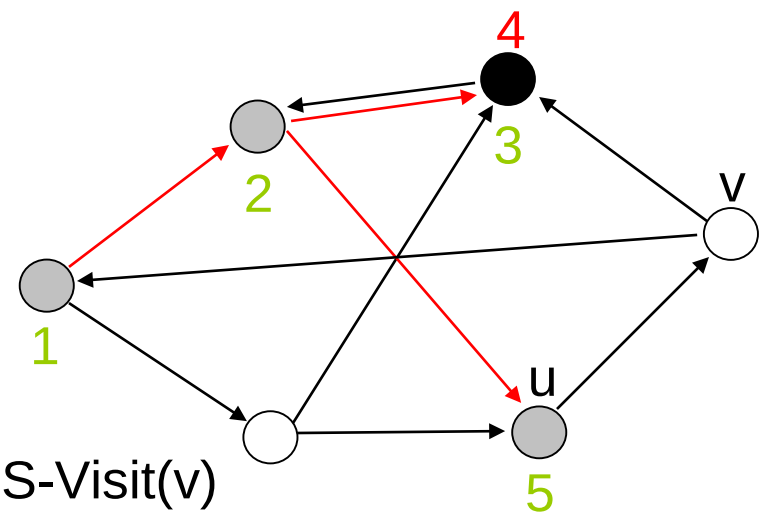
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=5



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

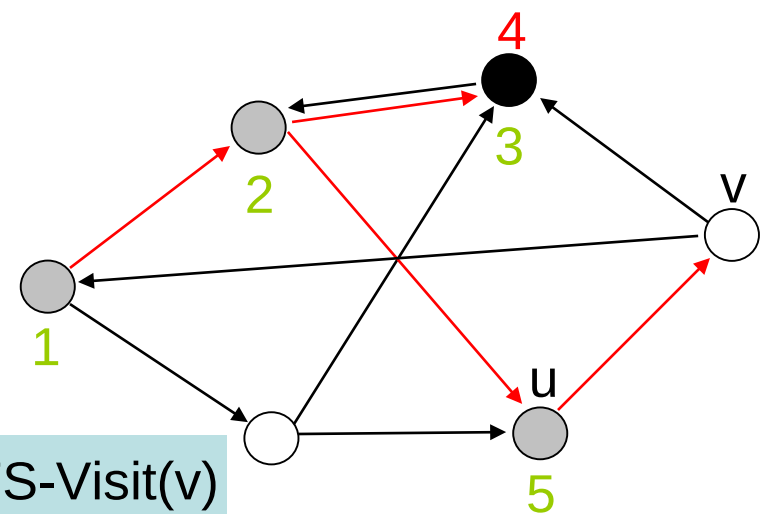
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=5

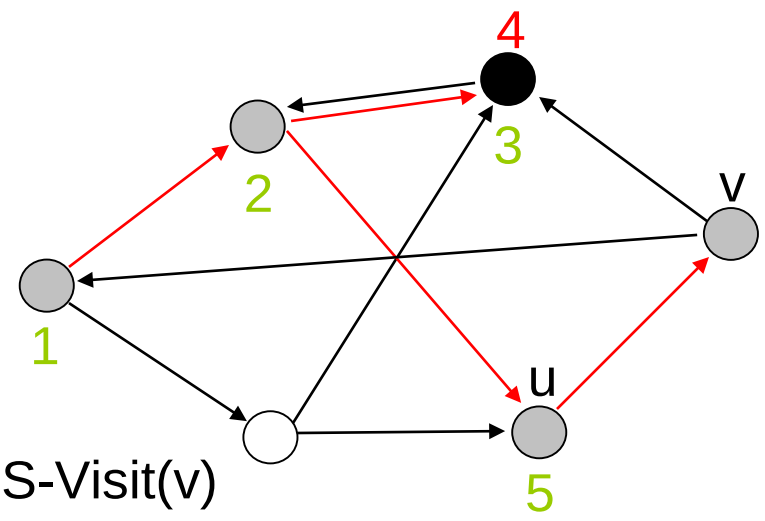


Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=5



DFS-Visit(u)

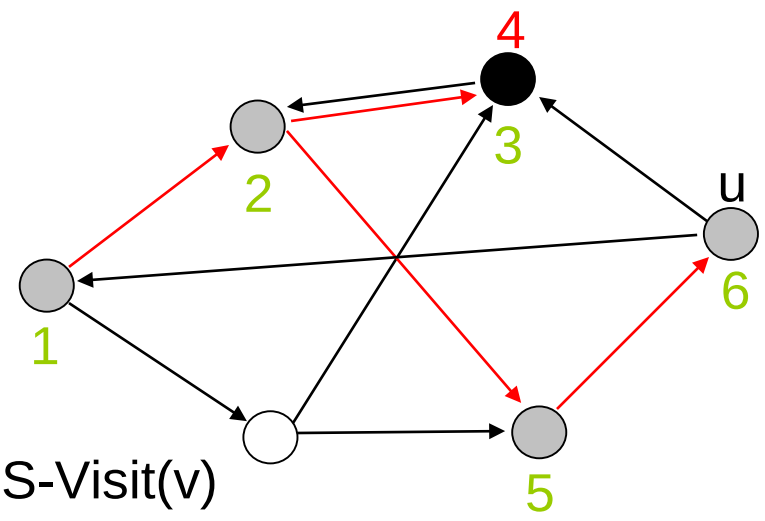
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

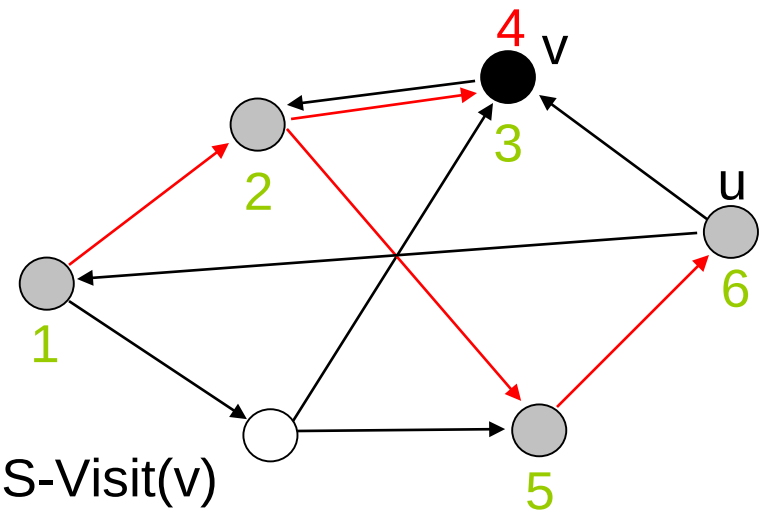
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

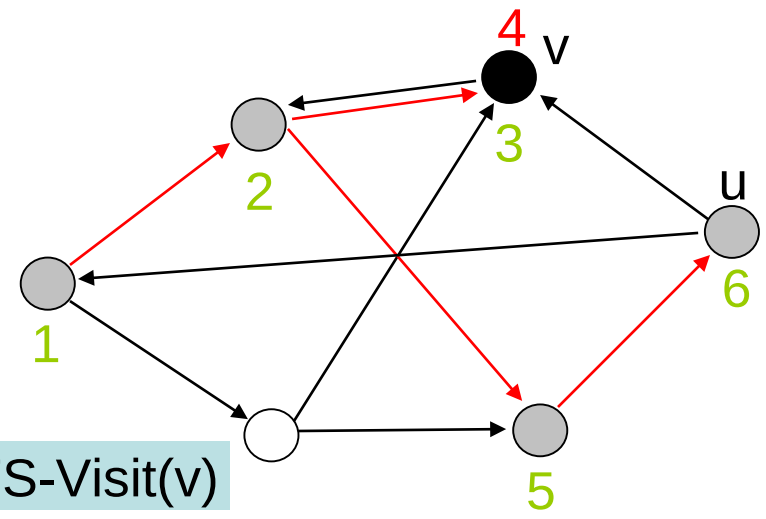
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=6

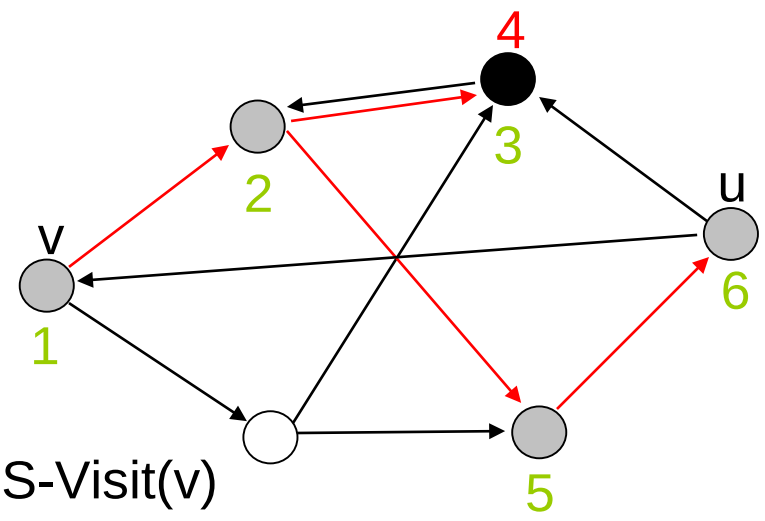


Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

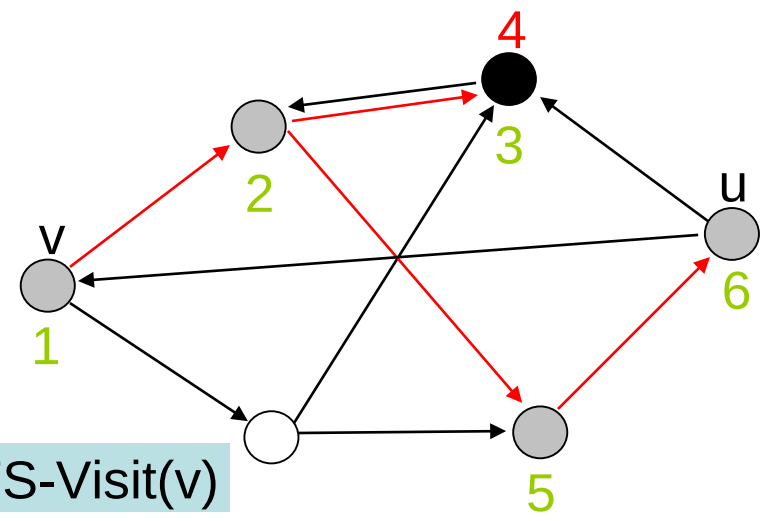
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

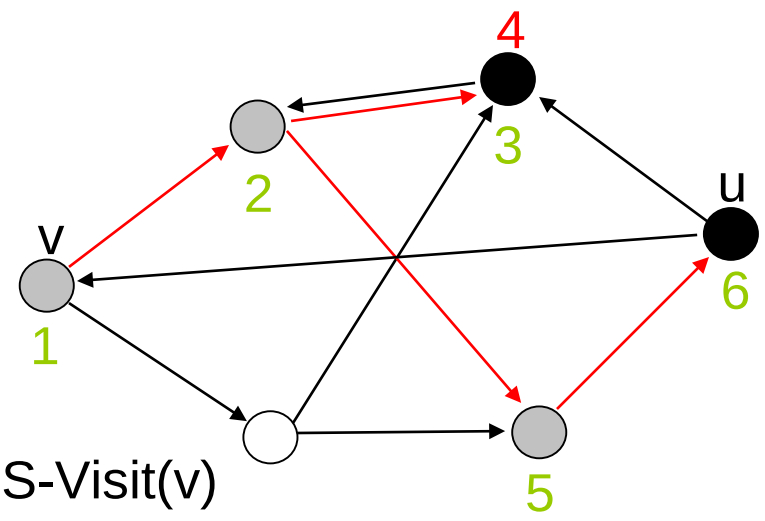
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=6



DFS-Visit(u)

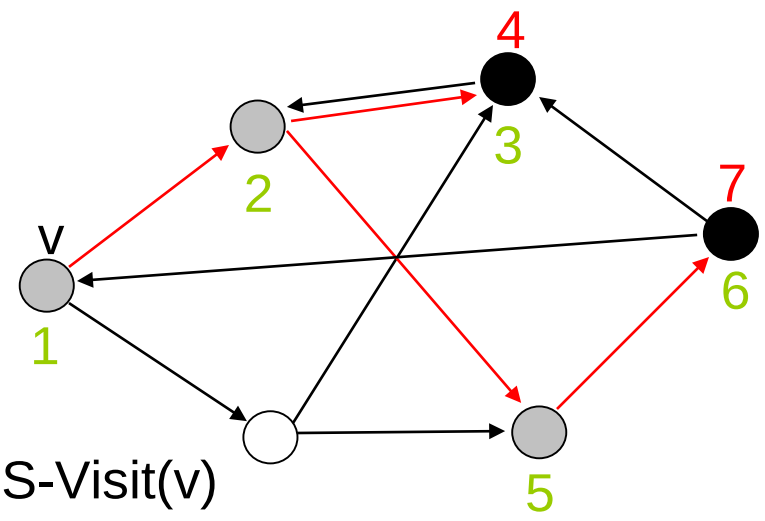
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=7



DFS-Visit(u)

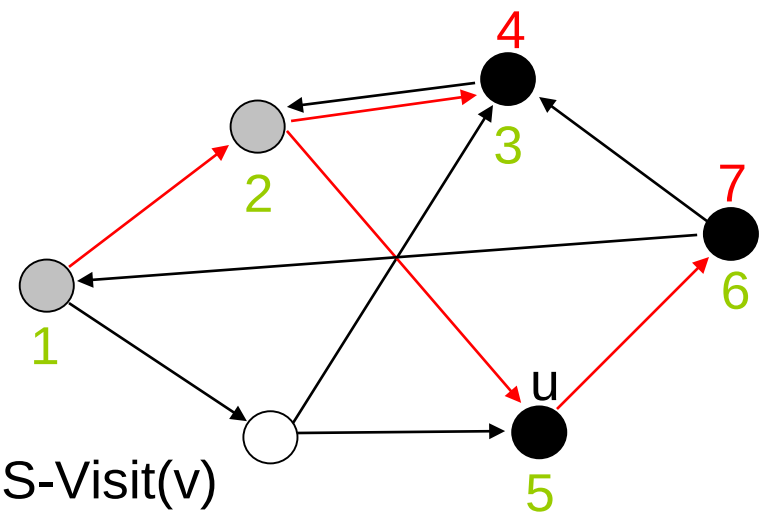
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=7



DFS-Visit(u)

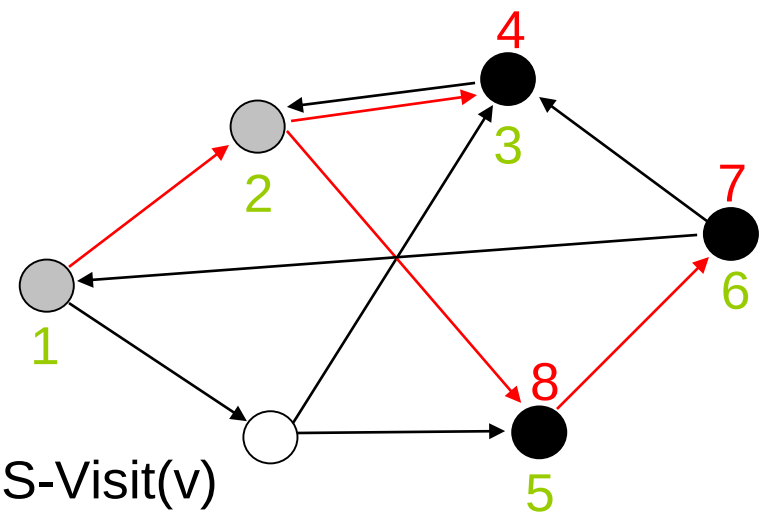
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=8



DFS-Visit(u)

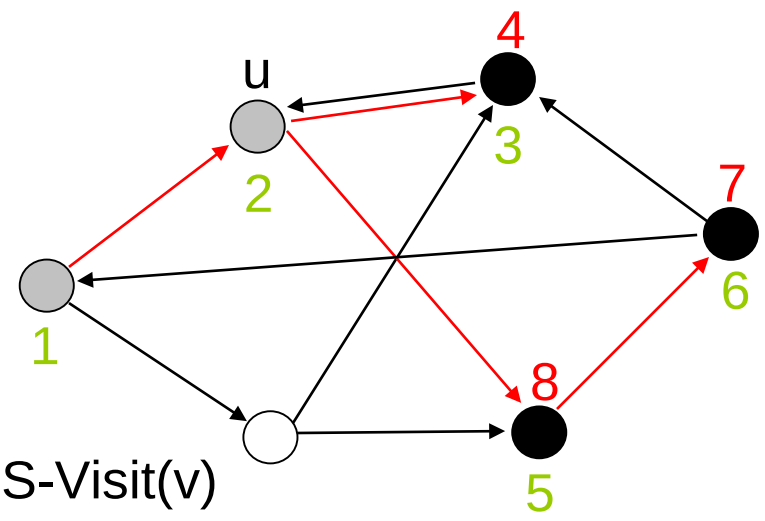
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=8



DFS-Visit(u)

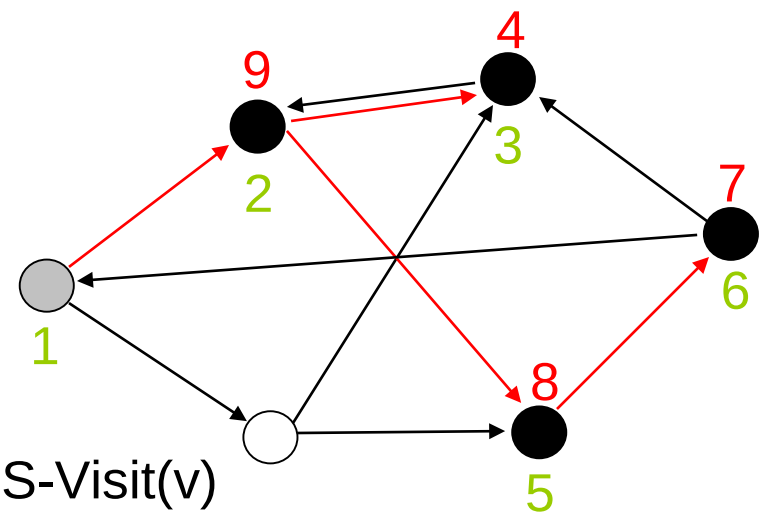
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=9



DFS-Visit(u)

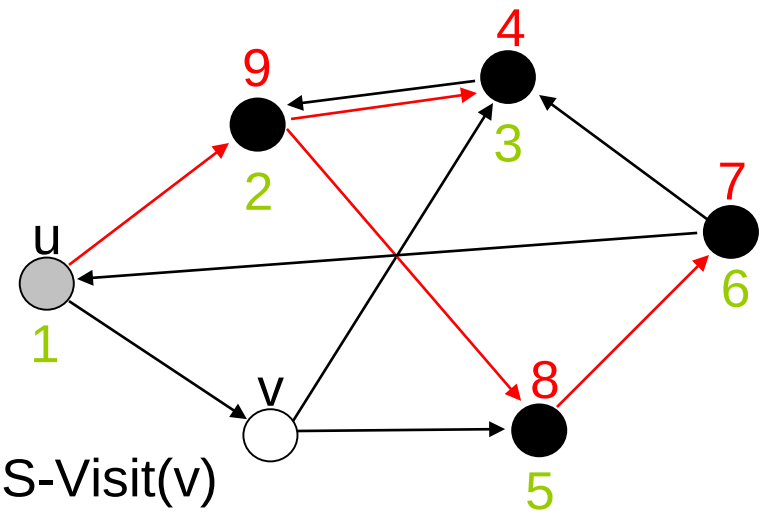
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=9



DFS-Visit(u)

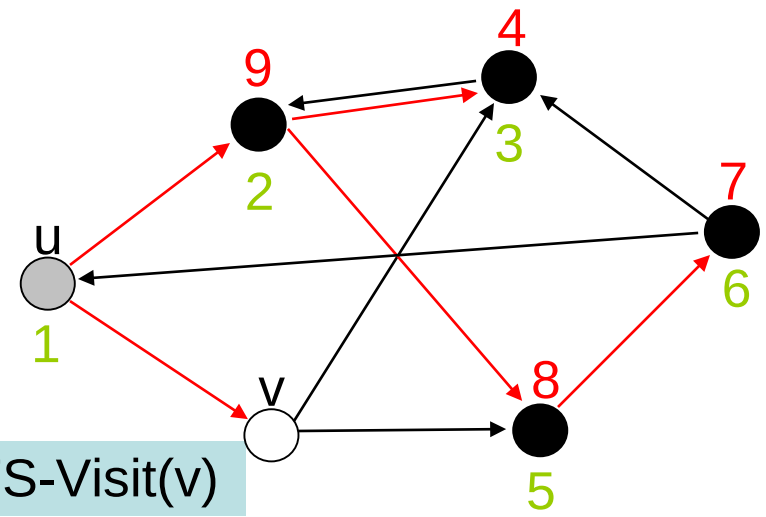
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=9



DFS-Visit(u)

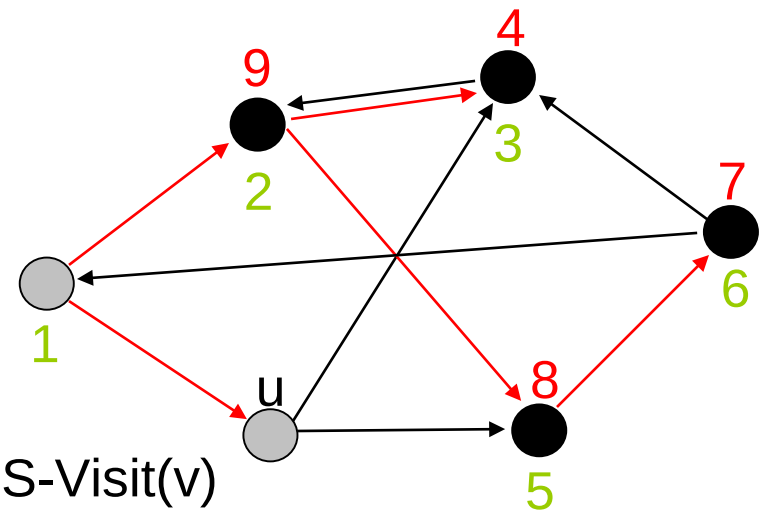
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=9



DFS-Visit(u)

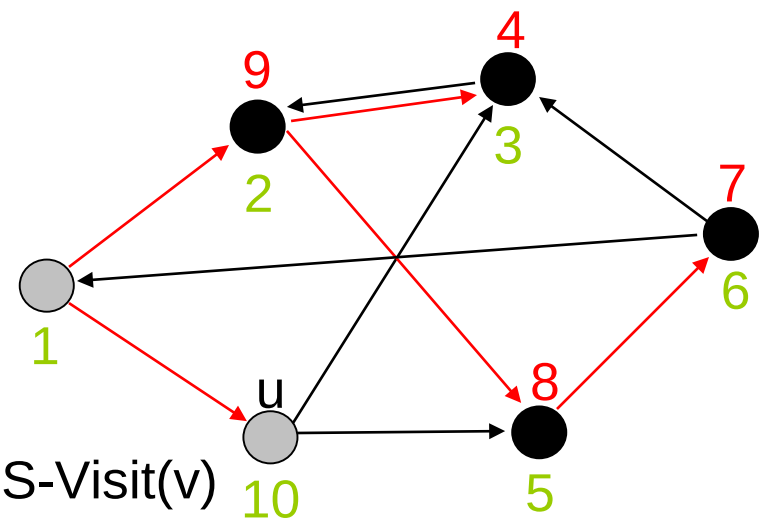
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

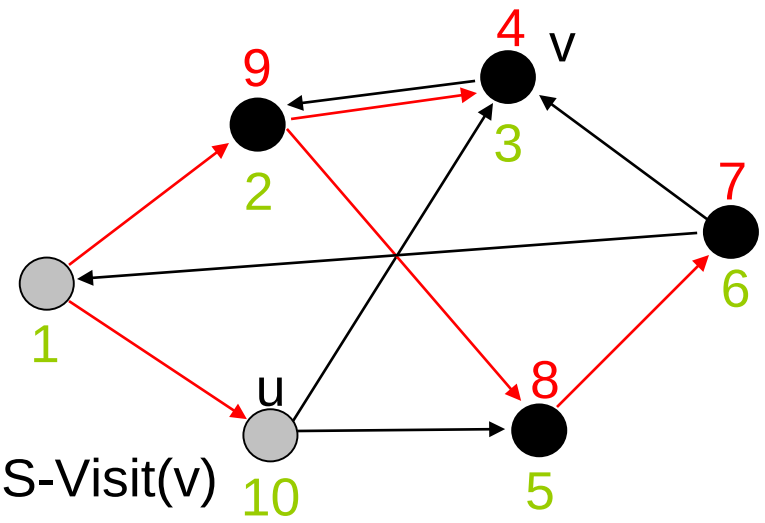
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

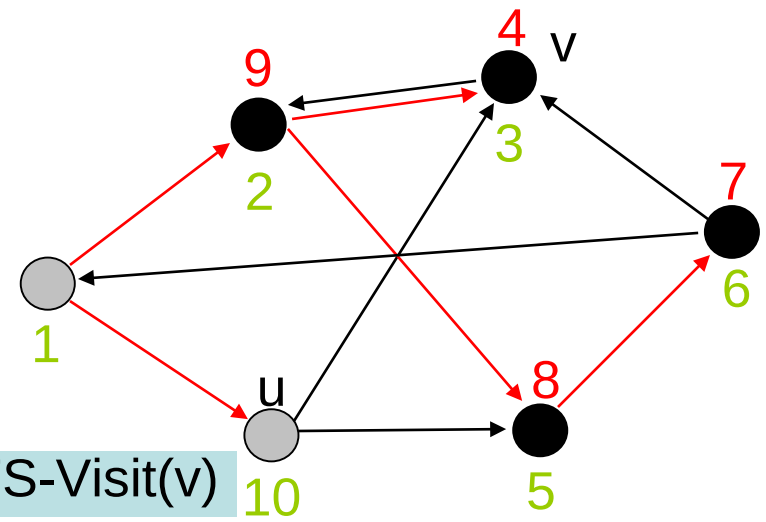
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

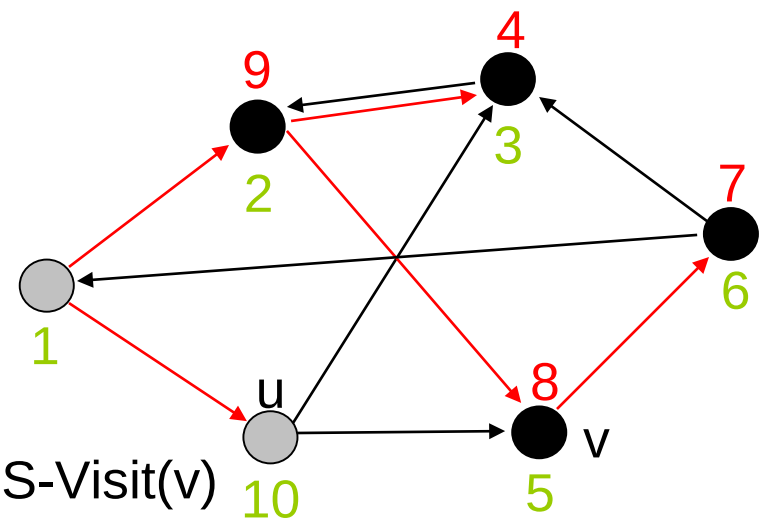
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

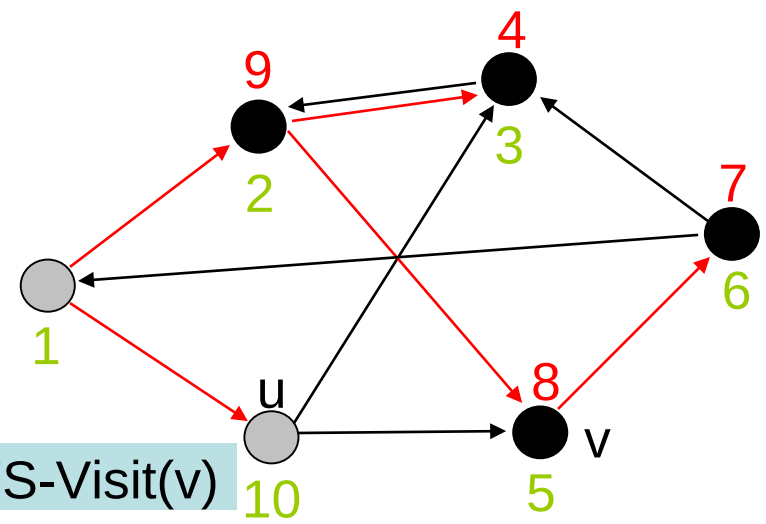
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

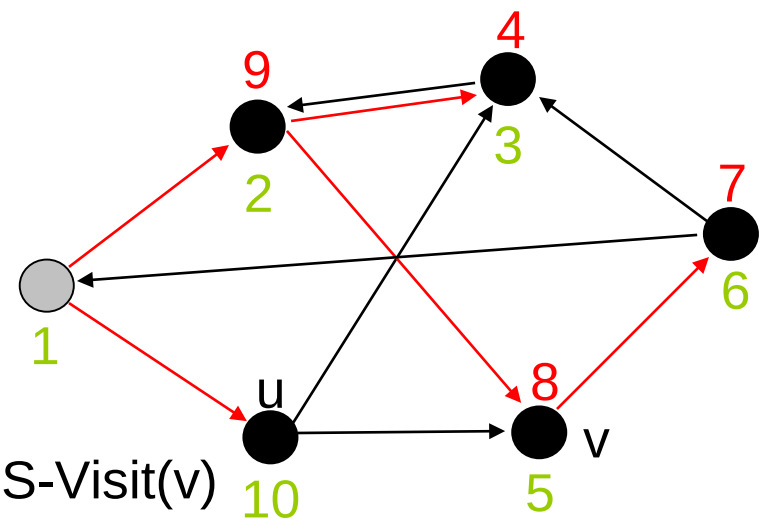
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=10



DFS-Visit(u)

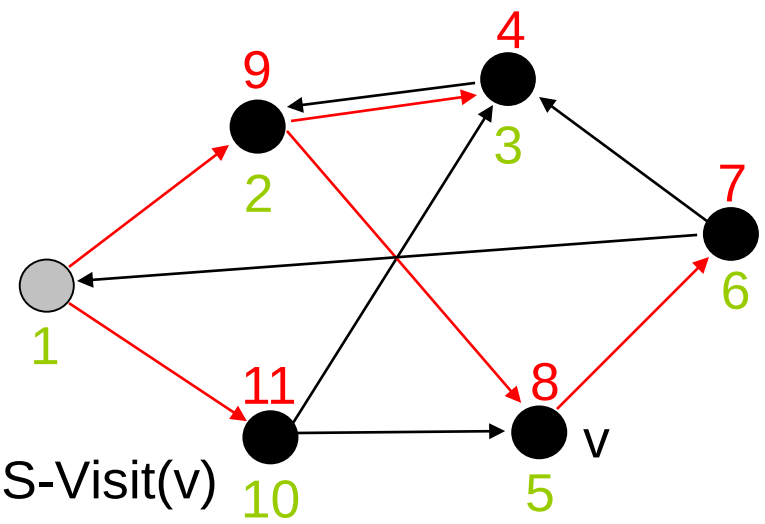
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=11



DFS-Visit(u)

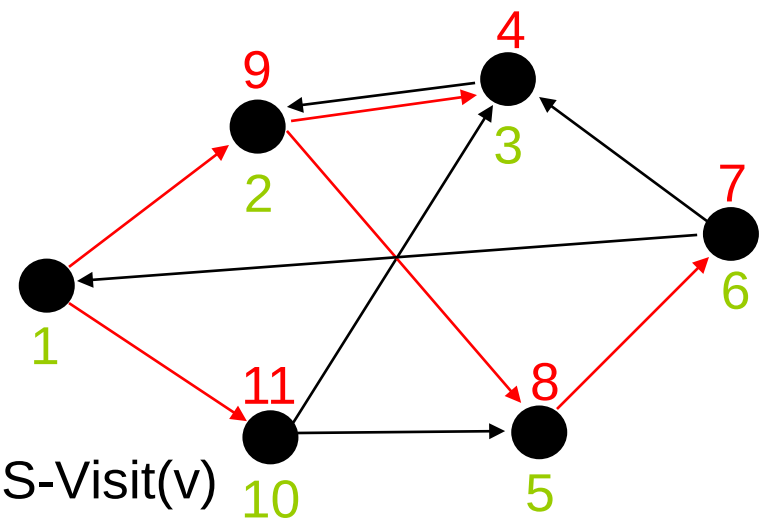
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=11



DFS-Visit(u)

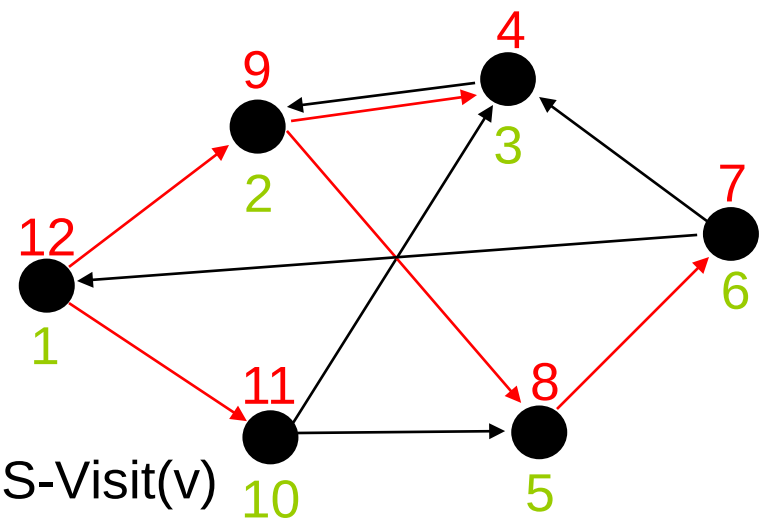
1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

time=12



DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

Tiefensuche

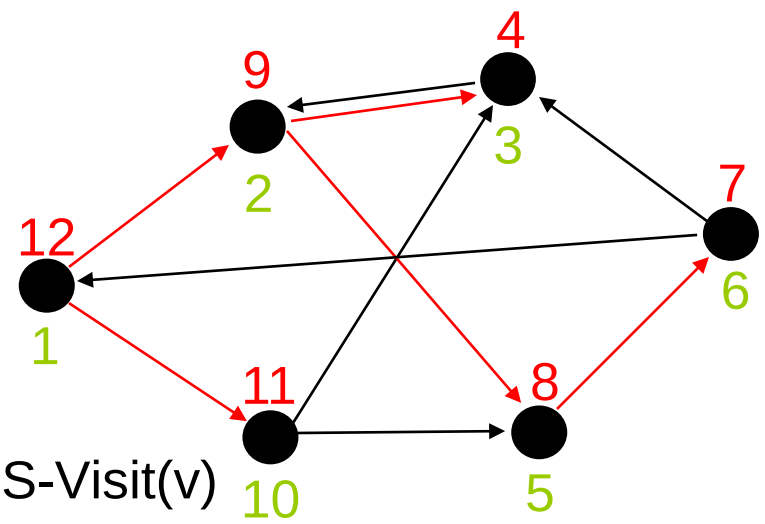
DFS(G)

1. **for each** vertex u∈V **do** color[u] ← weiß ; π[u] ← nil
2. time ← 0
3. **for each** vertex u∈V **do**
4. **if** color[u]=weiß **then** DFS-Visit(u)

DFS-Visit(u)

1. color[u] ← grau
2. time ← time +1; d[u] ← time
4. **for each** v∈Adj[u] **do**
5. **if** color[v] = weiß **then** π[v] ← u ; DFS-Visit(v)
6. color[u] ← schwarz
7. time ← time+1; f[u] ← time

time=12



Laufzeit:
O(|V|+|E|)

Tiefensuche

DFS-Baum: Baum der rekursiven Aufrufe (bzw. der Kanten (π(u),u)) im DFS Algorithmus.

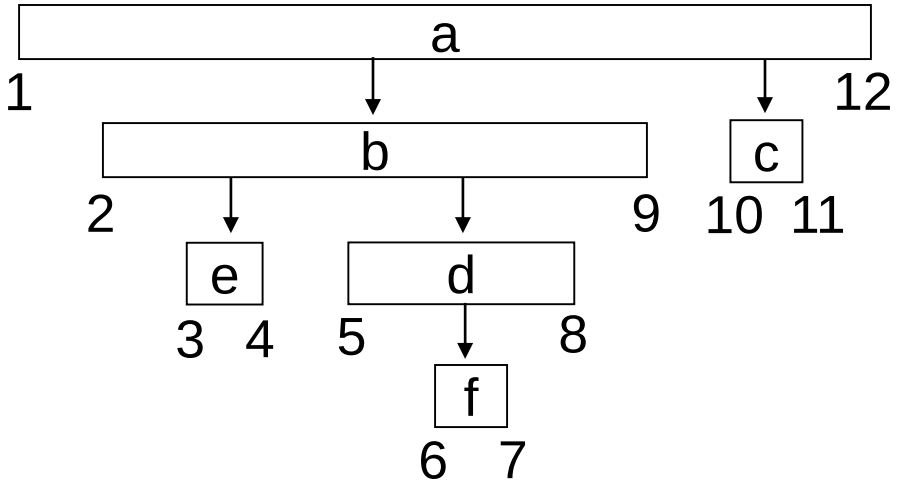
Satz 14.8 (Klammersatz zur Tiefensuche):

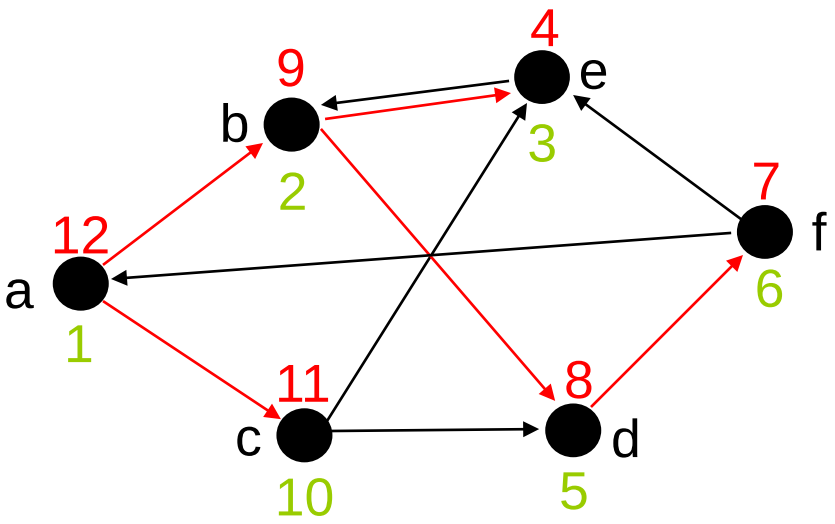
In jeder Tiefensuche eines gerichteten oder ungerichteten Graphen gilt für jeden Knoten u und v genau eine der folgenden drei Bedingungen:

- Die Intervalle [d[u],f[u]] und [d[v],f[v]] sind vollständig disjunkt
- Intervall [d[u],f[u]] ist vollständig im Interval [d[v],f[v]] enthalten und u ist Nachfolger von v im DFS-Baum
- Intervall [d[v],f[v]] ist vollständig im Interval [d[u],f[u]] enthalten und v ist Nachfolger von u im DFS-Baum

Tiefensuche

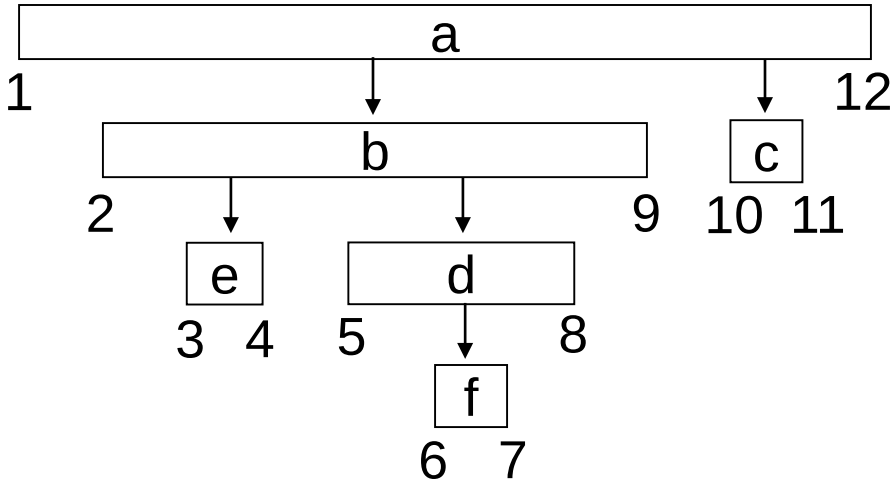
Beispiel:

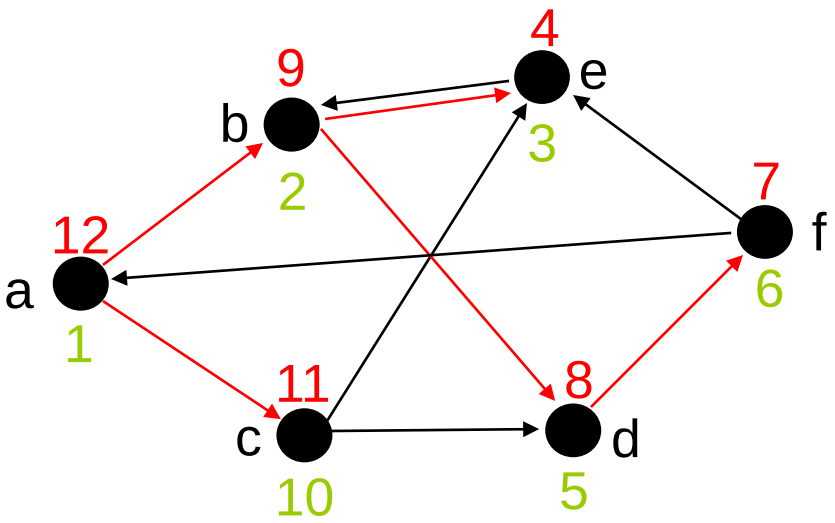




Tiefensuche

Beispiel:



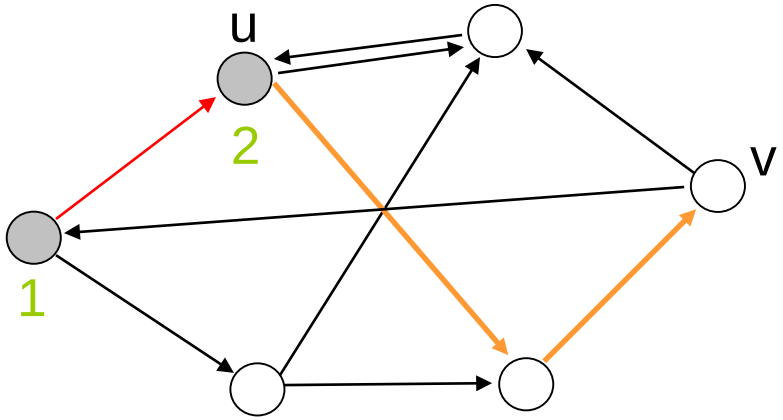


Korollar 14.9:

Knoten v ist echter Nachfolger von Knoten u im DFS-Baum von G, gdw. d[u]<d[v]<f[v]<f[u].

Satz 14.10 (Satz vom weißen Weg)

In einem DFS-Wald eines gerichteten oder ungerichteten Graphen G ist Knoten v ein Nachfolger von Knoten u gdw. zum Zeitpunkt d[u] v über einen Pfad weißer Knoten erreicht werden kann.



Tiefensuche - Laufzeitanalyse

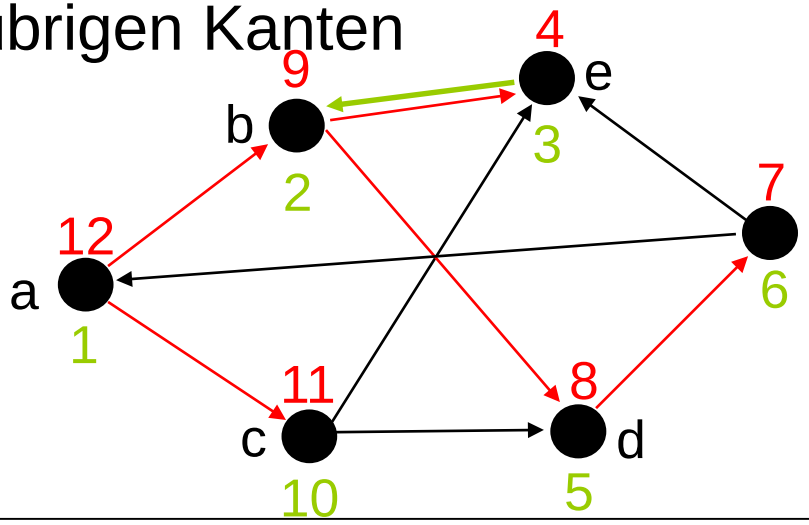
Satz 14.11: Bei Eingabe von Graph G=(V,E) besitzt Algorithmus DFS Laufzeit $O(|V| + |E|)$.

Analyse: Wie bei Breitensuche. Nutzen aus dass Gesamtgröße aller Adjazenzlisten $O(|E|)$

Tiefensuche

Klassifikation von Kanten:

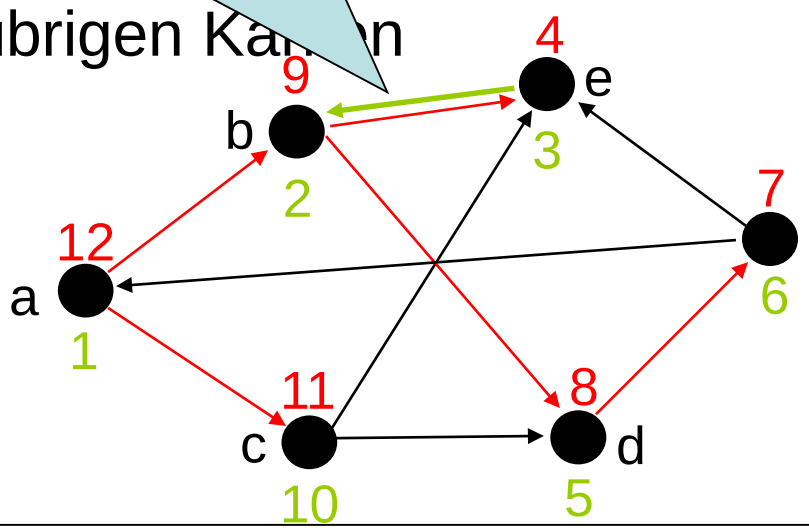
- Baumkanten sind Kanten des DFS-Walds G
- Rückwartskanten sind Kanten (u,v), die Knoten u mit Ahnen von u im DFS-Baum verbinden
- Vorwärtskanten sind die nicht-Baum Kanten (u,v), die u mit einem Nachfolger v in einem DFS-Baum verbinden
- Kreuzungskanten sind alle übrigen Kanten



Tiefensuche

Klassifikation von Kanten:

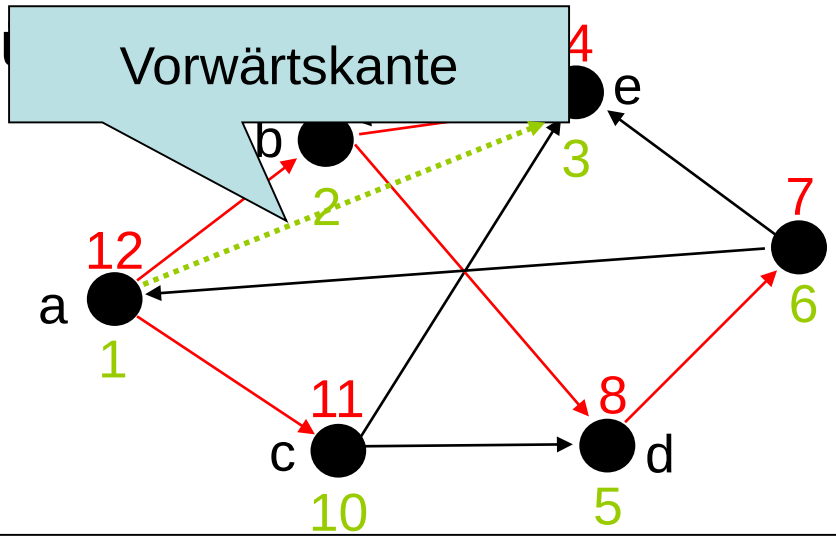
- Baumkanten sind Kanten des DFS-Walds G
- Rückwartskanten sind Kanten (u,v), die Knoten u mit Ahnen von u im DFS-Baum verbinden
- Vorwärtskanten sind die nicht-Baum Kanten (u,v), die u mit einem Nachfolger v in einem [illegible]len
- Kreuzungskanten sind alle übrigen Ka[illegible]n



Tiefensuche

Klassifikation von Kanten:

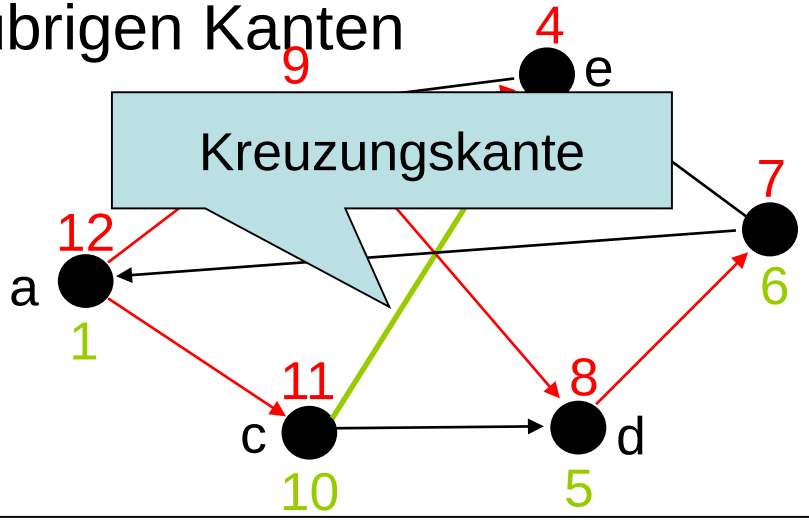
- Baumkanten sind Kanten des DFS-Walds G
- Rückwartskanten sind Kanten (u,v), die Knoten u mit Ahnen von u im DFS-Baum verbinden
- Vorwärtskanten sind die nicht-Baum Kanten (u,v), die u mit einem Nachfolger v in einem DFS-Baum verbinden
- Kreuzungskanten sind alle



Tiefensuche

Klassifikation von Kanten:

- Baumkanten sind Kanten des DFS-Walds G
- Rückwartskanten sind Kanten (u,v), die Knoten u mit Ahnen von u im DFS-Baum verbinden
- Vorwärtskanten sind die nicht-Baum Kanten (u,v), die u mit einem Nachfolger v in einem DFS-Baum verbinden
- Kreuzungskanten sind alle übrigen Kanten



Tiefensuche

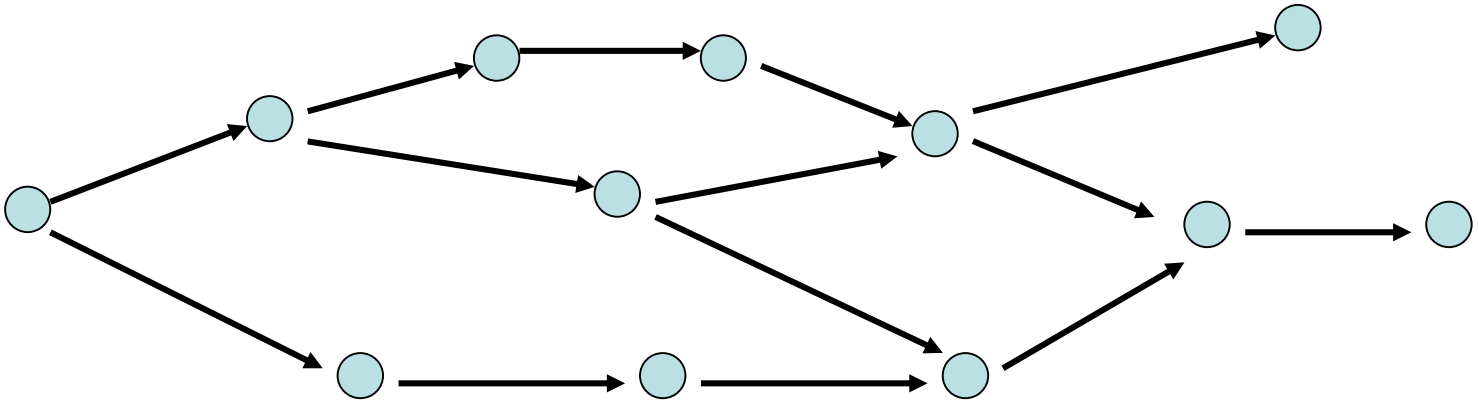
Beobachtung für Kante (v,w):

Kantentyp	d[v]< d[w]	f[v]>f[w]
Baum & Vorwärts	Ja	Ja
Rückwärts	Nein	Nein
Kreuz	Nein	Ja

Tiefensuche

Anwendung:

- Erkennung eines azyklischen gerichteten Graphen (engl. DAG)



Merkmal: keine gerichtete Kreise

Tiefensuche

Lemma 14.12: Das Folgende ist äquivalent:

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall (v,w) \in E: f[v] > f[w]$

Beweis:

2 $\Leftrightarrow$ 3: folgt aus Tabelle

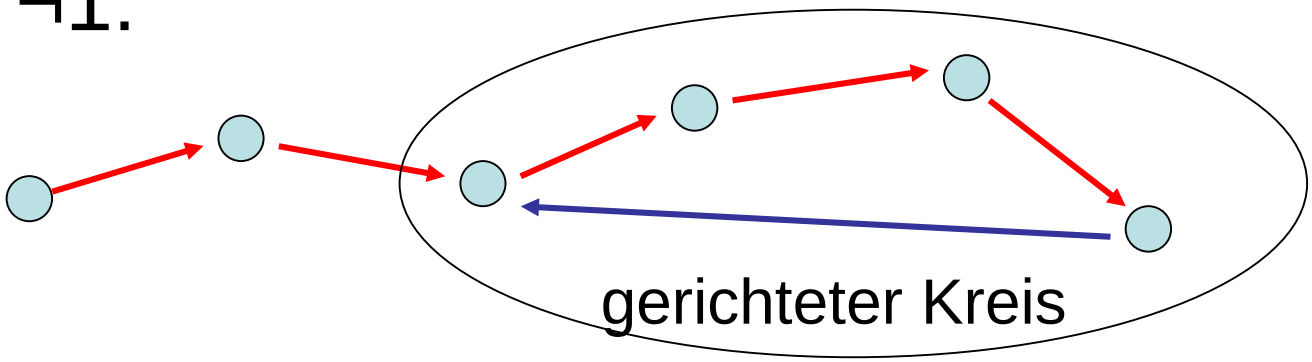
Tiefensuche

Lemma 14.12: Das Folgende ist äquivalent:

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall(v,w)\in E: f[v]>f[w]$

Beweis:

$\neg 2 \Rightarrow \neg 1$:



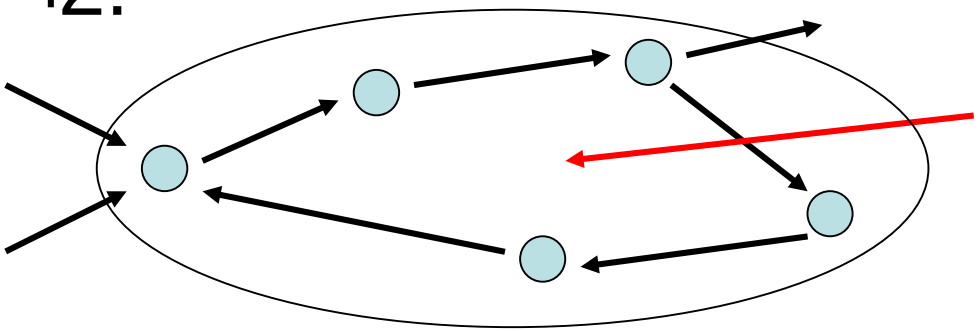
Tiefensuche

Lemma 14.12: Das Folgende ist äquivalent:

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall (v,w) \in E: f[v] > f[w]$

Beweis:

¬1 ⇒ ¬2:



Eine davon Rückwärtskante

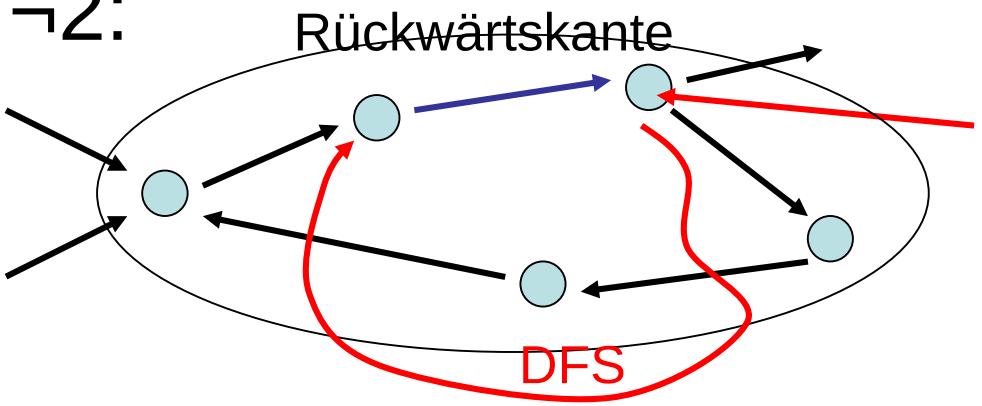
Tiefensuche

Lemma 14.12: Das Folgende ist äquivalent:

1. G ist ein DAG
2. DFS enthält keine Rückwärtskante
3. $\forall (v,w) \in E: f[v] > f[w]$

Beweis:

$\neg 1 \Rightarrow \neg 2$:



Annahme: Erster von DFS besuchter Knoten im Kreis

Zusammenfassung

- Breitensuche: Zunächst alle Nachbarn abarbeiten; liefert kürzeste Wege in (ungewichteten) Graphen. Laufzeit O(|V|+|E|).

- Tiefensuche: Zunächst in die „Tiefe“ gehen; Durchläuft alle Knoten eines Graphen in Zeit O(|V|+|E|).