

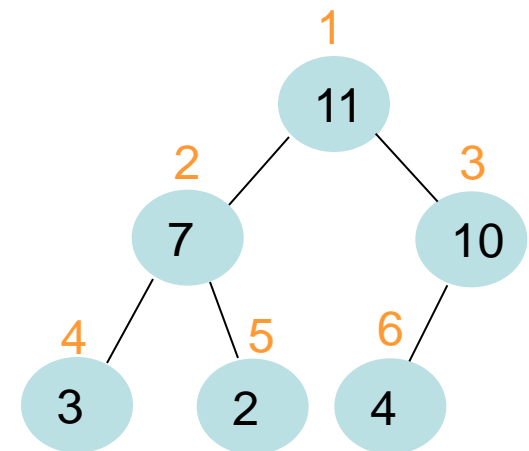
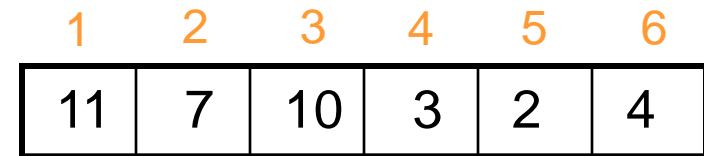


## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

## Datenstrukturen (siehe Vollversion)

Heapsort(A)

1. Build-Heap(A)
2. **for**  $i \leftarrow \text{length}[A]$  **downto** 2 **do**
3.  $A[1] \leftrightarrow A[i]$
4.  $\text{heap-size}[A] \leftarrow \text{heap-size}[A]-1$
5. Heapify(A,1)



## Datenstrukturen (siehe Vollversion)

### Heapsort(A)

1. Build-Heap(A)
2. **for**  $i \leftarrow \text{length}[A]$  **downto** 2 **do**
3.      $A[1] \leftrightarrow A[i]$
4.      $\text{heap-size}[A] \leftarrow \text{heap-size}[A]-1$
5.     Heapify(A,1)

1	2	3	4	5	6
2	3	4	7	10	11

### *Laufzeit*

- $O(n \log n)$

## Datenstrukturen

### *Zusammenfassung (Halden)*

- Einfügen, Löschen, Maximum extrahieren in  $O(\log n)$  Zeit
- Sortieren mit Hilfe von Halden in  $O(n \log n)$
- Heapsort braucht keinen zusätzlichen Speicherplatz
- Einfache Implementierung

## Graphalgorithmen

### *Definition*

- Ein Graph  $G$  ist ein Paar  $(V, E)$ .  $V$  heißt Knotenmenge und  $E \subseteq V \times V$  Kantenmenge des Graphen.

### *Darstellung von Graphen*

- Adjazenzlisten (dünne Graphen,  $|E| \ll |V|^2$ )
- Adjazenzmatrix (dichte Graphen,  $|E|$  nah an  $|V|^2$ )

### *Arten von Graphen*

- Ungerichtet, gerichtet
- Ungewichtet, gewichtet

## Graphalgorithmen

### *Adjazenzmatrixdarstellung*

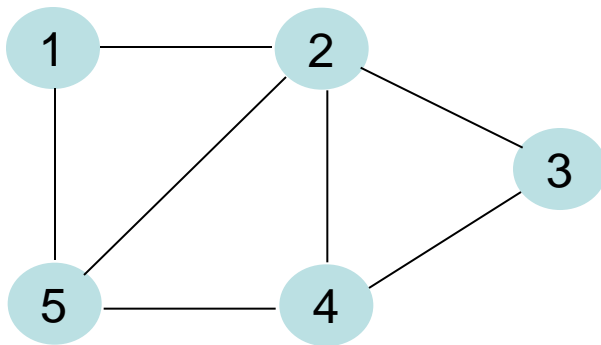
- Knoten sind nummeriert von 1 bis  $|V|$
- $|V| \times |V|$  Matrix  $A = (a_{ij})$  mit

$$a_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{sonst} \end{cases}$$

- Bei ungerichteten Graphen gilt  $A = A^T$

## Graphalgorithmen

### Beispiel



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

## Graphalgorithmen

### *Adjazenzlistendarstellung*

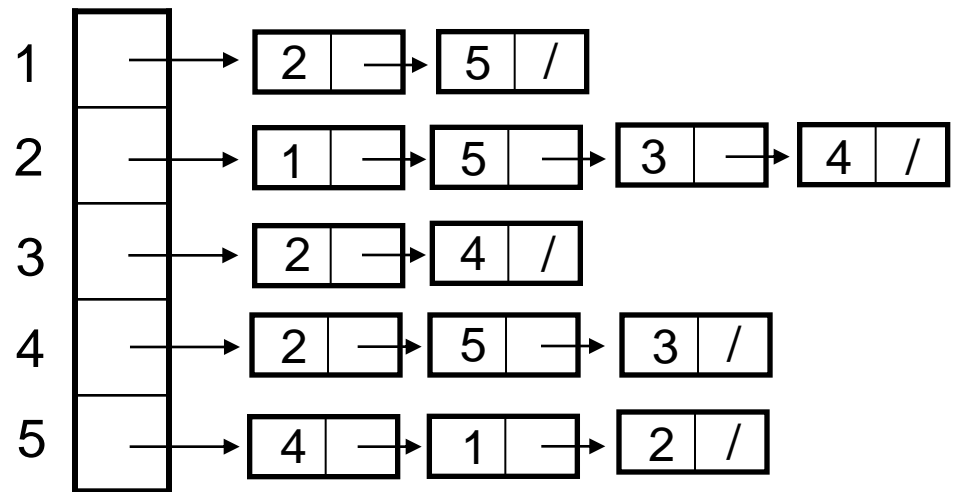
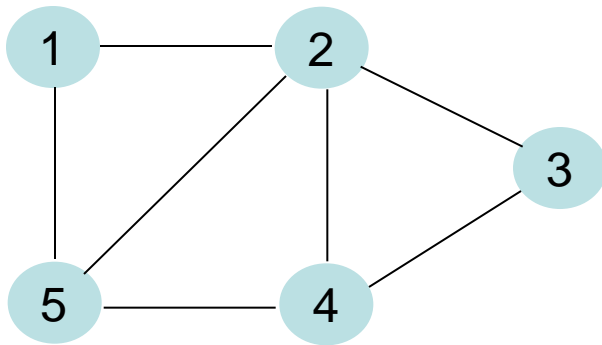
- Feld Adj mit  $|V|$  Listen (eine pro Knoten)
- Für Knoten  $v$  enthält Adj[ $v$ ] eine Liste aller Knoten  $u$  mit  $(v,u) \in E$
- Die Knoten in Adj[ $u$ ] heißen zu  $u$  **adjazent**
- Ist  $G$  ungerichtet, so gilt:  $v \in \text{Adj}[u] \Leftrightarrow u \in \text{Adj}[v]$

### *Gewichtete Graphen*

- Kanten haben Gewicht gegeben durch Funktion  $w: E \rightarrow \mathcal{R}$
- Gewicht  $w(u,v)$  von Kante  $(u,v)$  wird mit Knoten  $v$  in  $u$ 's Adjazenzliste gespeichert

# Graphalgorithmen

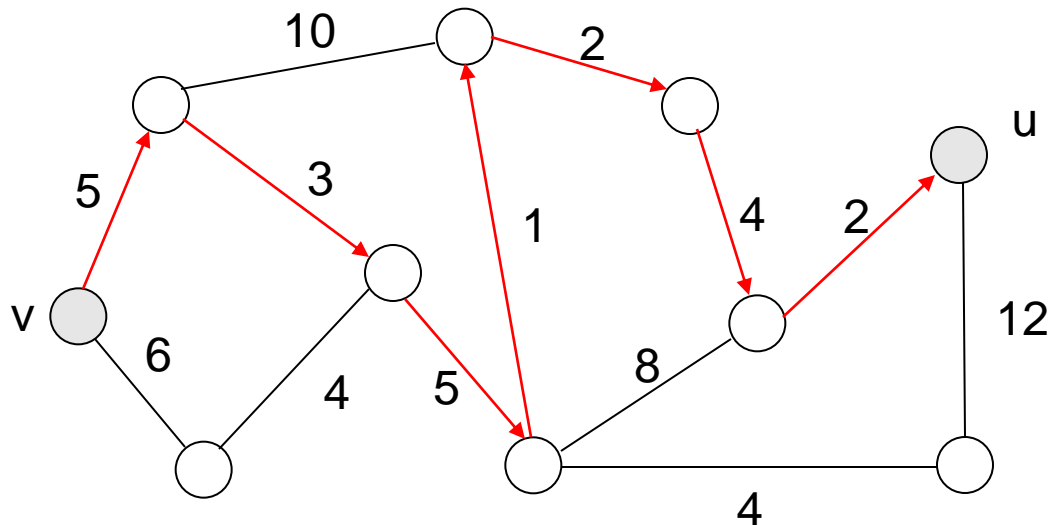
## Beispiel



## Graphalgorithmen

### *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)



## Graphalgorithmen

### *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

### *SSSP in ungerichteten Graphen (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

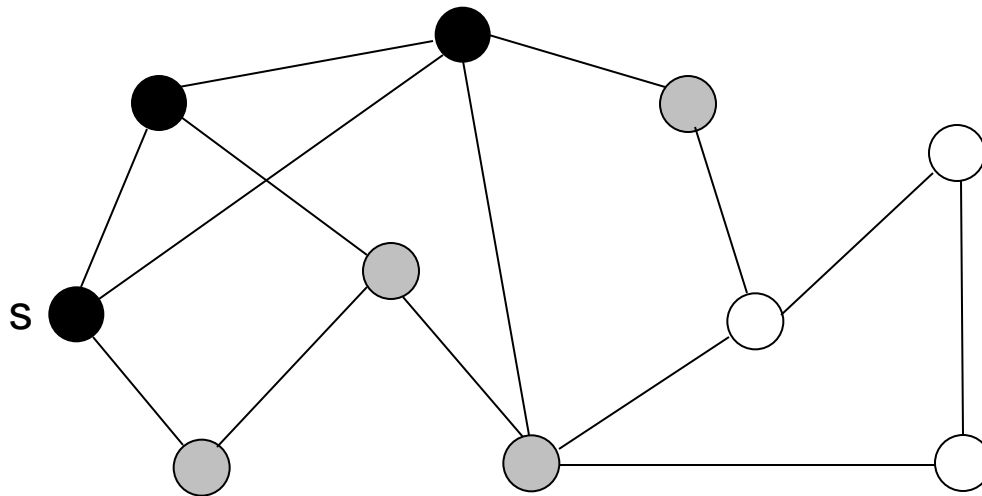
## Graphalgorithmen

### *Invariante (Breitensuche)*

- Knoten haben 3 Farben: weiß, grau und schwarz
- Zu Beginn: Alle Knoten sind weiß
- 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

## Graphalgorithmen

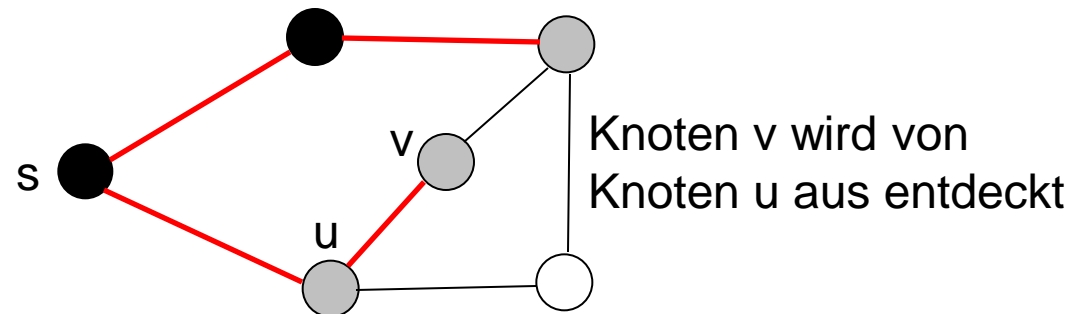
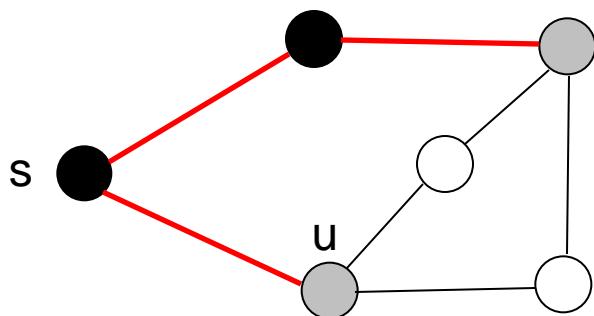
*Beispiel (mögl. Zustand bei einer Breitensuche)*



## Graphalgorithmen

### Breitensuche

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



## Graphalgorithmen

### *Datenstruktur Schlange*

- Operationen: head, enqueue, dequeue
- head: Gibt Referenz auf das erste in der Schlange gespeicherte Element zurück
- enqueue: Fügt neues Element ans Ende der Schlage
- dequeue: Entfernt Kopf der Schlange

### *Wir verwenden*

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

## Graphalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while**  $Q \neq \emptyset$  **do**
3.      $u \leftarrow \text{head}[Q]$
4.     **for each**  $v \in \text{Adj}[u]$  **do**
5.         **if**  $\text{color}[v] = \text{weiß}$  **then**
6.              $\text{color}[v] \leftarrow \text{grau}$
7.              $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8.              $\text{enqueue}(Q, v)$
9.      $\text{dequeue}(Q)$
10.  $\text{color}[u] \leftarrow \text{schwarz}$

$d[u]$ : Abstand zu s (zu Beginn  $\infty$ )

$\pi[u]$ : Vaterknoten von u (zu Beginn nil)

## Graphalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while**  $Q \neq \emptyset$  **do**
3.      $u \leftarrow \text{head}[Q]$
4.     **for each**  $v \in \text{Adj}[u]$  **do**
5.         **if**  $\text{color}[v] = \text{weiß}$  **then**
6.              $\text{color}[v] \leftarrow \text{grau}$
7.              $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8.              $\text{enqueue}(Q, v)$
9.      $\text{dequeue}(Q)$
10.     $\text{color}[u] \leftarrow \text{schwarz}$

Für jeden Knoten u:

- $\text{color}[u] = \text{weiß}$
- $d[u] = \infty$
- $\pi[u] = \text{nil}$

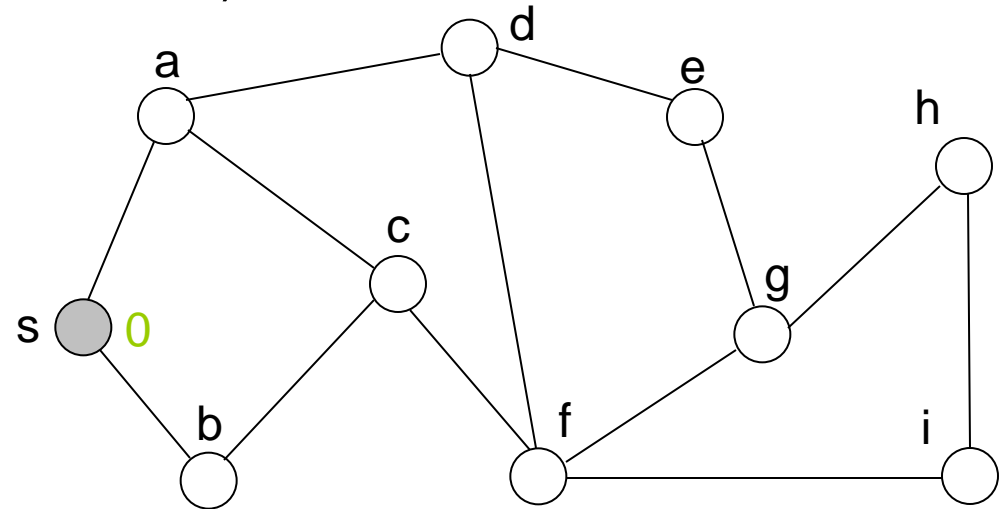
Für Knoten s:

- $\text{color}[s] = \text{grau}$
- $d[s] = 0$
- $\pi[s] = \text{nil}$
- s wird in Schlange Q eingefügt

## Graphalgorithmen (siehe Vollversion)

BFS(G,s)

1. „initialisiere BFS“
2. **while**  $Q \neq \emptyset$  **do**
3.      $u \leftarrow \text{head}[Q]$
4.     **for each**  $v \in \text{Adj}[u]$  **do**
5.         **if**  $\text{color}[v] = \text{weiß}$  **then**
6.              $\text{color}[v] \leftarrow \text{grau}$
7.              $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8.              $\text{enqueue}(Q, v)$
9.      $\text{dequeue}(Q)$
10.     $\text{color}[u] \leftarrow \text{schwarz}$

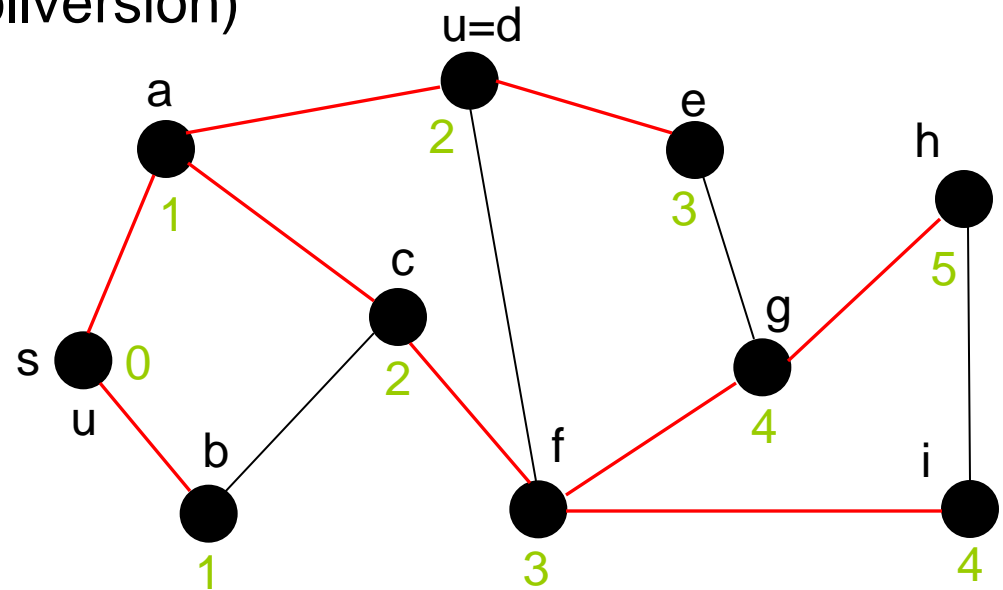


Q: s

## Graphalgorithmen (siehe Vollversion)

BFS(G,s)

1. „initialisiere BFS“
2. **while**  $Q \neq \emptyset$  **do**
3.      $u \leftarrow \text{head}[Q]$
4.     **for each**  $v \in \text{Adj}[u]$  **do**
5.         **if**  $\text{color}[v] = \text{weiß}$  **then**
6.              $\text{color}[v] \leftarrow \text{grau}$
7.              $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8.              $\text{enqueue}(Q, v)$
9.      $\text{dequeue}(Q)$
10.     $\text{color}[u] \leftarrow \text{schwarz}$



## Graphalgorithmen

### Satz 47

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 Liste:  $O(|E|)$
- Gesamtlaufzeit:  $O(|V|+|E|)$

## Graphalgorithmen

### *Kürzeste Wege in ungewichteten Graphen*

- 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

## Graphalgorithmen

### *Lemma 48*

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.

## Graphalgorithmen

### *Lemma 49*

Sei  $G=(V,E)$  ein gerichteter oder ungerichteter Graph und es laufe die Breitensuche von einem Startknoten  $s \in V$ . Nach Abschluss 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 in  $Q$  eingefügt wird
- (I.A.) Nach Initialisierung gilt  $d[s]=0=\delta(s,s)$  und  $d[v]=\infty \geq \delta(s,v)$  für alle  $v \in V - \{s\}$
- (I.V.) Aussage gilt nach  $m$  Einfügeoperationen
- (I.S.) Betrachte nach  $m$  Einfügeoperationen einen weißen Knoten  $v$ , der während einer Suche von  $u$  entdeckt wird. Nach (I.V.) gilt  $d[u] \geq \delta(s,u)$ .
- 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 48

## Graphalgorithmen

### *Lemma 49*

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

### *Beweis*

- 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

## Graphalgorithmen

### *Lemma 50*

Sei  $\langle v_1, \dots, v_r \rangle$  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  
 $d[v_r] \leq d[v_1] + 1$  und  $d[v_i] \leq d[v_{i+1}]$  für  $i=1,2,\dots,r-1$ .

### *Beweis*

- Induktion über die Anzahl Schlangenoperationen
- (I.A.) Die Schlange enthält nur  $s$ , damit gilt das Lemma
- (I.V.) Das Lemma gilt nach  $m$  Schlangenoperationen
- (I.S.) 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

## Graphalgorithmen

### *Lemma 50*

Sei  $\langle v_1, \dots, v_r \rangle$  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  
 $d[v_r] \leq d[v_1] + 1$  und  $d[v_i] \leq d[v_{i+1}]$  für  $i=1,2,\dots,r-1$ .

### *Beweis*

- 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

## Graphalgorithmen

### *Lemma 50*

Sei  $\langle v_1, \dots, v_r \rangle$  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  
 $d[v_r] \leq d[v_1] + 1$  und  $d[v_i] \leq d[v_{i+1}]$  für  $i=1,2,\dots,r-1$ .

### *Beweis*

- 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

## Graphalgorithmen

### Satz 51

Sei  $G=(V,E)$  ein gerichteter oder ungerichteter Graph und sei  $s \in V$  Startknoten der Breitensuche. Dann entdeckt die Breitensuche alle Knoten  $v \in V$ , die von  $s$  aus erreichbar sind und nach Terminierung gilt  $d[v]=\delta(s,v)$  für alle  $v \in V$ . Außerdem gilt für jeden von  $s$  erreichbaren Knoten  $v \neq s$ , dass ein kürzester Weg von  $s$  nach  $\pi[v]$  gefolgt von der Kante  $(\pi[v],v)$  ein kürzester  $s$ - $v$ -Weg ist.

### Beweis

- Fall 1 ( $v$  nicht erreichbar von  $s$ ):
- Nach Lemma 49 gilt  $d[v] \geq \delta(s,v) = \infty$
- Es kann keinen ersten Knoten geben, dessen  $d$ -Wert in Zeile 7 auf  $\infty$  gesetzt wird
- Somit wird  $v$  nie entdeckt

## Graphalgorithmen

### *Beweis*

- Fall 2 ( $v$  erreichbar von  $s$ ):
- Sei  $V_k = \{v \in V : \delta(s, v) = k\}$
- Wir zeigen per Induktion über  $k$ :
- Es gibt genau einen Zeitpunkt, zu dem jeder Knoten  $v \in V_k$ 
  - (a) grau gefärbt wird
  - (b)  $d[v]=k$  gesetzt wird
  - (c) wenn  $v \neq s$ , dann  $\pi[v]$  auf  $u$  gesetzt wird für ein  $u \in V_{k-1}$
  - (d)  $v$  in Schlange  $Q$  eingefügt wird
- Da nur zur Initialisierung Knoten weiß gefärbt werden, gibt es maximal einen solchen Zeitpunkt

## Graphalgorithmen

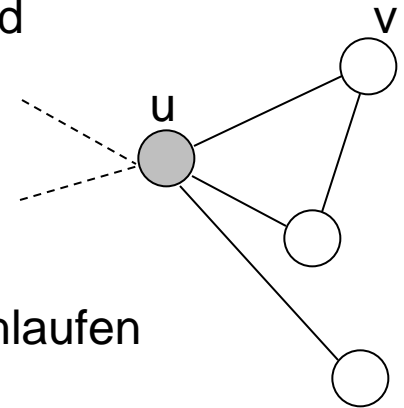
### Beweis

- (I.A.)  $V_0 = \{s\}$ . Während der Initialisierung wird  $s$  grau gefärbt,  $d[s]$  auf 0 gesetzt, und  $s$  in  $Q$  eingefügt. Somit gilt die Aussage.
- (I.V.) Aussage gilt für alle Knoten aus  $V_{k-1}$
- (I.S.)  $Q$  ist nie leer bis Algorithmus terminiert.
- Nachdem  $v$  in  $Q$  eingefügt wurde, ändern sich  $d[v]$  und  $\pi[v]$  nicht mehr
- Nach Lemma 50 sind die  $d$ -Werte monoton steigend, wenn Knoten in die Schlange eingefügt werden
- Betrachte nun  $v \in V_k$ ,  $k > 0$ .  
Monotonie mit  $d[v] \geq k$  (Lemma 49) und (I.V.): wenn  $v$  entdeckt wird, dann erst nachdem alle Knoten aus  $V_{k-1}$  in die Schlange eingefügt wurden

## Graphalgorithmen

### Beweis

- Da  $\delta(s,v)=k$  gibt es Pfad mit  $k$  Kanten von  $s$  nach  $v$  und Knoten  $u$  mit  $(u,v) \in E$  und  $u \in V_{k-1}$
- ObdA. Sei  $u$  der erste solche Knoten, der grau gefärbt wird (existiert wegen I.V.)
- Wird Knoten grau gefärbt, so wird er auch in Schlange eingefügt und muss irgendwann als Kopf der Schlange auftauchen
- Ist  $u$  Kopf der Schlange, so wird seine Adjazenzliste durchlaufen und  $v$  in Zeile 4 entdeckt
- Dann wird  $v$  in Zeile 6 grau gefärbt und Zeile 7 setzt  $d[v]=k$  und  $\pi[v]=u$ .
- Zeile 8 fügt  $v$  in die Schlange ein
- Damit folgt unsere Aussage per Induktion für alle  $V_k$



## Graphalgorithmen

### *Beweis*

- Abschließend beobachten wir, dass wenn  $v \in V_k$  ist, dann ist  $\pi[v]$  in  $V_{k-1}$
- Damit können wir einen kürzesten Weg von  $s$  nach  $v$  bekommen, indem wir einen kürzesten Weg von  $s$  nach  $\pi[v]$  nehmen und der Kante  $(\pi[v], v)$  folgen

## Graphalgorithmen

### *Zusammenfassung (Breitensuche)*

- Die Breitensuche kann dazu genutzt werden, um das SSSP Problem in ungewichteten Graphen zu lösen
- Die Laufzeit der Breitensuche ist  $O(|V|+|E|)$