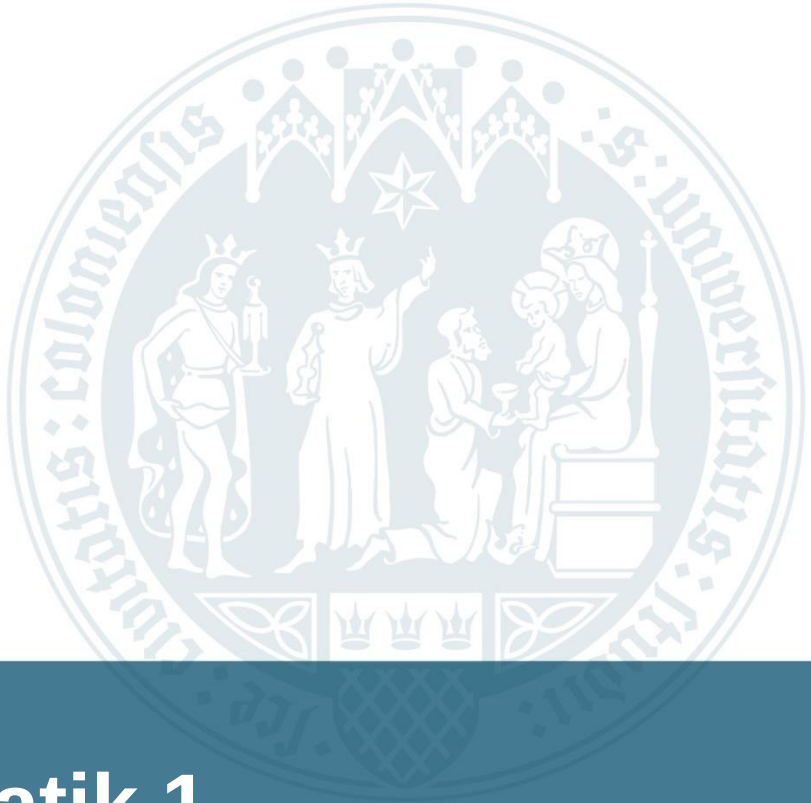
Universität zu Köln



Grundzüge der Informatik 1

Vorlesung 20

Überblick Vorlesung

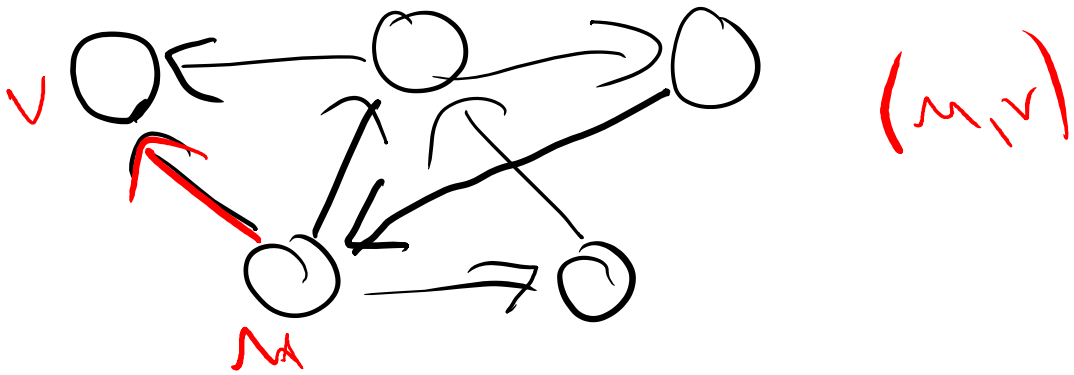
Graphenalgorithmen

- Wiederholung:
  - Repräsentation von Graphen
- Kürzeste Wege in ungewichteten Graphen (Breitensuche)

Graphalgorithmen

Definition (gerichteter Graph)

- Ein *gerichteter Graph* ist ein Paar (V,E), wobei V eine endliche Menge ist und $E \subseteq V \times V$.
- V heißt Knotenmenge des Graphen
- Die Elemente aus V sind die Knoten des Graphen
- E heißt Kantenmenge des Graphen
- Die Elemente aus E sind die Kanten des Graphen



Graphalgorithmen

Definition (ungerichteter Graph)

- Ein *ungerichteter Graph* ist ein Paar (V,E), wobei V eine endliche Menge ist und E Teilmenge der Menge aller Paare von Elementen aus V ist
- V heißt Knotenmenge des Graphen
- Die Elemente aus V sind die Knoten des Graphen
- E heißt Kantenmenge des Graphen
- Die Elemente aus E sind die Kanten des Graphen
- Wir stellen Kanten aus V wie im gerichteten Fall durch (u,v) dar und nehmen an, dass die Kante (u,v) gleich der Kante (v,u) ist
- Manchmal repräsentieren wir einen ungerichteten Graph durch einen gerichteten, indem wir jede Kante (u,v) durch die gerichteten Kanten (u,v) und (v,u) ersetzen

Graphalgorithmen

Datenstrukturen zur Repräsentation eines Graphen

- Adjazenzlisten: Dünn besetzen Graphen ($|E| << n^2$)
- Adjazenzmatrix: Dicht besetzte Graphen ($|E|$ nah an n^2)

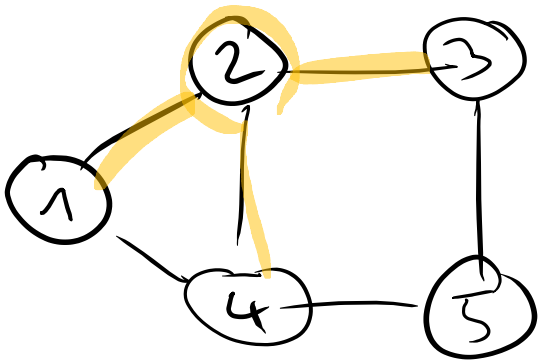
Arten von Graphen

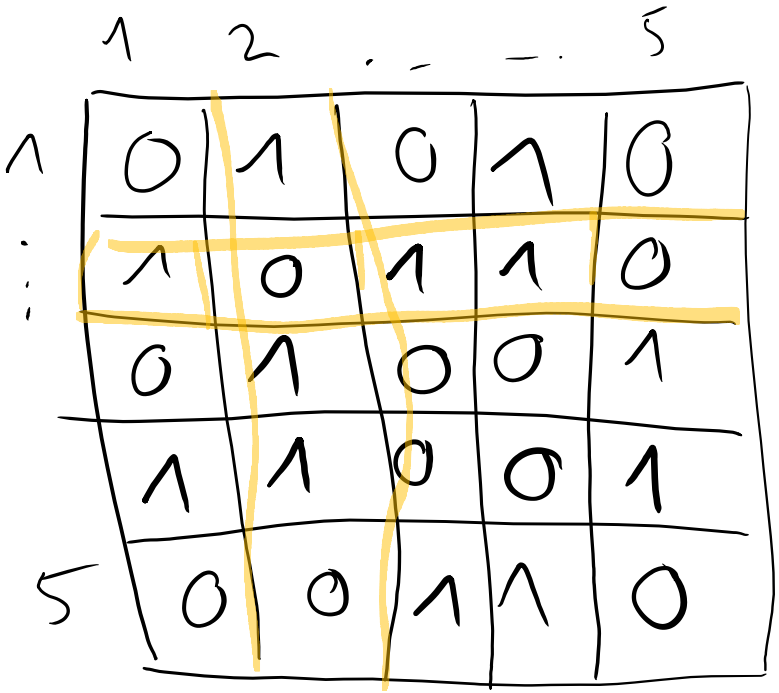
- Ungerichtet, gerichtet
- Ungewichtet, gewichtet (Knoten und/oder Kanten haben Gewichte)

Graphalgorithmen

Adjazenzmatrixdarstellung

- Knoten sind nummeriert von 1 bis |V|
- |V|×|V| Matrix A = (a_{ij}) mit
- a_{ij} = 1, wenn (i,j)∈E und a_{ij} = 0, sonst
- Bei ungerichteten Graphen gilt A = A^T

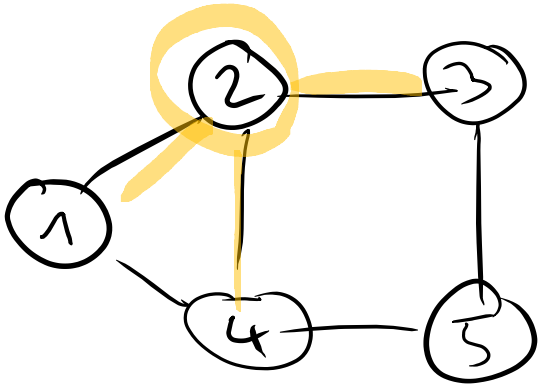


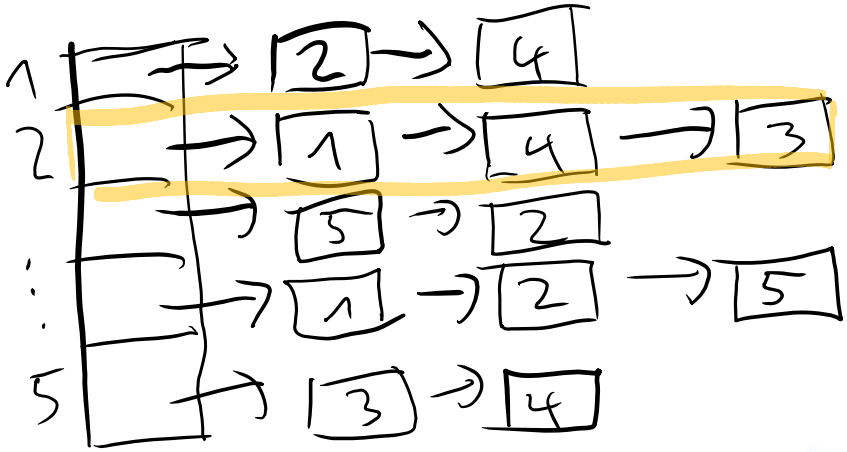


Graphenalgorithmen

Adjazenzlistendarstellung

- Feld Adj mit |V| Listen (eine pro Knoten)
- Für Knoten v enthält Adj[v] eine Liste aller Knoten u mit $(v,u) \in E$
- Die Knoten in Adj[v] heißen zu v benachbart
- Ist G ungerichtet, so gilt: $v \in Adj[u] \Leftrightarrow u \in Adj[v]$





Graphenalgorithmen

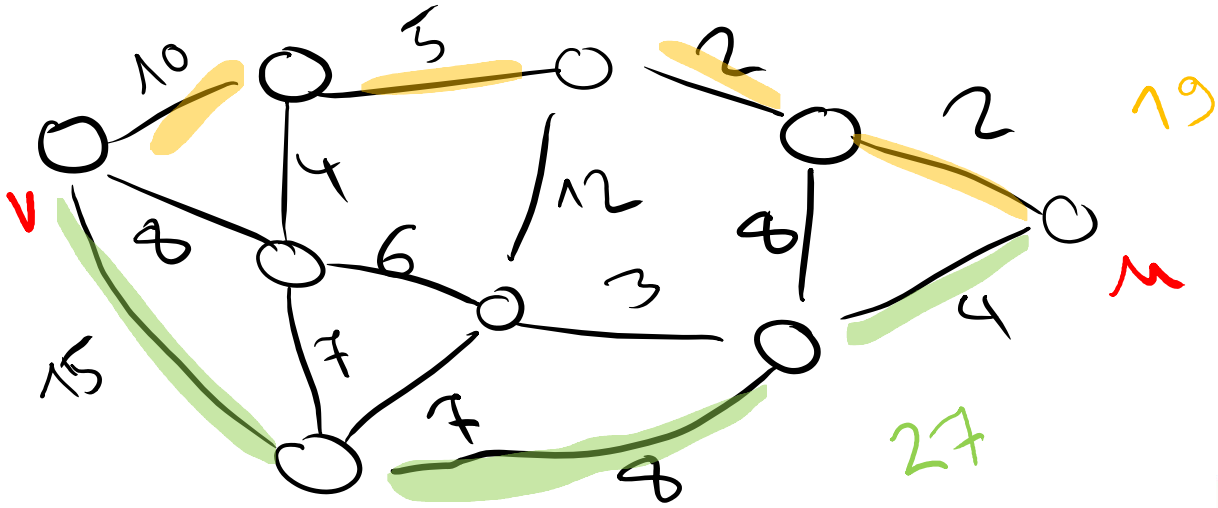
Graphen mit Kantengewichten

- Adjazenzmatrix: Gewicht einer Kante steht in der Adjazenzmatrix
- Adjazenzlisten: Gewicht w(u,v) von Kante (u,v) wird mit Knoten v in u‘s Adjazenzliste gespeichert

Graphenalgorithmen

Kürzeste Wege in Graphen

- Gegeben (möglicherweise gewichteter) Graph G=(V,E)
- Frage: Was ist der kürzeste Weg Knoten v nach Knoten u?
- Länge des Weges: Summe der Kantengewichte (bzw. Anzahl Kanten, wenn ungewichtet)



Graphenalgorithmen

Single Source Shortest Path (SSSP)

- Startknoten s
- Aufgabe: Berechne kürzeste Wege von s zu allen anderen Knoten

All Pairs Shortest Path (APSP)

- Aufgabe: Berechne kürzeste Wege zwischen allen Knotenpaaren

Graphenalgorithmen

SSSP in ungewichteten Graphen mit Breitensuche

- Graph in Adjazenzlistendarstellung
- Startknoten s
- Nutze Kanten von G, um alle Knoten zu finden, die von s aus erreichbar sind
- Finde kürzeste Distanz (Anzahl Kanten) zu jedem anderen Knoten

Idee

- Bearbeitet den Graphen „schichtweise“ nach Entfernung vom Startknoten: Besucht zuerst alle Knoten mit Entfernung 1; dann alle mit Entfernung 2; usw.

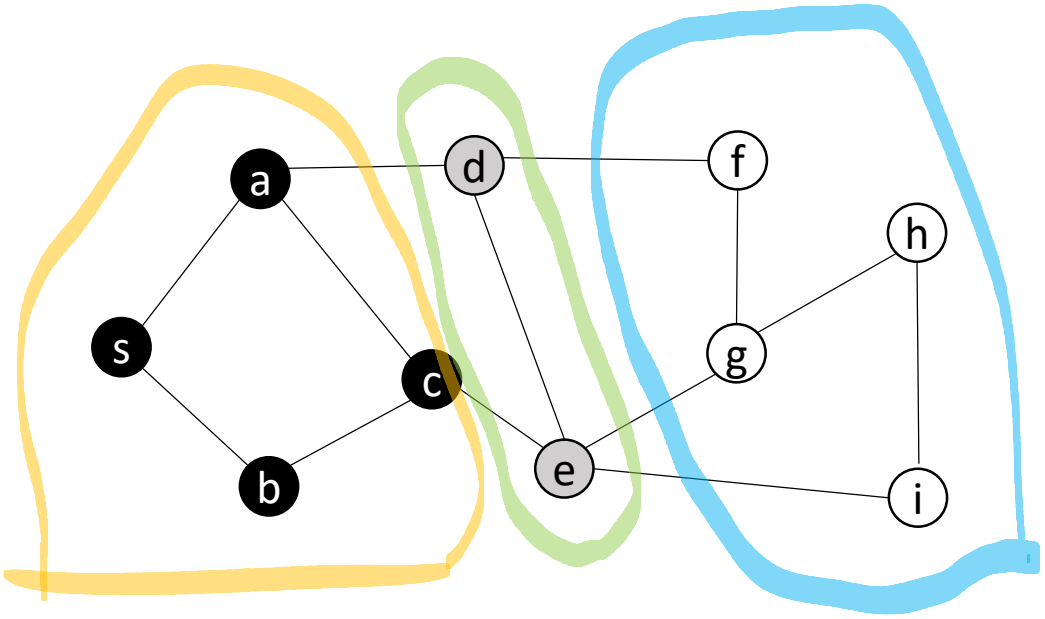
Graphenalgorithmen

Technische Invariante (Breitensuche)

- Knoten haben 3 Farben: weiß, grau und schwarz
- Zu Beginn: Alle Knoten sind weiß; Knoten s ist grau
- Ein nicht-weißer Knoten heißt „entdeckt“
- Unterscheidung grau-schwarz dient zur Steuerung des Algorithmus
- Wenn $(u,v) \in E$ ist und u ist schwarz, dann sind seine adjazenten Knoten grau oder schwarz
- Graue Knoten können adjazente weiße Knoten haben

Graphenalgorithmen

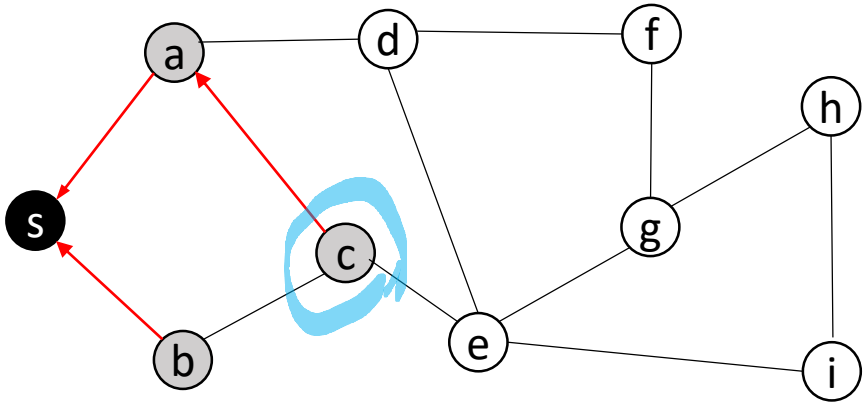
Beispiel Invariante



Graphenalgorithmen

Breitensuche

- Baut Breitensuche-Baum (BFS-Baum)
- Zu Beginn enthält der Baum nur die Wurzel, nämlich s
- Wenn weißer Knoten v beim Durchsuchen der Adjazenzliste eines bereits entdeckten Knotens u entdeckt wird, dann werden v und (u,v) dem Baum hinzugefügt
- u ist dann Vaterknoten von v



Graphenalgorithmen

Datenstruktur Schlange Q

- Operationen: head, enqueue, dequeue
- head(Q): Gibt Referenz auf das erste in der Schlange gespeicherte Element zurück
- enqueue(Q,x): Fügt neues Objekt x am Ende der Schlange ein
- dequeue(Q): Entfernt Objekt am Kopf der Schlange und gibt dieses zurück

Wir verwenden

- Doppelt verkettete Liste
- Zusätzlich halten wir Zeiger auf das letzte Element aufrecht
- Alle Operationen in O(1) Zeit

Graphenalgorithmen

d[u]: Abstand zu s (zu Beginn ∞)
π[u]: Vaterknoten von u (zu Beginn NIL)

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

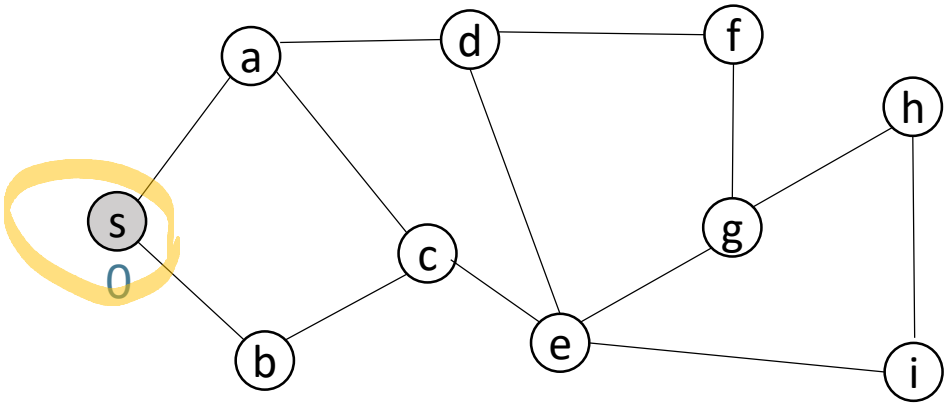
Für jeden Knoten u:
- color[u] = weiß
- d[u] = ∞
- π[u] = nil

Für Knoten s:
- color[s] = grau
- d[s]=0
- π[s]=nil
- s wird in Schlange Q eingefügt

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

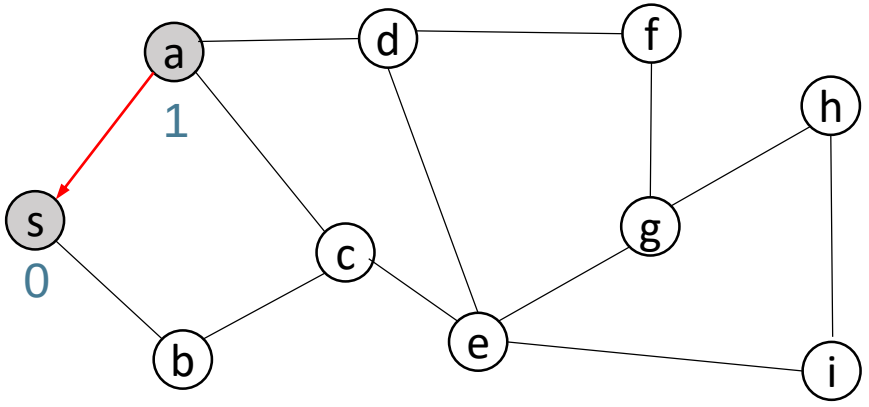


Q: s

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

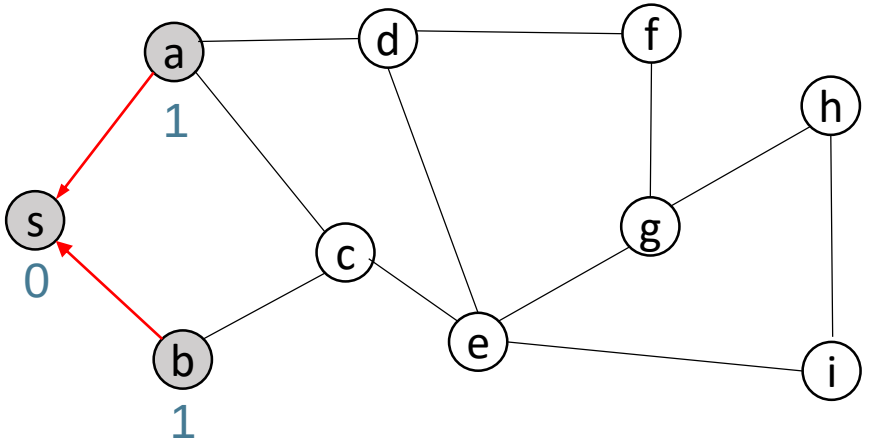


Q: s, a

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

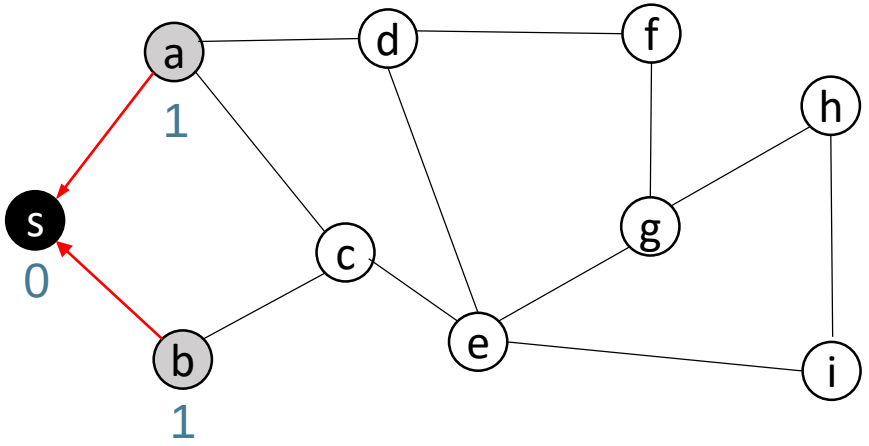


Q: s, a, b

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

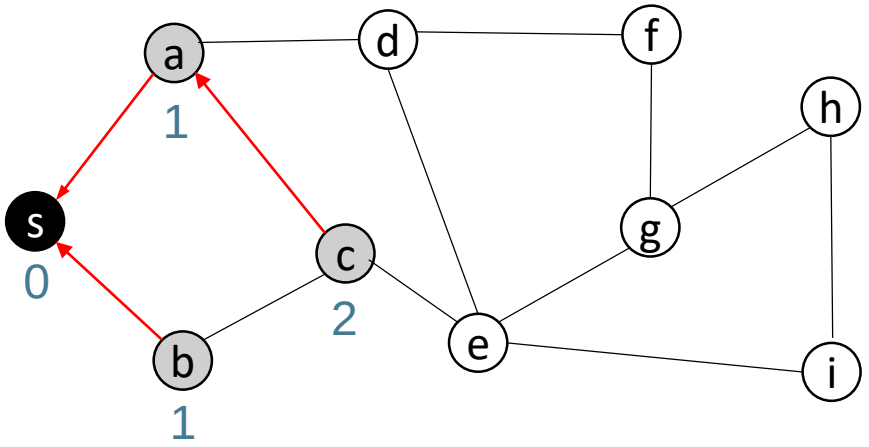


Q: a, b

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

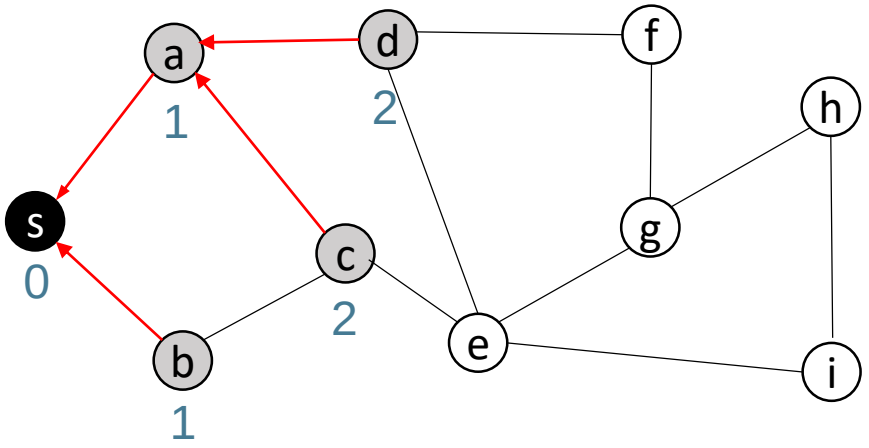


Q: a, b, c

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

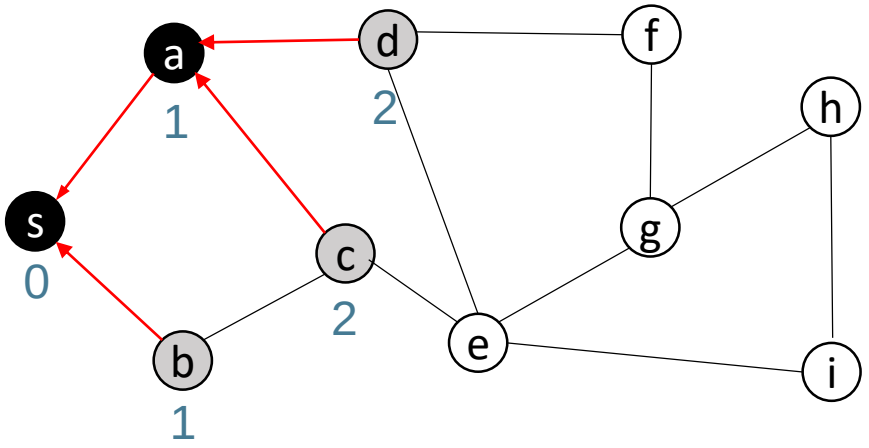


Q: a, b, c, d

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

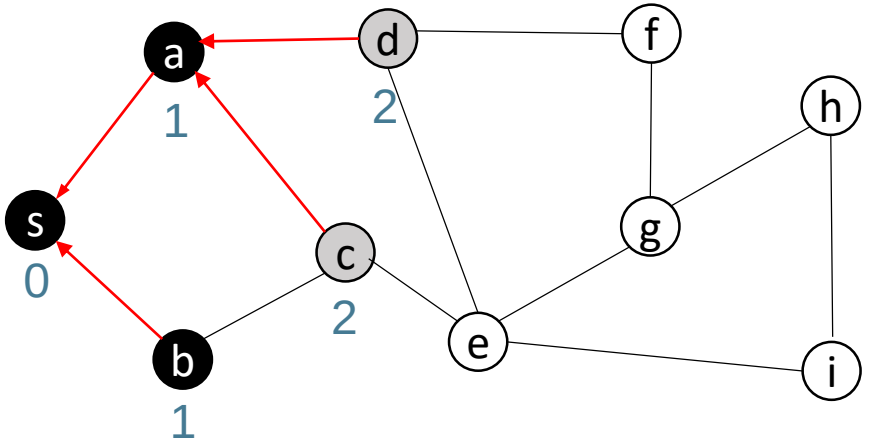


Q: b, c, d

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

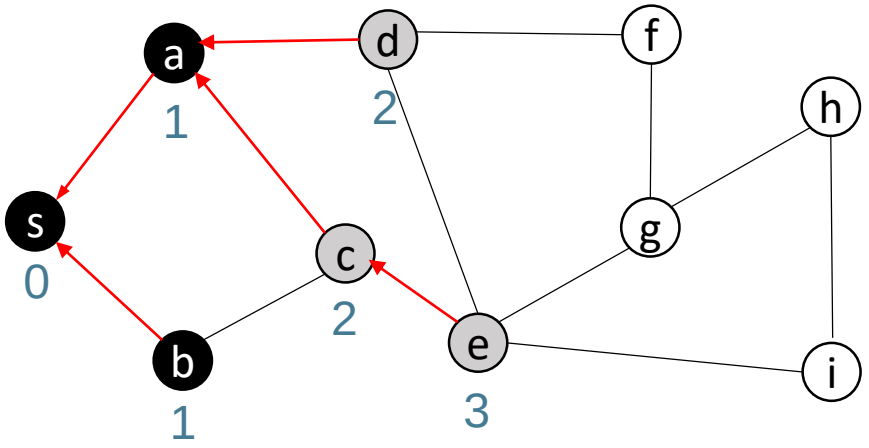


Q: c, d

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

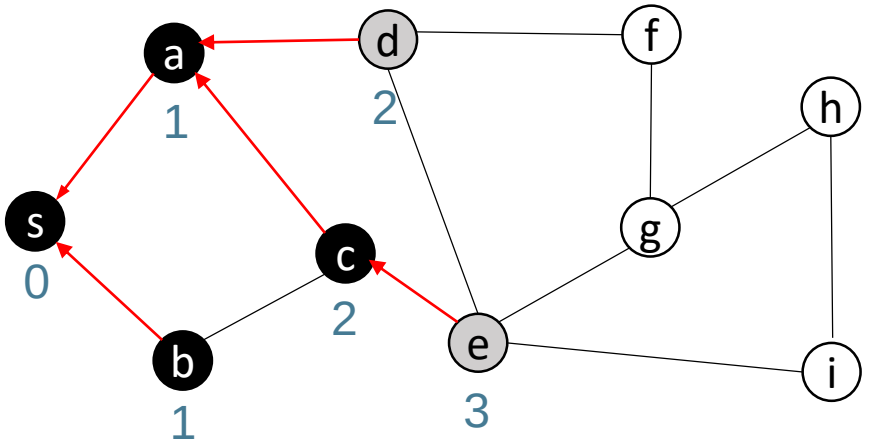


Q: c, d, e

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

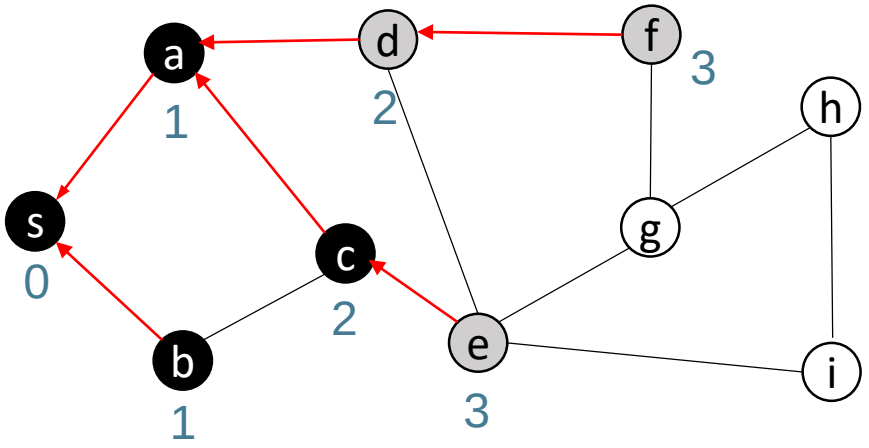


Q: d, e

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

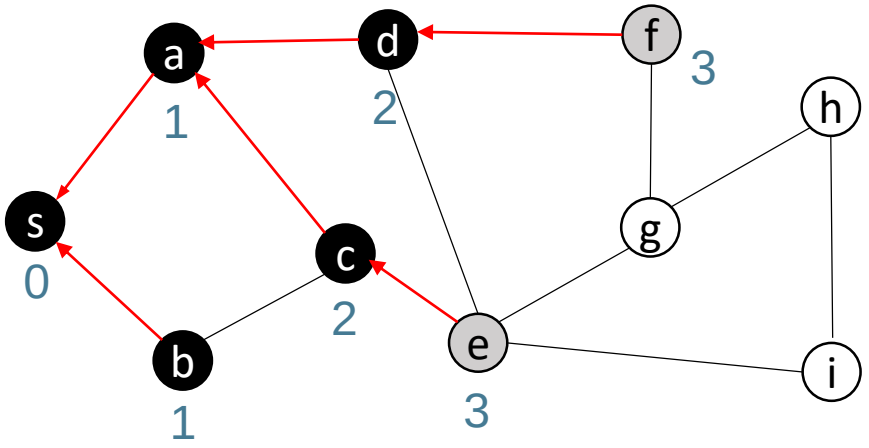


Q: d, e, f

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

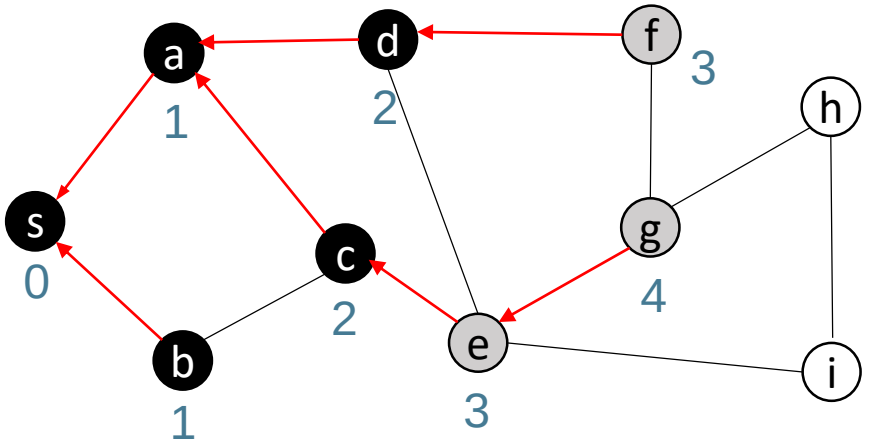


Q: e, f

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

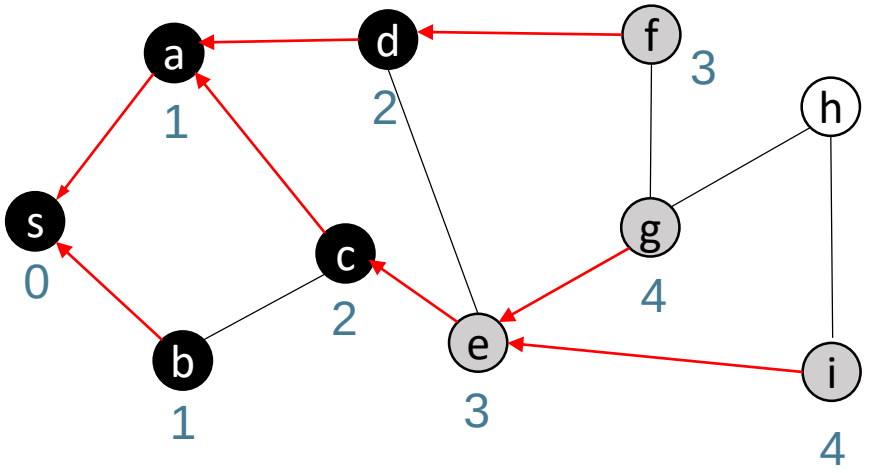


Q: e, f, g

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

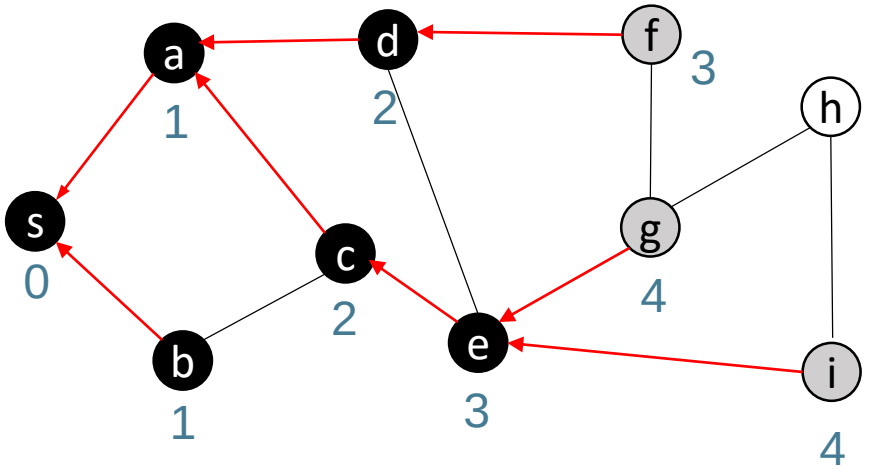


Q: e, f, g, i

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

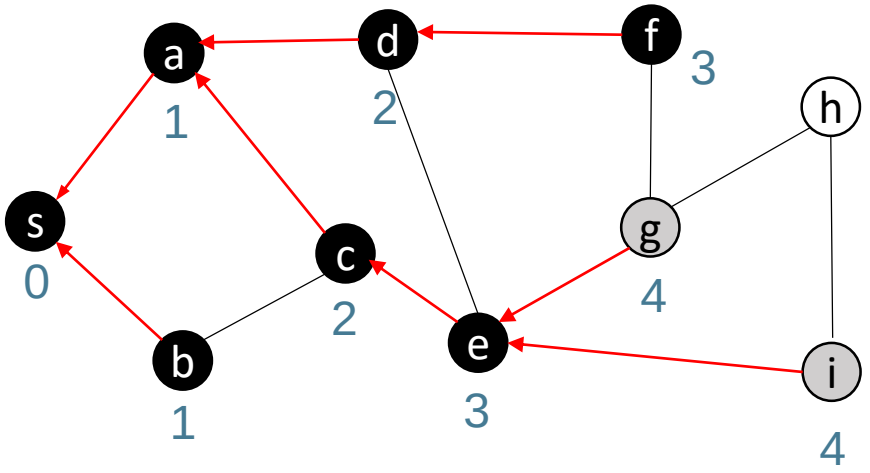


Q: f, g, i

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

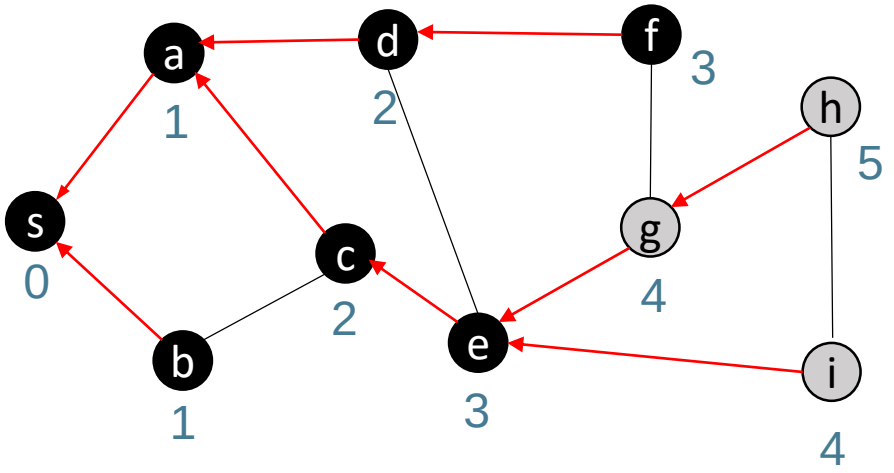


Q: g, i

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

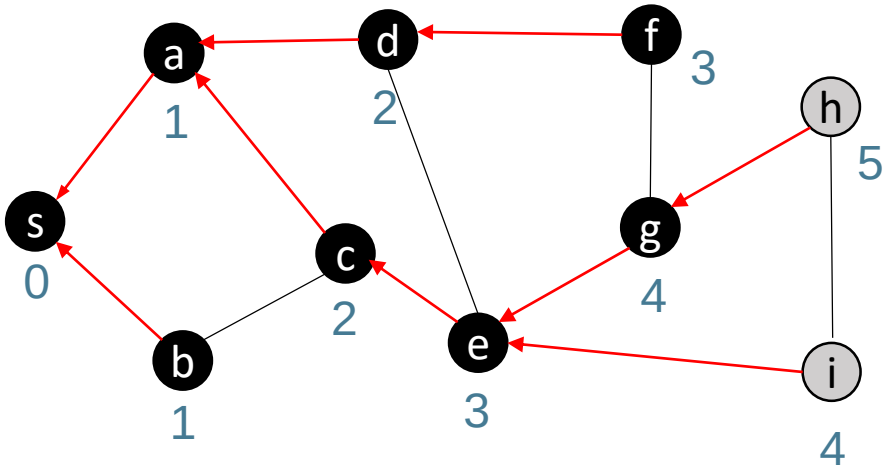


Q: g, i, h

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

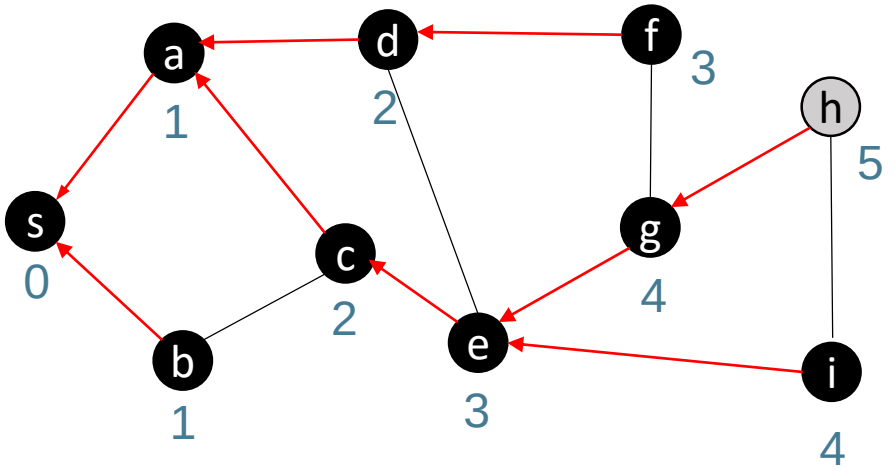


Q: i, h

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

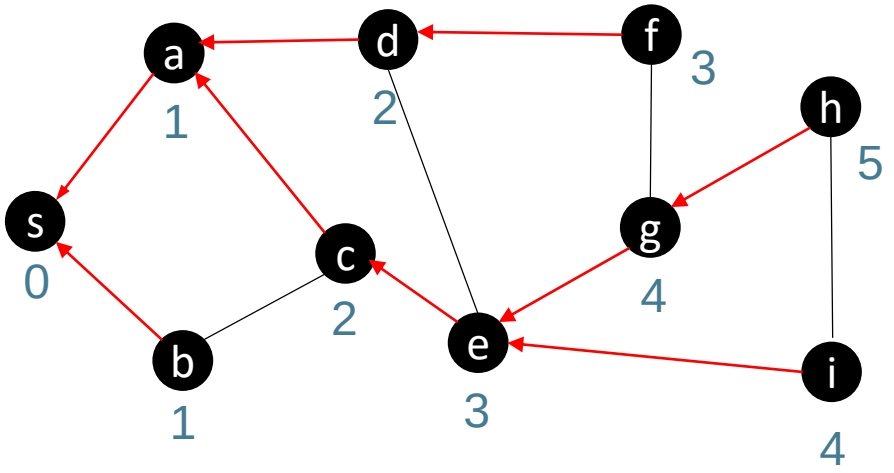


Q: h

Graphenalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** Q≠∅ **do**
3. u = head[Q]
4. **for** each v∈Adj[u] **do**
5. **if** color[v] = weiß **then**
6. color[v] = grau
7. d[v] = d[u]+1; π[v] = u
8. enqueue(Q,v)
9. dequeue(Q)
10. color[u] = schwarz

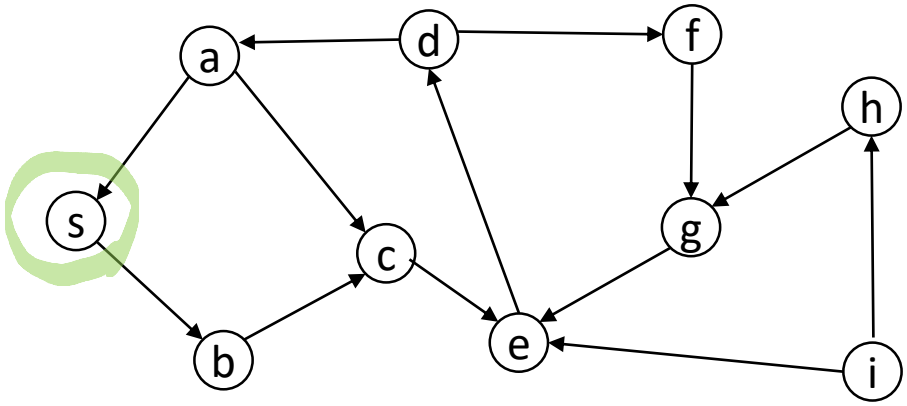


Q:

Graphenalgorithmen

Aufgabe

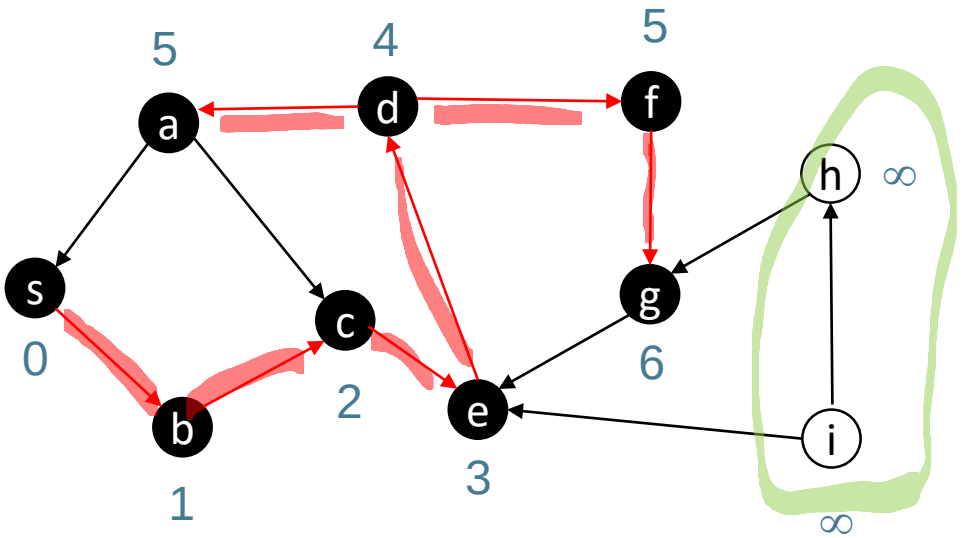
- Führen Sie eine Breitensuche auf folgendem gerichteten Graphen durch
- Bestimmen Sie insbesondere die d-Werte und den Breitensuche-Baum
- Was sind die d-Werte der Knoten g und h?



Graphenalgorithmen

Aufgabe

- Führen Sie eine Breitensuche auf folgendem gerichteten Graphen durch
- Bestimmen Sie insbesondere die d-Werte und den Breitensuche-Baum
- Was sind die d-Werte der Knoten g und h?



Graphenalgorithmen

Satz 20.1

- Sei G=(V,E) ein Graph. Die Laufzeit des Algorithmus BFS beträgt O(|V|+|E|).

Beweis

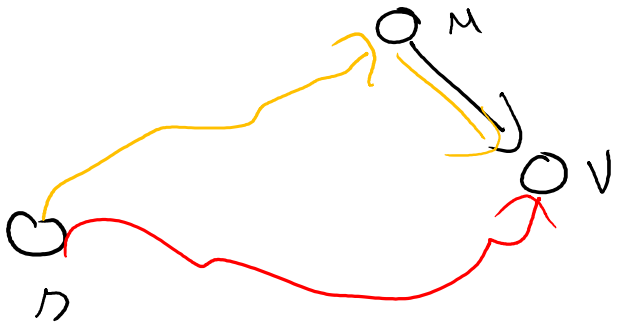
- Laufzeit Initialisierung: O(|V|)
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange
- $\Rightarrow$ Zeit für Schlangenoperationen ist O(|V|)
- Adjazenzliste jedes Knotens wird nur durchlaufen, wenn er aus der Schlange entfernt wird
- Damit wird jede Adjazenzliste maximal einmal durchlaufen (d.h. jede Kante maximal zweimal) $\Rightarrow$ Laufzeit für Listen: O(|V|+|E|)
- Gesamtlaufzeit: O(|V|+|E|)

Graphenalgorithmen

Kürzeste Wege in *ungewichteten* Graphen

- Ein s-t-Weg ist ein Weg mit Startknoten s und Endknoten t
- Sei $\delta(s,t)$ die minimale Anzahl Kanten in einem s-t-Weg
- Ein s-t-Weg der Länge $\delta(s,t)$ heißt *kürzester Weg*
- Wollen zeigen, dass BFS korrekt kürzeste Wege berechnet

Graphenalgorithmen



Lemma 20.2

- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei $s \in V$ ein beliebiger Knoten. Dann gilt für jede Kante $(u,v) \in E$: $\delta(s,v) \leq \delta(s,u)+1$.

Beweis

- Ist u erreichbar von s, dann ist es auch v
- Der kürzeste Weg von s nach v kann nicht länger sein, als der kürzeste Weg von s nach u gefolgt von der Kante (u,v). Damit gilt die Ungleichung.
- Ist u nicht erreichbar von s, dann ist $\delta(s,u)=\infty$ und die Ungleichung gilt.

Graphenalgorithmen

Lemma 20.3

- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und es laufe die Breitensuche von einem Startknoten $s \in V$. Während der Breitensuche gilt für jeden Knoten v, dass $d[v] \geq \delta(s,v)$ ist.

Beweis

- Induktion über Anzahl von Zeitpunkten, an denen ein Knoten mit enqueue in Q eingefügt wird
- Induktionsanfang: Nach Initialisierung gilt $d[s]=0=\delta(s,s)$ und $d[v]=\infty \geq \delta(s,v)$ für alle $v \in V-\{s\}$
- Induktionsannahme: Aussage gilt nach m enqueue Operationen
- Induktionsschluss: Betrachte nach m enqueue Operationen den nächsten weißen Knoten v, der während einer Suche von u entdeckt wird. Nach Induktionsannahme gilt $d[u] \geq \delta(s,u)$.

Graphenalgorithmen

Lemma 20.3

- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und es laufe die Breitensuche von einem Startknoten $s \in V$. Während der Breitensuche gilt für jeden Knoten v, dass $d[v] \geq \delta(s,v)$ ist.

Beweis

- Zeile 7: d[v] wird auf d[u]+1 gesetzt
- Es gilt: $d[v] = d[u]+1 \geq \delta(s,u)+1 \geq \delta(s,v)$ nach Lemma 20.2
- Knoten v wird dann in die Schlange eingefügt und grau gefärbt
- Damit ändert sich d[v] im Laufe des Algorithmus nicht mehr und die Aussage des Lemmas bleibt erhalten

Graphenalgorithmen

Lemma 20.4

- Sei $<v_1,\ldots, v_r>$ der Inhalt der Schlange Q während eines Durchlaufs der Breitensuche auf einem Graph G=(V,E), wobei v_1 Kopf und v_r Ende der Schlange ist. Dann gilt

$$d[v_r] \leq d[v_1]+1 \text{ und } d[v_i] \leq d[v_{i+1}] \text{ für } i=1,2,\ldots,r-1.$$

Beweis (Teil 1)

- Induktion über die Anzahl Schlangenoperationen dequeue und enqueue
- Induktionsanfang: Die Schlange enthält nur s, damit gilt das Lemma
- Induktionsannahme: Das Lemma gilt nach m Schlangenoperationen
- Induktionsschluss: Wir müssen zeigen, dass das Lemma immer noch nach m+1 Schlangenoperationen gilt. Die (m+1)ste Schlangenoperation ist entweder eine enqueue oder dequeue Operation

Graphenalgorithmen

Lemma 20.4

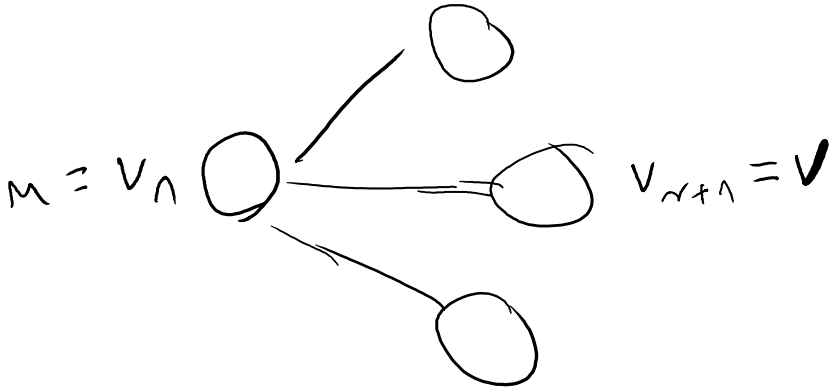
- Sei $<v_1,\ldots, v_r>$ der Inhalt der Schlange Q während eines Durchlaufs der Breitensuche auf einem Graph G=(V,E), wobei v_1 Kopf und v_r Ende der Schlange ist. Dann gilt

$$d[v_r] \leq d[v_1]+1 \text{ und } d[v_i] \leq d[v_{i+1}] \text{ für } i=1,2,\ldots,r-1.$$

Beweis (Teil 2)

- dequeue:
- Wird v_1 aus der Schlange entfernt, so wird v_2 der neue Kopf
- Dann gilt aber sicherlich $d[v_r] \leq d[v_1]+1 \leq d[v_2]+1$
- Alle anderen Ungleichungen sind nicht betroffen, also gilt das Lemma weiterhin

Graphenalgorithmen



Lemma 20.4

- Sei $\langle v_1,\ldots, v_r\rangle$ der Inhalt der Schlange Q während eines Durchlaufs der Breitensuche auf einem Graph G=(V,E), wobei v_1 Kopf und v_r Ende der Schlange ist. Dann gilt

$$d[v_r] \leq d[v_1]+1 \text{ und } d[v_i] \leq d[v_{i+1}] \text{ für } i=1,2,\ldots,r-1.$$

Beweis (Teil 3)

- enqueue:
- Wird in Zeile 8 ein Knoten v eingefügt (und damit zu v_{r+1}), so ist v_1 der Knoten u, von dem aus v entdeckt wurde
- Es gilt: $d[v_{r+1}] = d[v] = d[u]+1 = d[v_1]+1$
- Außerdem: $d[v_r] \leq d[v_1]+1 = d[u]+1 = d[v] = d[v_{r+1}]$
- Die anderen Ungleichungen bleiben erhalten; Also gilt das Lemma

Graphenalgorithmen

Satz 20.5

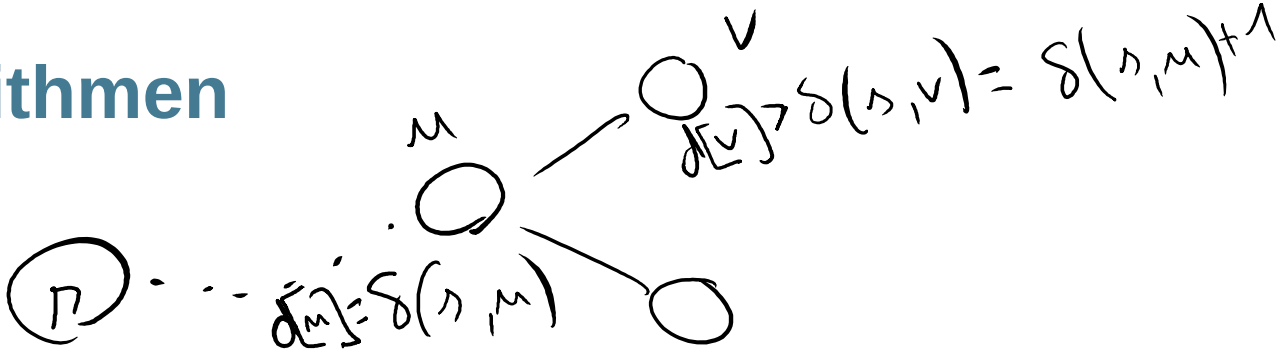
- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei $s \in V$ Startknoten der Breitensuche. Dann gilt:

  (1) nach Terminierung gilt $d[v]=\delta(s,v)$ für alle $v \in V$.

  (2) die Breitensuche entdeckt alle Knoten $v \in V$, die von s aus erreichbar sind

  (3) für jeden von s erreichbaren Knoten $v \neq s$ gilt, dass ein kürzester Weg von s nach $\pi[v]$ gefolgt von der Kante $(\pi[v],v)$ ein kürzester s-v-Weg ist.

Graphenalgorithmen



Satz 20.5

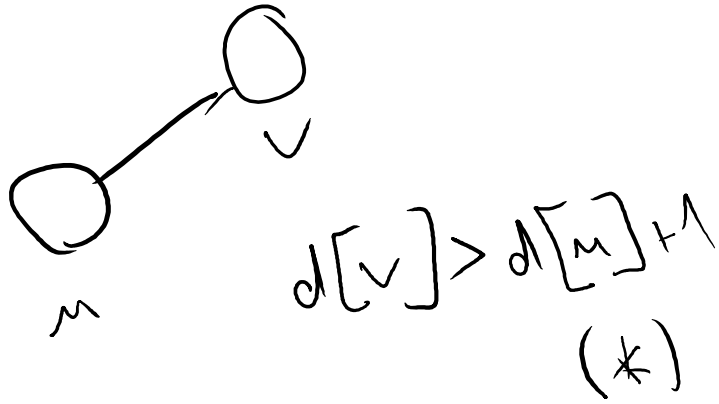
- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei s∈V Startknoten der Breitensuche. Dann gilt:

  (1) nach Terminierung gilt d[v]=δ(s,v) für alle v∈V.

Beweis (Teil 1 von (1))

- Annahme: Es gibt einen Knoten, der inkorrekten d-Wert erhält
- Sei v ein Knoten mit minimalem δ(s,v), der einen inkorrekten d-Wert erhält
- Nach Lemma 20.3 gilt: d[v]≥δ(s,v) und somit d[v]>δ(s,v) und v ist erreichbar von s
- Sei u der Vorgängerknoten von s auf einem kürzesten s-v-Weg, so dass δ(s,v)= δ(s,u) +1
- Nach unserer Wahl von v ist d[u]= δ(s,u)
- Somit gilt: d[v] > δ(s,v) = δ(s,u) +1 = d[u]+1 (*)

Graphenalgorithmen



Satz 20.5

- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei s∈V Startknoten der Breitensuche. Dann gilt:

  (1) nach Terminierung gilt d[v]=δ(s,v) für alle v∈V.

Beweis (Teil 2 von (1))

- Da u einen endlichen d-Wert hat, wurde u in die Schlange eingefügt
- Betrachte den Zeitpunkt, an dem u Kopf der Schlange wird
- Ist v zu diesem Zeitpunkt weiß, so wird d[v] = d[u]+1 gesetzt (Widerspruch zu (*))
- Ist v zu diesem Zeitpunkt schwarz, so war v vor u in der Schlange und wegen Lemma 20.4 und der Tatsache, dass der d-Wert nur einmal geändert wird, gilt: d[v]≤d[u] (Widerspruch zu (*))

Graphenalgorithmen

Satz 20.5

- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei $s \in V$ Startknoten der Breitensuche. Dann gilt:

  (1) nach Terminierung gilt $d[v]=\delta(s,v)$ für alle $v \in V$.

Beweis (Teil 3 von (1))

- Ist v zu diesem Zeitpunkt grau, so wurde er von einem Knoten w entdeckt, der vor u aus der Schlange entfernt wurde und es gilt $d[v] = d[w] +1$
- Aus Lemma 20.4 folgt außerdem $d[w] \leq d[u]$ und somit $d[v] \leq d[u]+1$
- Widerspruch zu (*)

Graphenalgorithmen

Satz 20.5

- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei $s \in V$ Startknoten der Breitensuche. Dann gilt:

  (1) nach Terminierung gilt $d[v]=\delta(s,v)$ für alle $v \in V$.

  (2) die Breitensuche entdeckt alle Knoten $v \in V$, die von s aus erreichbar sind

Beweis (von (2))

- (2) folgt aus (1) da jeder erreichbare Knoten einen endlichen d-Wert erhält
- Wenn dies zum ersten Mal passiert, wird der Knoten grau gefärbt und ist somit entdeckt

Graphenalgorithmen

Satz 20.5

- Sei G=(V,E) ein gerichteter oder ungerichteter Graph und sei $s \in V$ Startknoten der Breitensuche. Dann gilt:

  (1) nach Terminierung gilt $d[v]=\delta(s,v)$ für alle $v \in V$.

  (2) die Breitensuche entdeckt alle Knoten $v \in V$, die von s aus erreichbar sind

  (3) für jeden von s erreichbaren Knoten $v \neq s$ gilt, dass ein kürzester Weg von s nach $\pi[v]$ gefolgt von der Kante $(\pi[v],v)$ ein kürzester s-v-Weg ist.

Beweis (von (3))

- Wir beobachten, dass $\pi[v] = u$ impliziert, dass $d[v]=d[u]+1$
- Damit folgt, dass man einen kürzester s-v-Weg, indem man einen kürzesten s-$\pi[v]$ -Weg nimmt und dann der Kante $(\pi[v],v)$ folgt

Graphenalgorithmen

Zusammenfassung

- Breitensuche traversiert einen Graph in O(|V|+|E|) Zeit
- Die Breitensuche kann zur Berechnung der kürzesten Wege in ungewichteten Graphen verwendet werden

Referenzen

- T. Cormen, C. Leisserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT press. Second edition, 2001.