

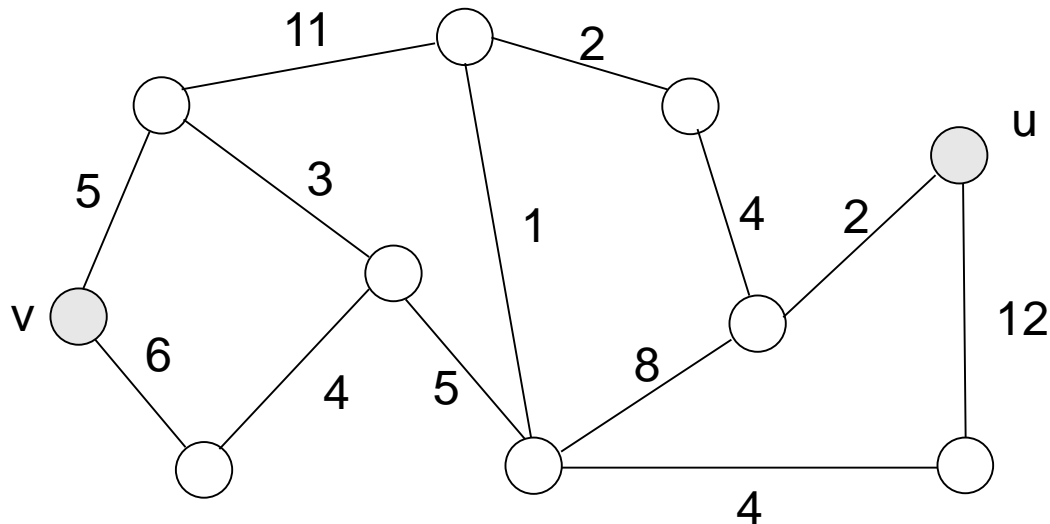


## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

## Graphalgorithmen

### *Kürzeste Wege in Graphen*

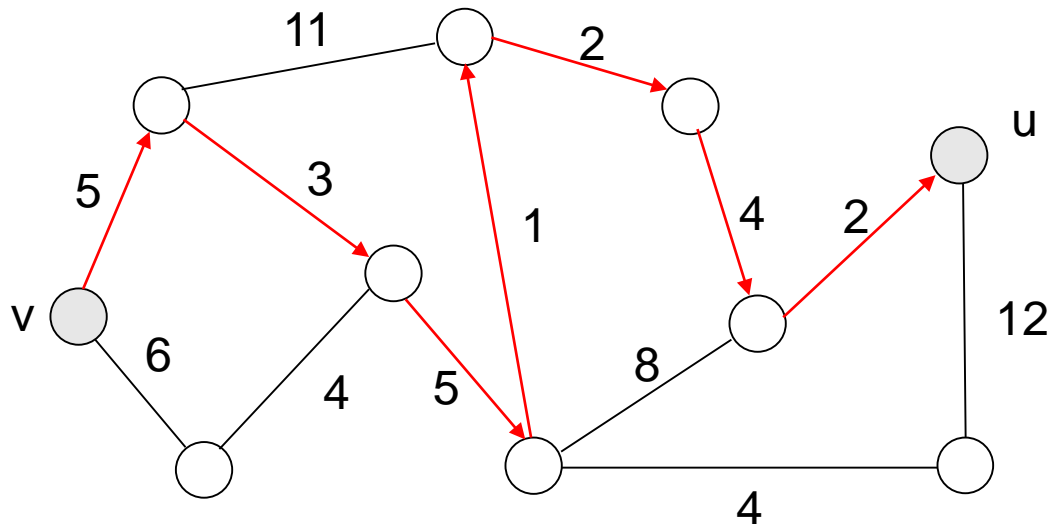
- Gegeben (möglicherweise gewichteter) Graph  $G = (V, E)$
- Frage: Was ist der kürzeste Weg Knoten  $v$  nach Knoten  $u$ ?
- Länge des Weges: Summe der Kantengewichte  
(bzw. Anzahl Kanten, wenn ungewichtet)



## Graphalgorithmen

### *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 ungewichteten 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

## *Breitensuche*

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

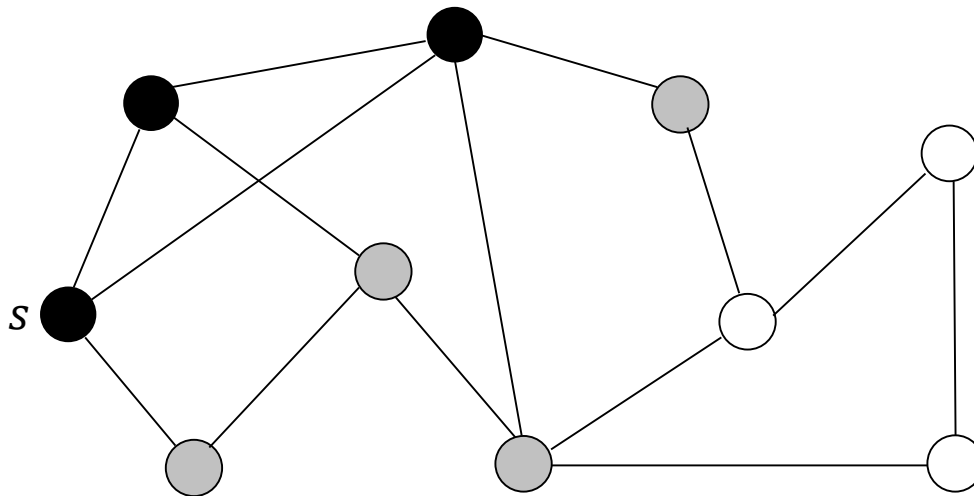
## Graphalgorithmen

### *Technische 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

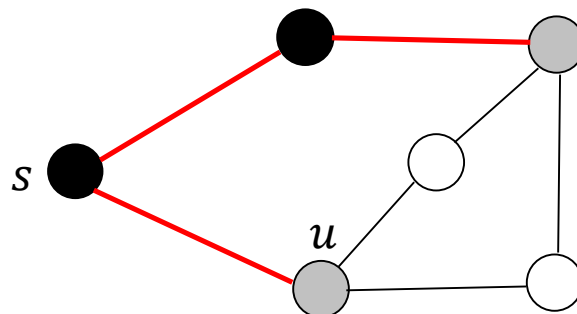
*Beispiel (mögl. Zustand bei einer Breitensuche)*



## 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  $v$

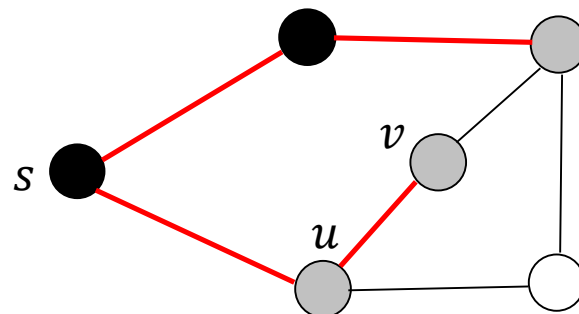




## Graphalgorithmen

### 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 Schlange
- 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.     $\text{color}[u] \leftarrow \text{schwarz}$

## 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  $u$ :

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

## 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  $u$ :

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

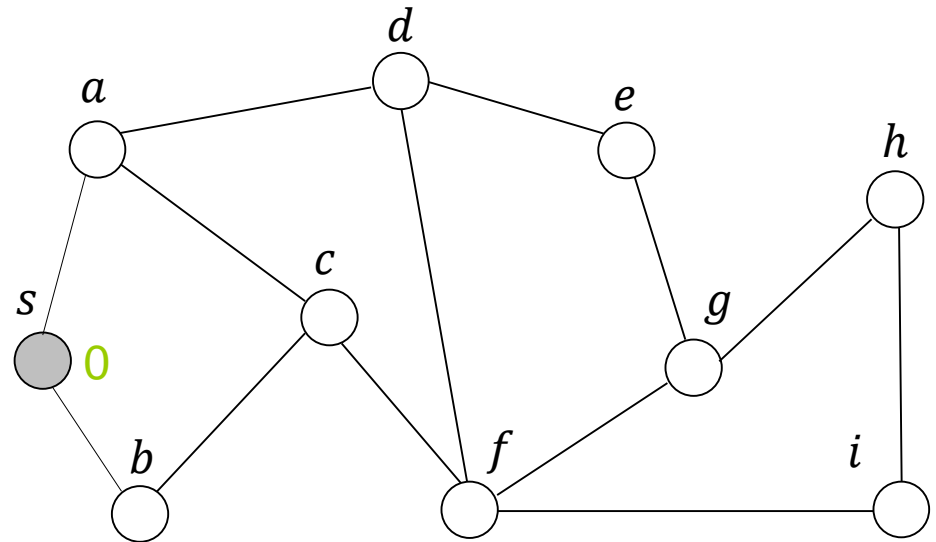
Für Knoten  $s$ :

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

## Graphalgorithmen

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.  $\text{color}[u] \leftarrow \text{schwarz}$

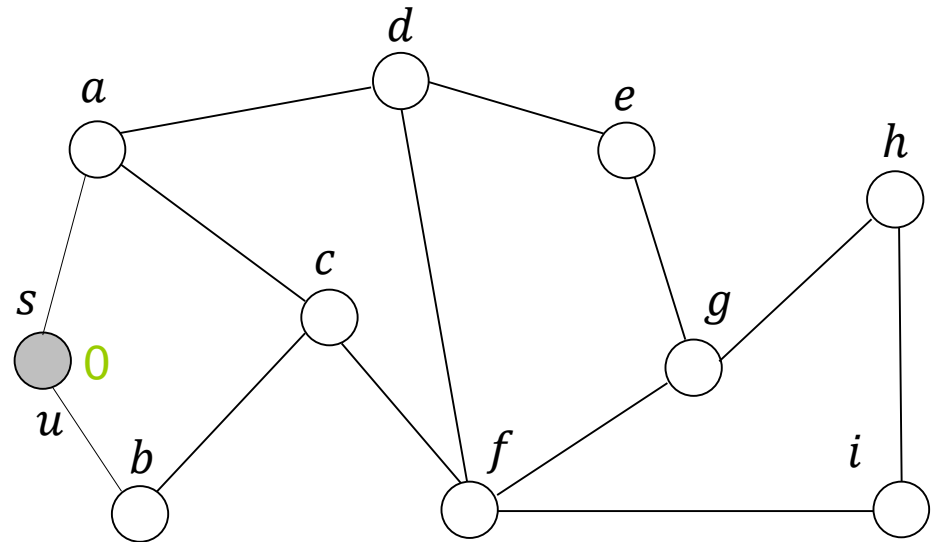


$Q: s$

## Graphalgorithmen

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.  $\text{color}[u] \leftarrow \text{schwarz}$



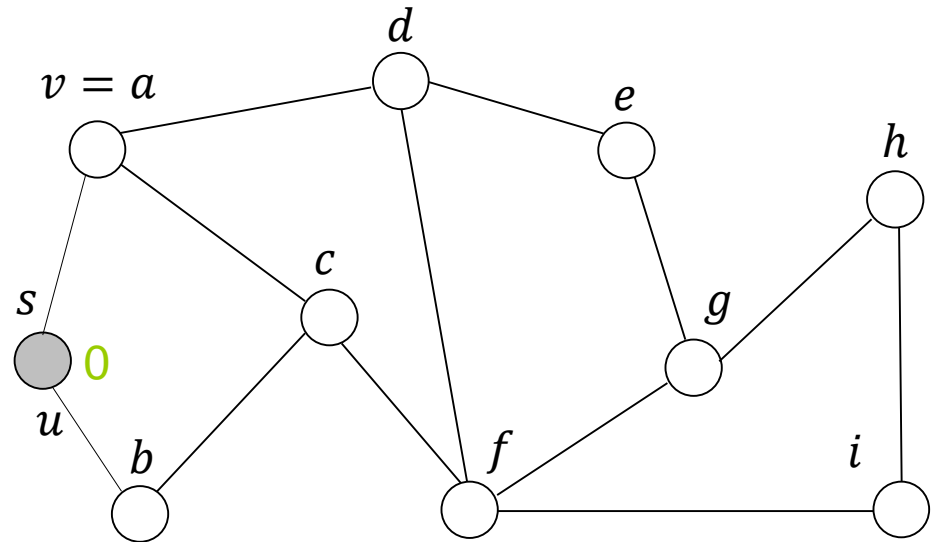
$Q: s$



## Graphalgorithmen

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.  $\text{color}[u] \leftarrow \text{schwarz}$

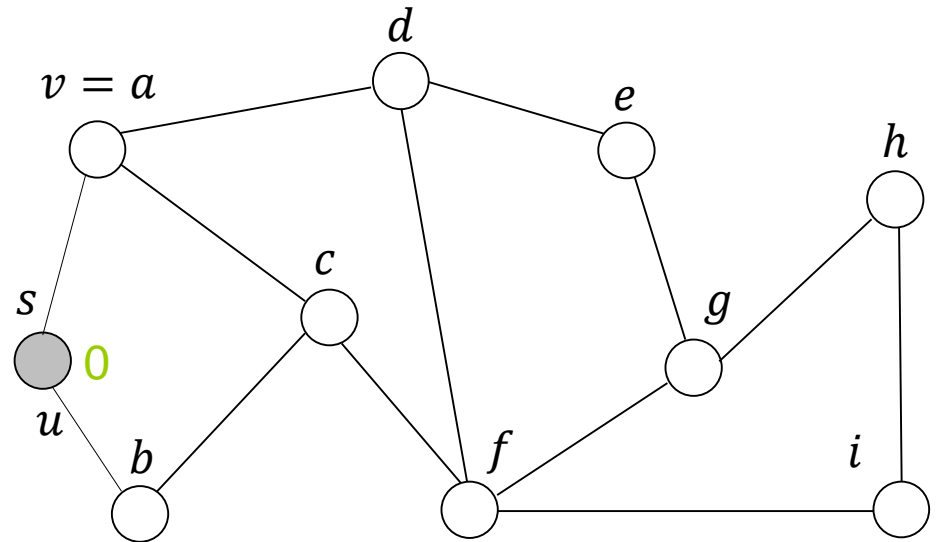


$Q: s$

## Graphalgorithmen

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.  $\text{color}[u] \leftarrow \text{schwarz}$

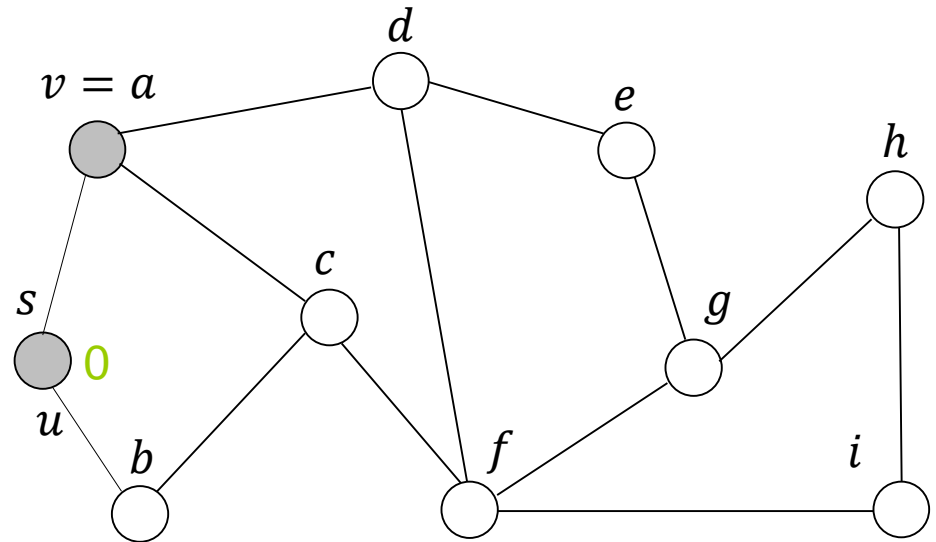


$Q: s$

## Graphalgorithmen

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.       enqueue( $Q, v$ )
9.   dequeue( $Q$ )
10.  $\text{color}[u] \leftarrow \text{schwarz}$

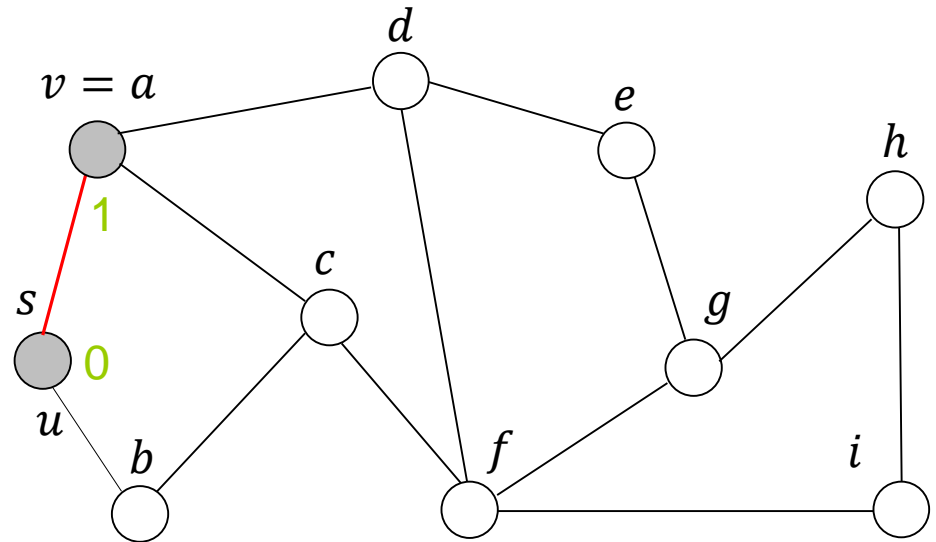


$Q: s$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

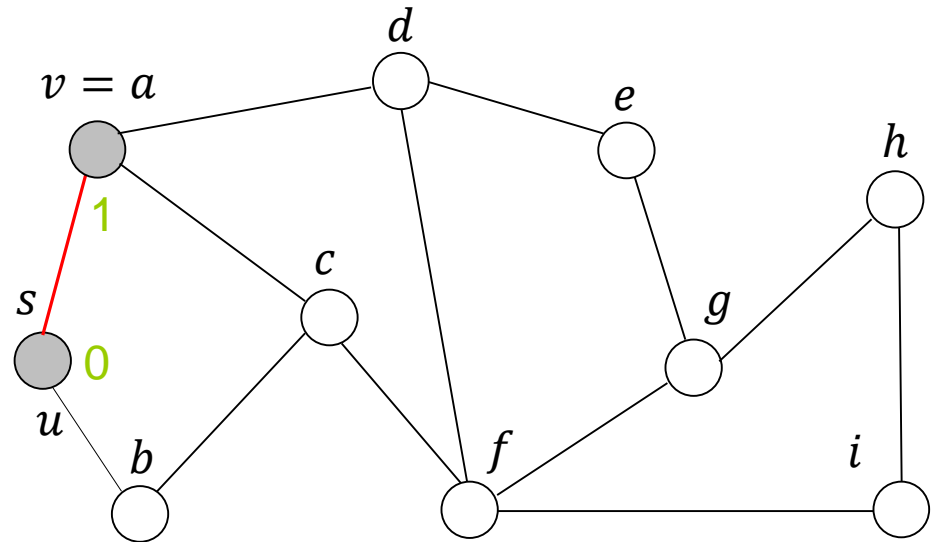


$Q: s$

## 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.       **enqueue**( $Q, v$ )
9.   **dequeue**( $Q$ )
10.  $\text{color}[u] \leftarrow \text{schwarz}$

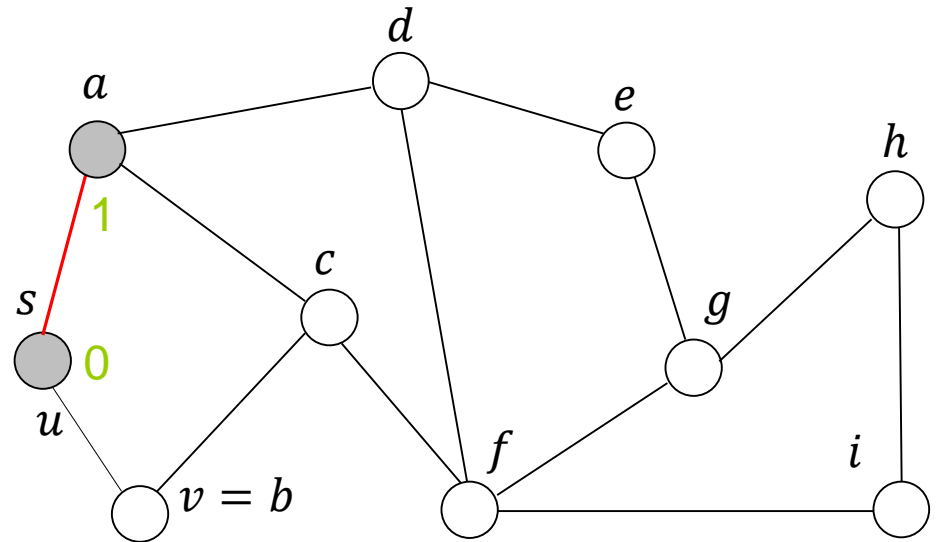


$Q: s, a$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

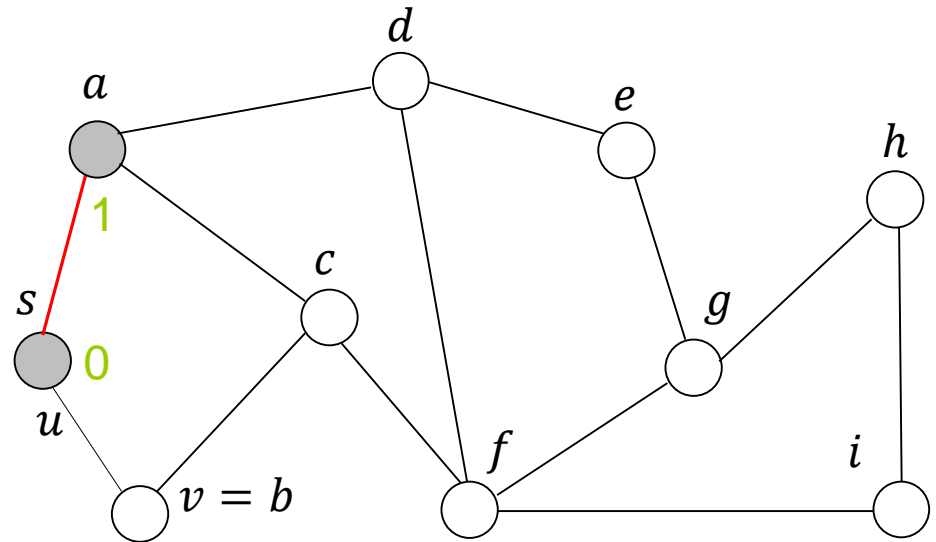


$Q: s, a$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

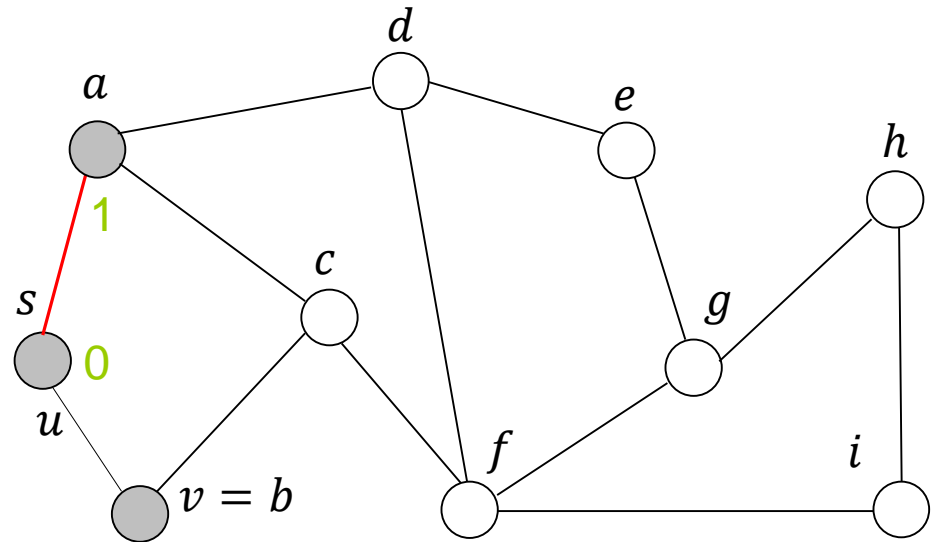


$Q: s, a$

## 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.       enqueue( $Q, v$ )
9.   dequeue( $Q$ )
10.  $\text{color}[u] \leftarrow \text{schwarz}$



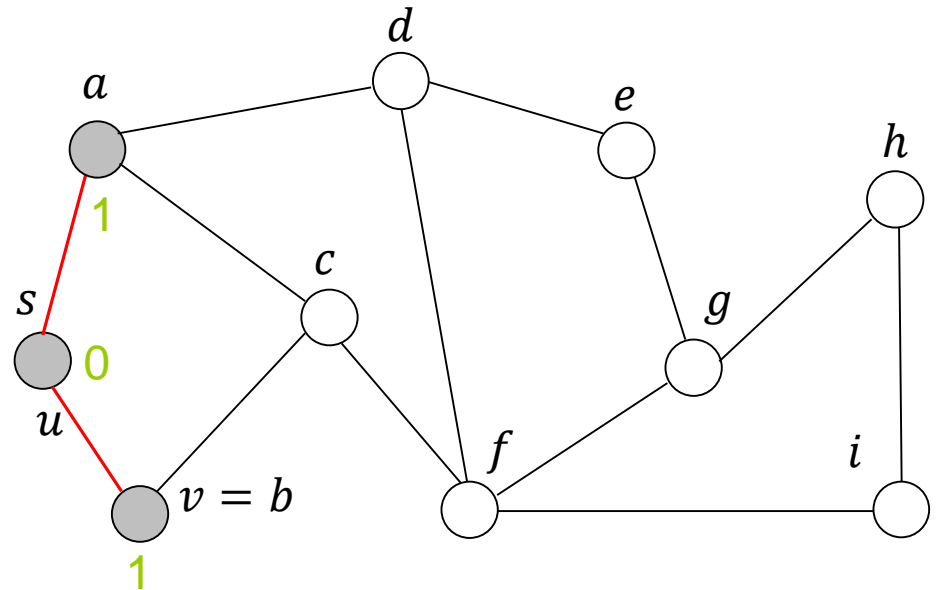
$Q: s, a$



## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

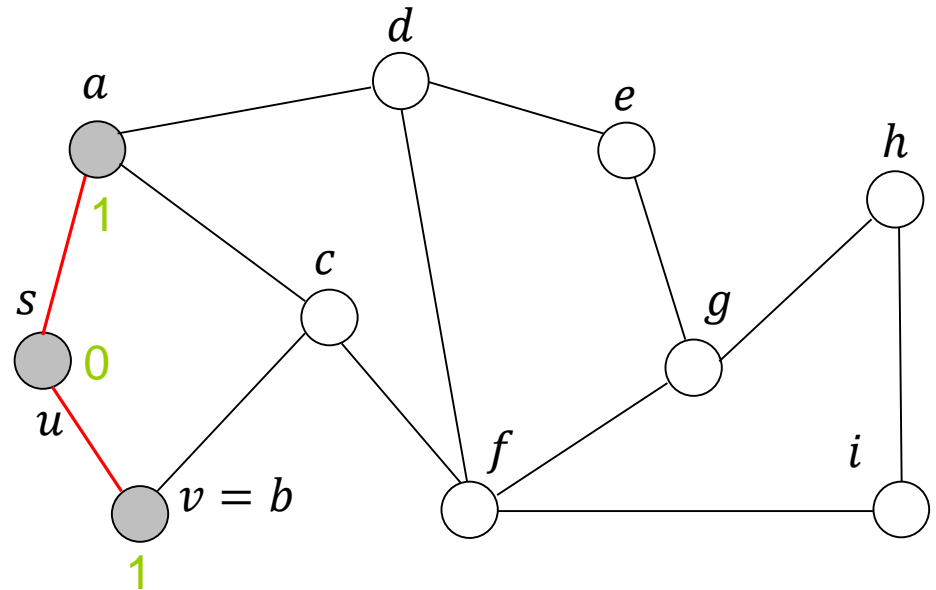


$Q: s, a$

## 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.       **enqueue**( $Q, v$ )
9.   **dequeue**( $Q$ )
10.  $\text{color}[u] \leftarrow \text{schwarz}$

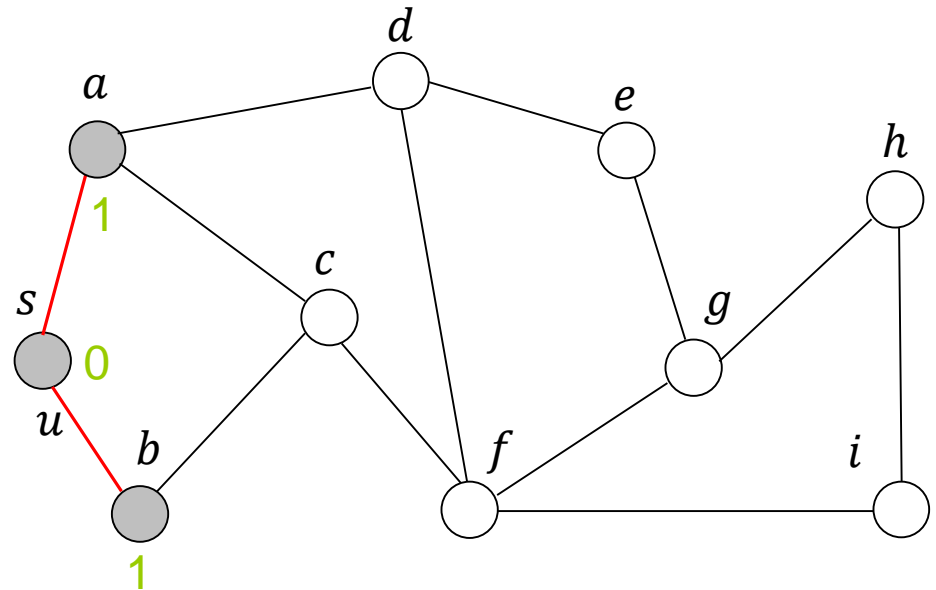


$Q: s, a, b$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

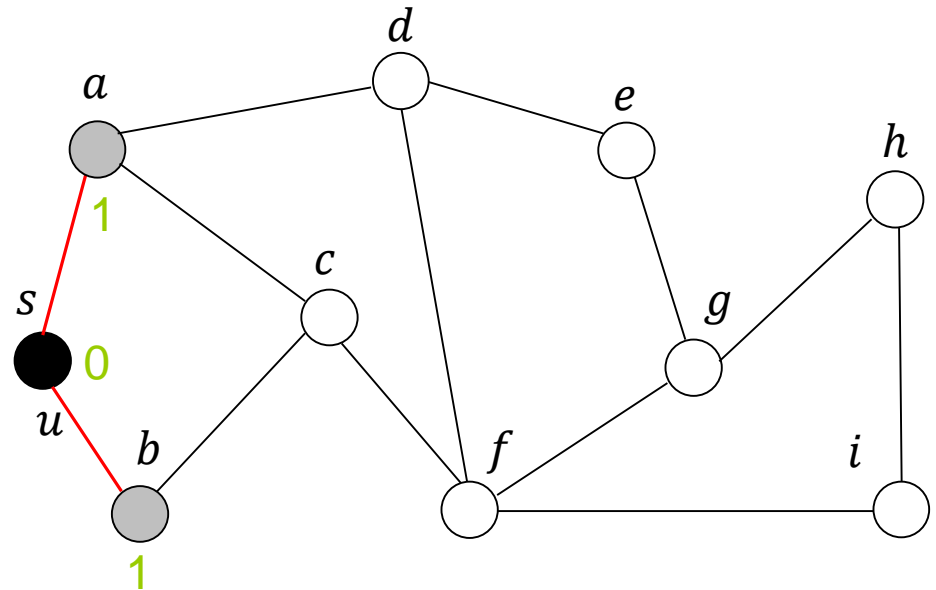


$Q: a, b$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

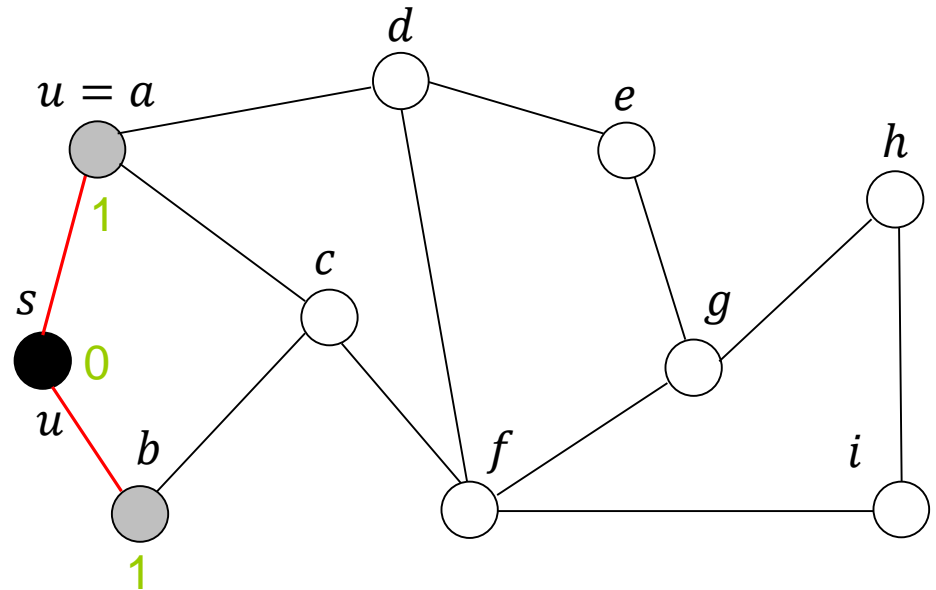


$Q: a, b$

## 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.       enqueue( $Q, v$ )
9.   dequeue( $Q$ )
10.  $\text{color}[u] \leftarrow \text{schwarz}$

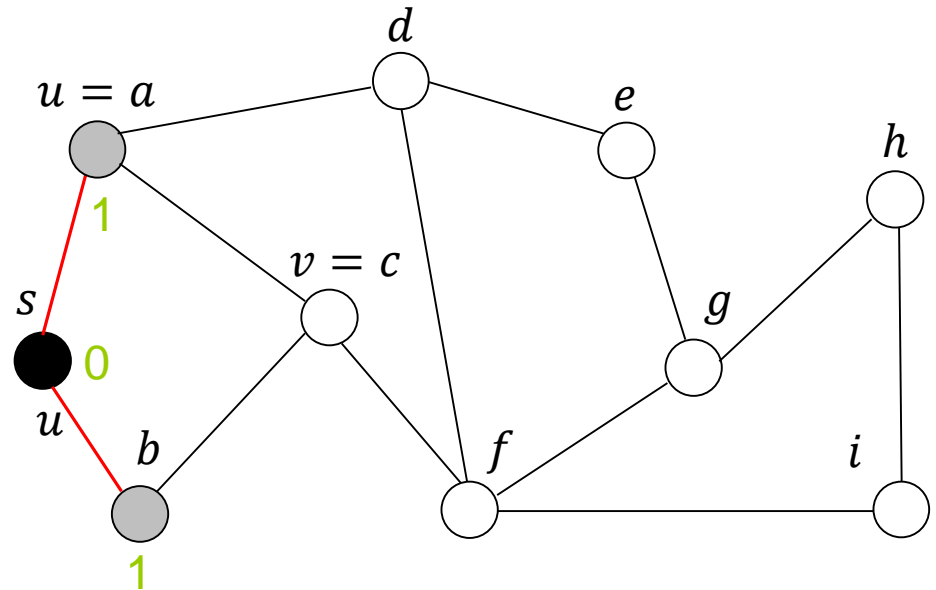


$Q: a, b$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

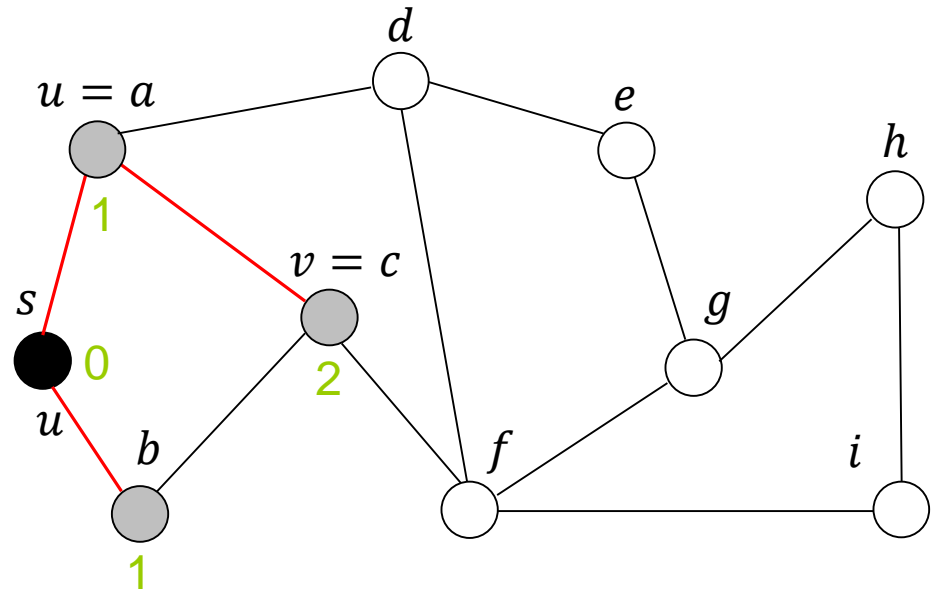


$Q: a, b$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

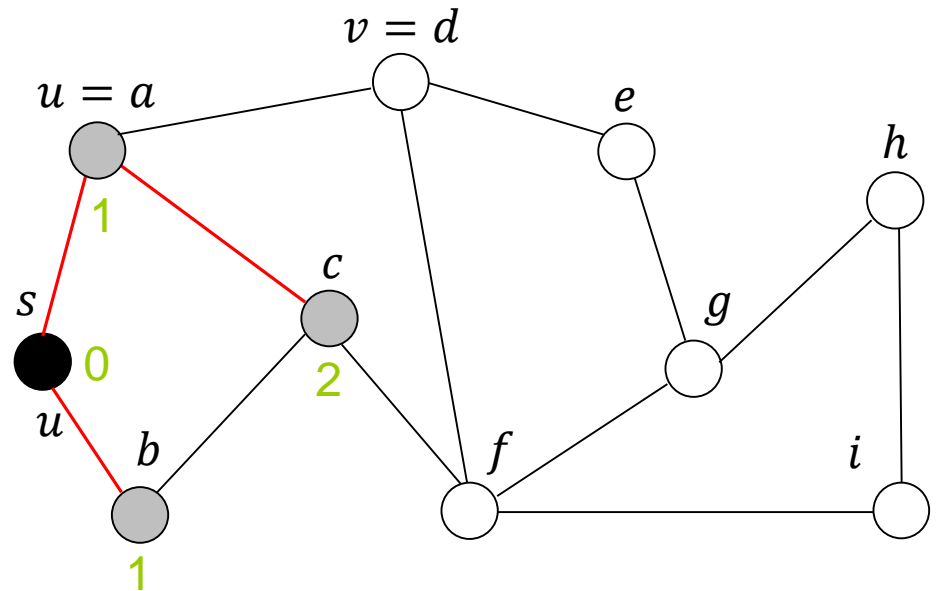


$Q: a, b, c$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$



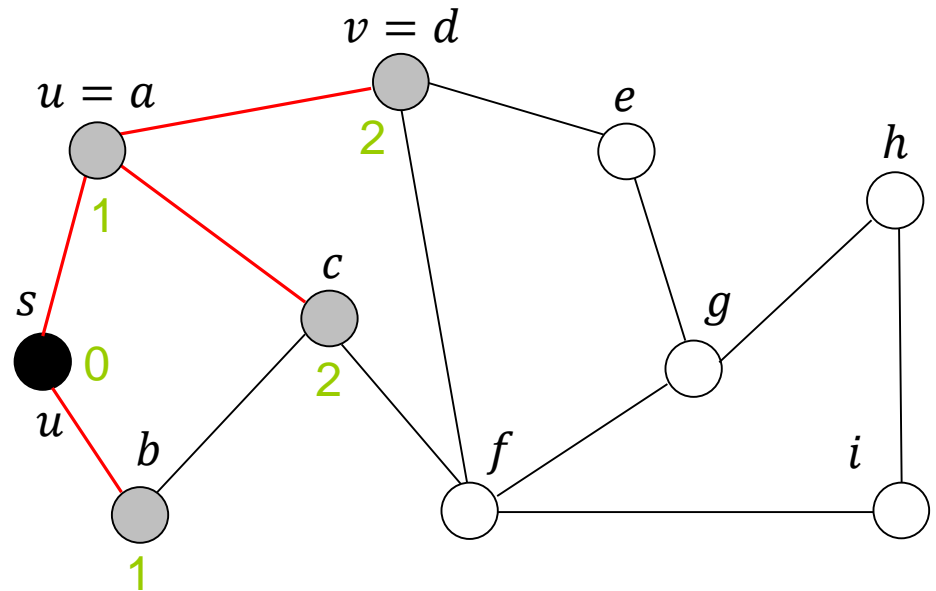
$Q: a, b, c$



## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

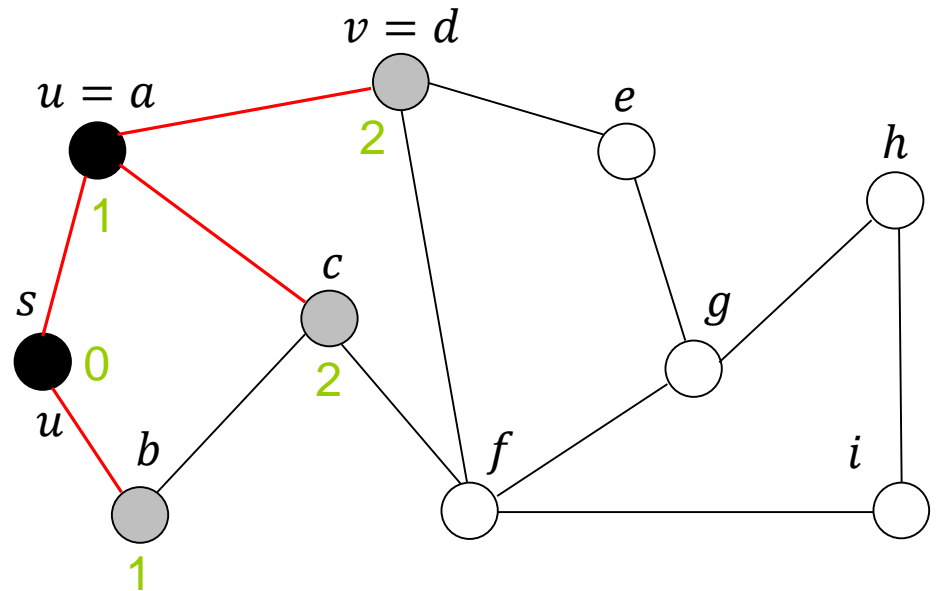


$Q: a, b, c, d$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

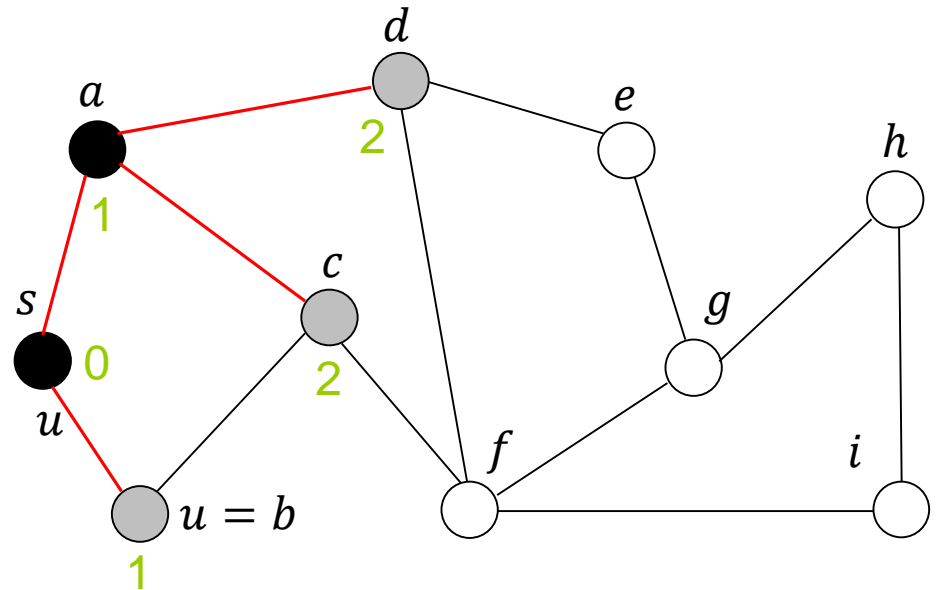


$Q: b, c, d$

## 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.       enqueue( $Q, v$ )
9.   dequeue( $Q$ )
10.  $\text{color}[u] \leftarrow \text{schwarz}$

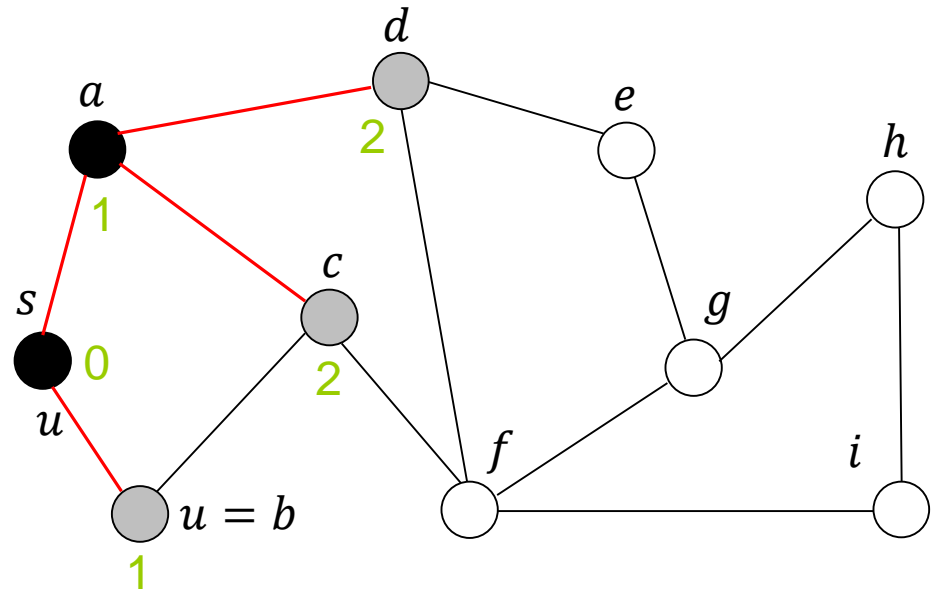


$Q: b, c, d$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

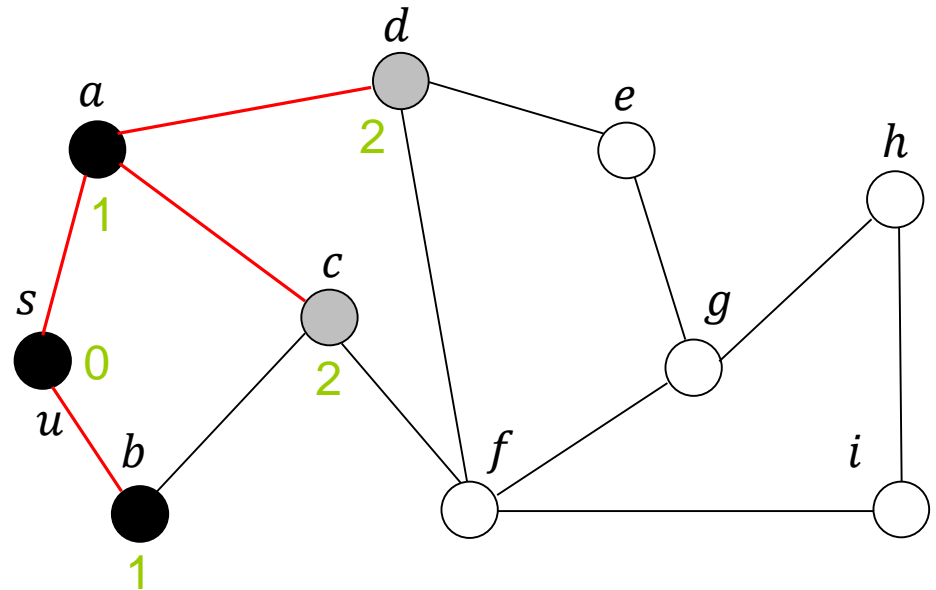


$Q: b, c, d$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

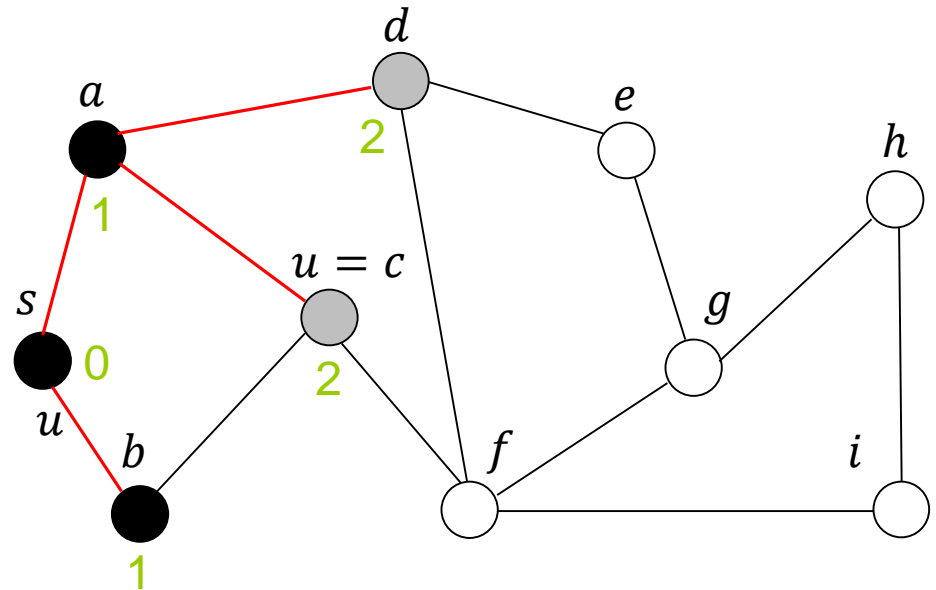


$Q: c, d$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

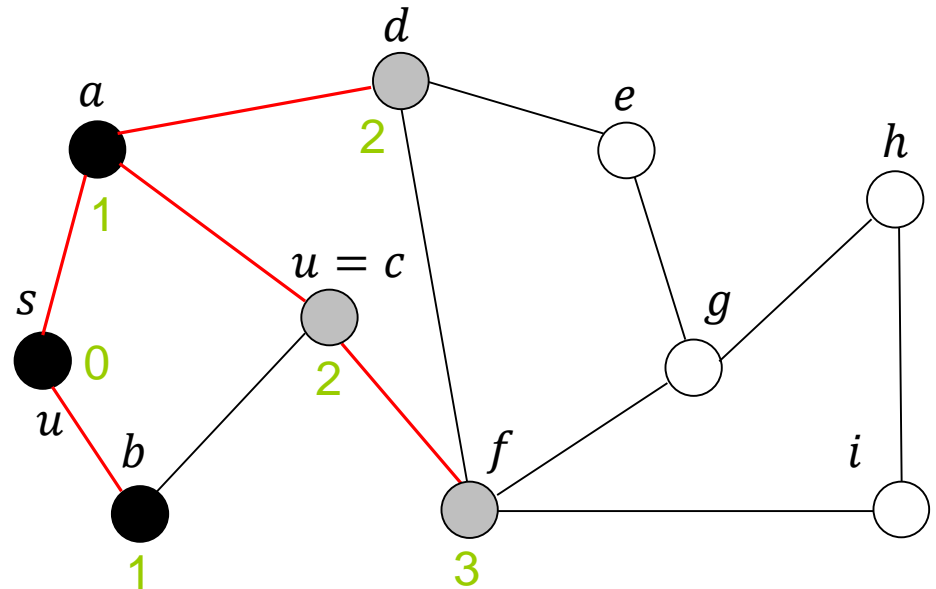


$Q: c, d$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

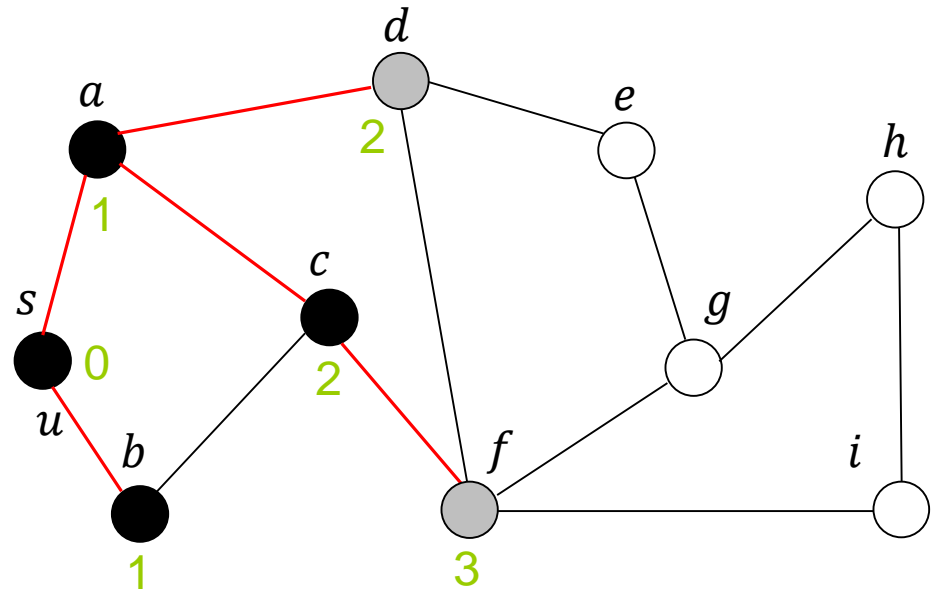


$Q: c, d, f$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$



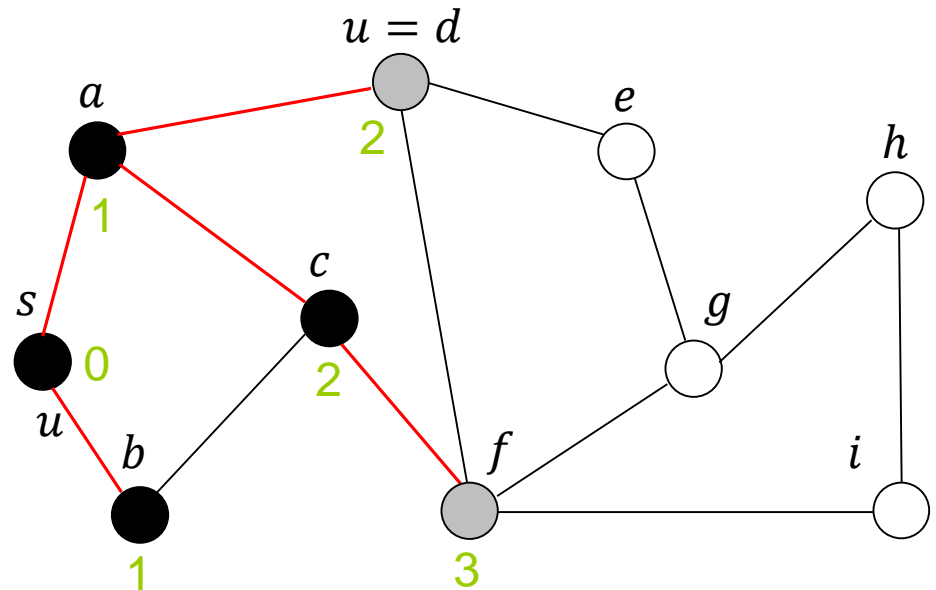
$Q: d, f$



## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

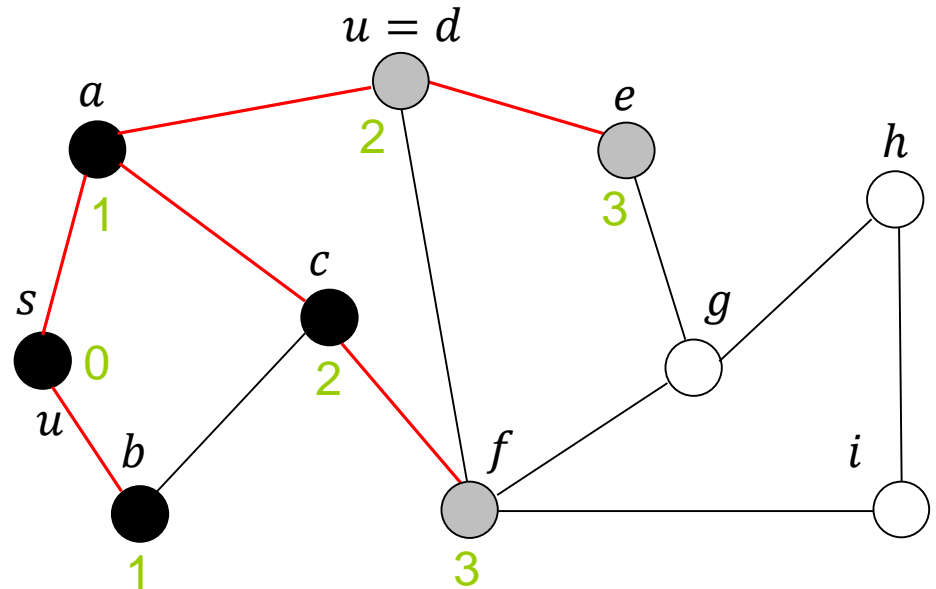


$Q: d, f$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

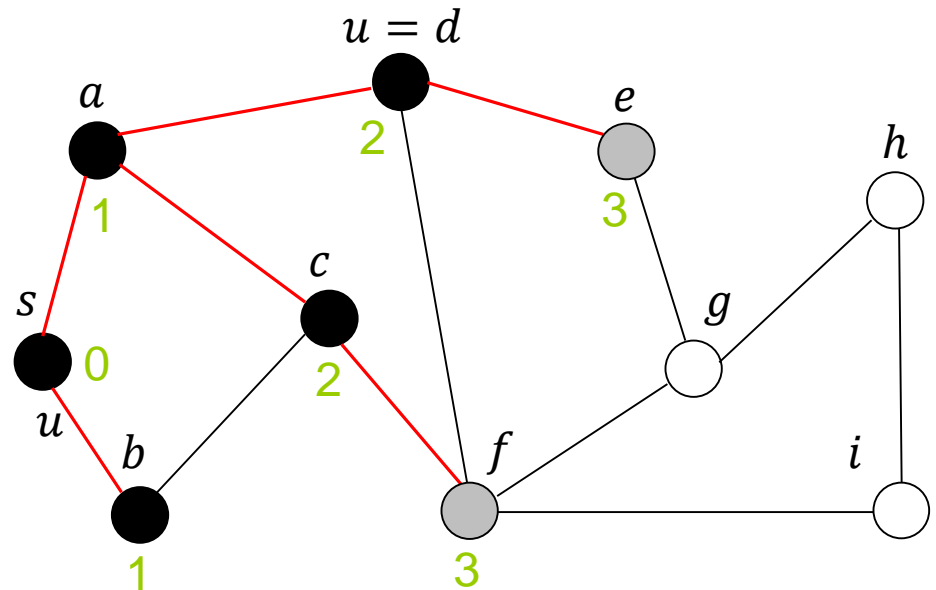


$Q: d, f, e$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

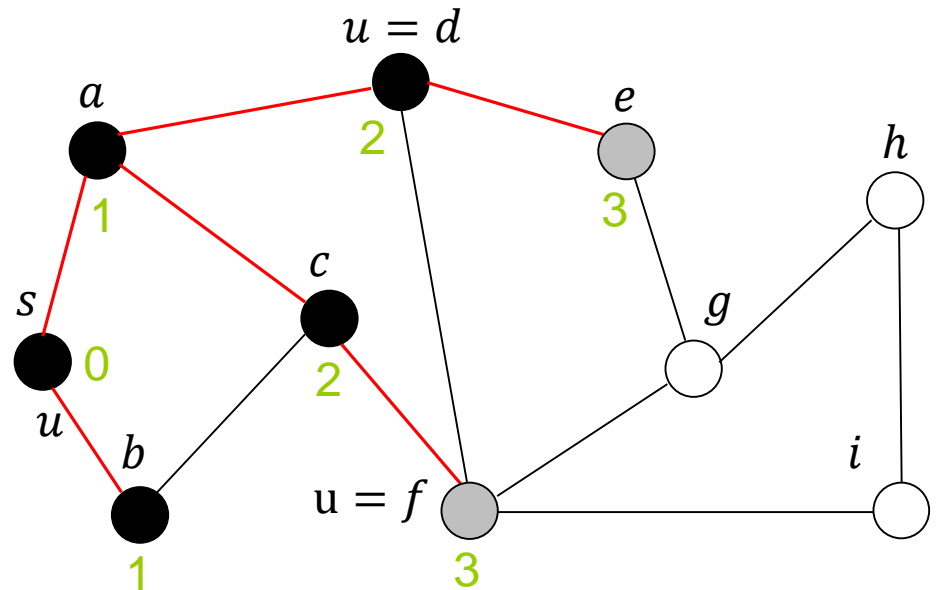


$Q: f, e$

## 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.       enqueue( $Q, v$ )
9.   dequeue( $Q$ )
10.  $\text{color}[u] \leftarrow \text{schwarz}$

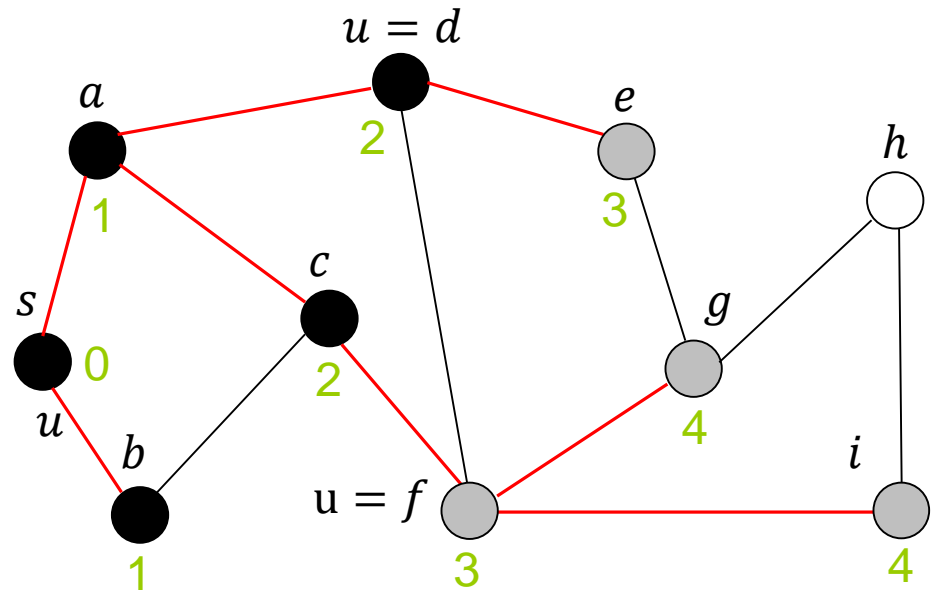


$Q: f, e$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

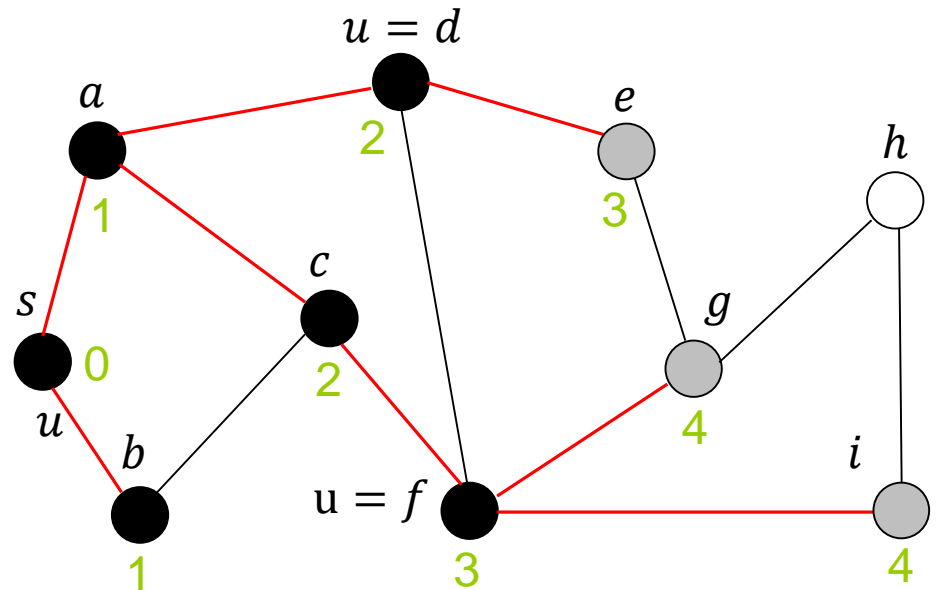


$Q: f, e, g, i$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

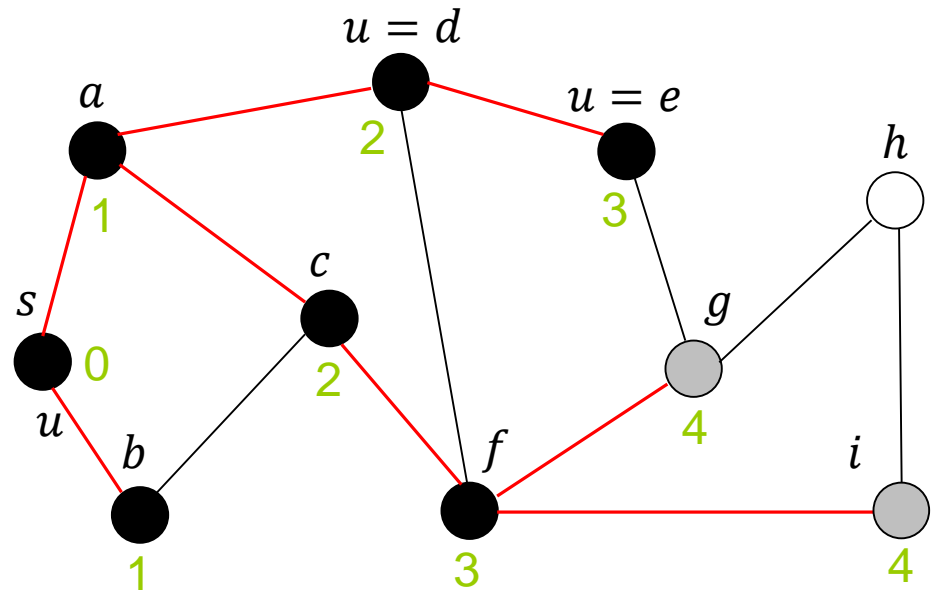


$Q: e, g, i$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

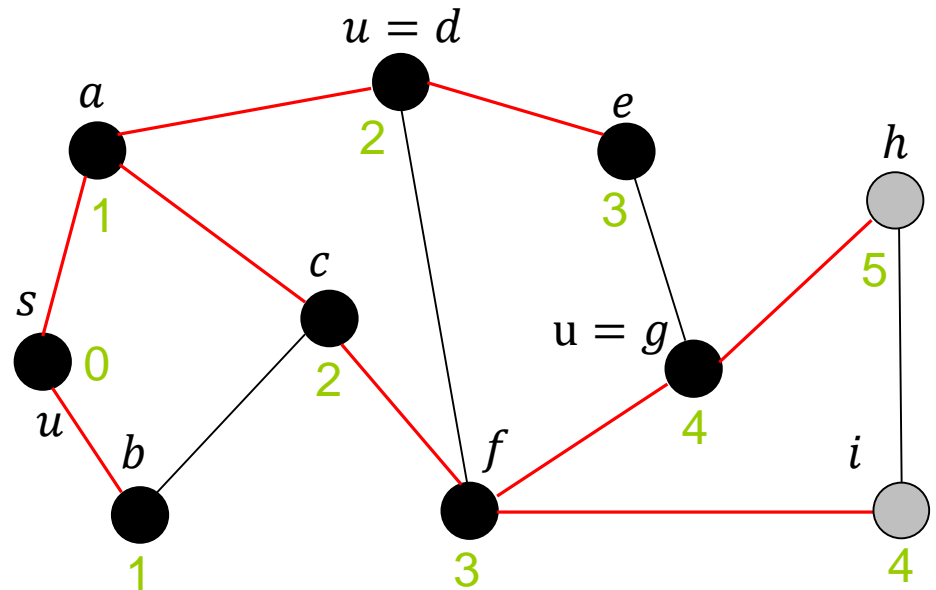


$Q: g, i$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$



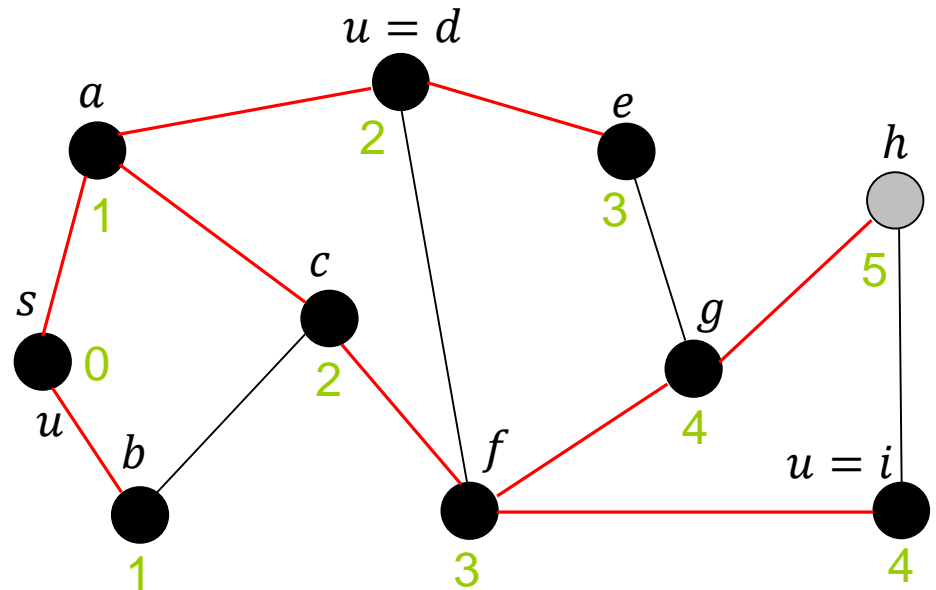
$Q: i, h$



## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

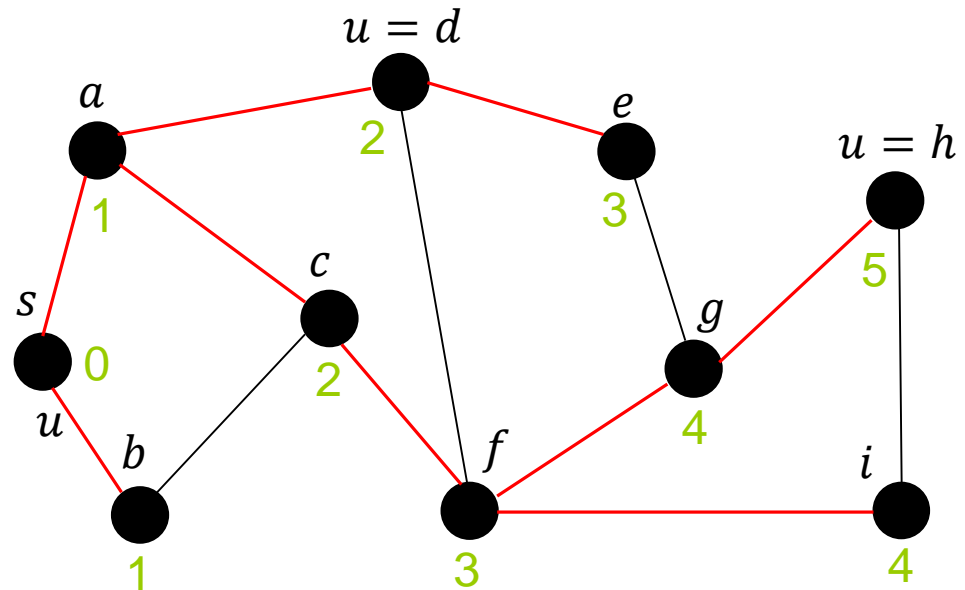


$Q: h$

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$

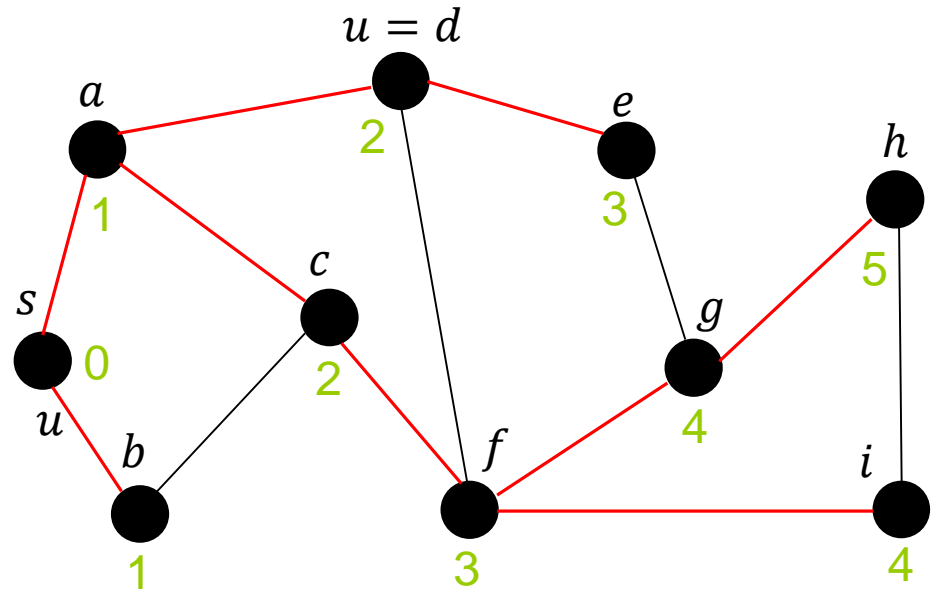


Q:

## 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.  $\text{color}[u] \leftarrow \text{schwarz}$



## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

## Graphalgorithmen

### *Satz 44*

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### *Beweis*

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$

## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### Beweis

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt

## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### Beweis

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange

## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### Beweis

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange
- $\Rightarrow$  Zeit für Schlangenoperationen ist  $\mathbf{O}(|V|)$



## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### Beweis

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange
- $\Rightarrow$  Zeit für Schlangenoperationen ist  $\mathbf{O}(|V|)$
- Adjazenzliste jedes Knotens wird nur durchlaufen, wenn er aus der Schlange entfernt wird

## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### Beweis

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange
- $\Rightarrow$  Zeit für Schlangenoperationen ist  $\mathbf{O}(|V|)$
- 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:  $\mathbf{O}(|V| + |E|)$

## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### Beweis

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange
- $\Rightarrow$  Zeit für Schlangenoperationen ist  $\mathbf{O}(|V|)$
- 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:  $\mathbf{O}(|V| + |E|)$
- **Gesamtlaufzeit:  $\mathbf{O}(|V| + |E|)$**

## Graphalgorithmen

### Satz 44

Sei  $G = (V, E)$  ein Graph. Die Laufzeit des Algorithmus BFS beträgt  $\mathbf{O}(|V| + |E|)$ .

### Beweis

- Laufzeit Initialisierung:  $\mathbf{O}(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange
- $\Rightarrow$  Zeit für Schlangenoperationen ist  $\mathbf{O}(|V|)$
- 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:  $\mathbf{O}(|V| + |E|)$
- Gesamtlaufzeit:  $\mathbf{O}(|V| + |E|)$

## 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 45*

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

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

## Graphalgorithmen

### *Lemma 45*

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  $v$

## Graphalgorithmen

### *Lemma 45*

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.



## Graphalgorithmen

### Lemma 45

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 45*

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 46*

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 in  $Q$  eingefügt wird

## Graphalgorithmen

### *Lemma 46*

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 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  $v \in V - \{s\}$

## Graphalgorithmen

### *Lemma 46*

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 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

## Graphalgorithmen

### *Lemma 46*

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 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 den nächsten weißen Knoten  $v$ , der während einer Suche von  $u$  entdeckt wird. Nach (I.V.) gilt  $d[u] \geq \delta(s, u)$ .

## Graphalgorithmen

### *Lemma 46*

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 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 den nächsten 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

## Graphalgorithmen

### Lemma 46

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 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 den nächsten 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 45



## Graphalgorithmen

### Lemma 46

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 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 den nächsten 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 45

## Graphalgorithmen

### *Lemma 46*

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*

- Knoten  $v$  wird dann in die Schlange eingefügt und grau gefärbt

## Graphalgorithmen

### *Lemma 46*

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*

- 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 46*

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*

- 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 47*

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

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

## Graphalgorithmen

### *Lemma 47*

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

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

### *Beweis*

- Induktion über die Anzahl Schlangenoperationen

## Graphalgorithmen

### Lemma 47

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

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

### Beweis

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

## Graphalgorithmen

### Lemma 47

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

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

### 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



## Graphalgorithmen

### Lemma 47

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

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

### 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 47

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

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

### 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 47*

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

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

### *Beweis*

- dequeue:
- Wird  $v_1$  aus der Schlange entfernt, so wird  $v_2$  der neue Kopf

## Graphalgorithmen

### *Lemma 47*

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

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

### *Beweis*

- dequeue:
- Wird  $v_1$  aus der Schlange entfernt, so wird  $v_2$  der neue Kopf
- Dann gilt aber sicherlich  $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$

## Graphalgorithmen

### *Lemma 47*

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

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

### *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 47*

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

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

### *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 47*

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

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

### *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

## Graphalgorithmen

### Lemma 47

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

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

### 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:  $d[v_{r+1}] = d[v] = d[u] + 1 = d[v_1] + 1$



## Graphalgorithmen

### Lemma 47

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

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

### 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:  $d[v_r] \leq d[v_1] + 1 = d[u] + 1 = d[v] = d[v_{r+1}]$

## Graphalgorithmen

### Lemma 47

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

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

### 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

### *Lemma 47*

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

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

### *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 48

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.

## Graphalgorithmen

### Satz 48

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  $s$ ):

## Graphalgorithmen

### Satz 48

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 46 gilt  $d[v] \geq \delta(s, v) = \infty$

## Graphalgorithmen

### Satz 48

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 46 gilt  $d[v] \geq \delta(s, v) = \infty$
- Es kann keinen ersten Knoten geben, dessen  $d$ -Wert in Zeile 7 auf  $\infty$  gesetzt wird

## Graphalgorithmen

### Satz 48

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 46 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

### Satz 48

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 46 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  $s$ ):

## Graphalgorithmen

### *Beweis*

- Fall 2 ( $v$  erreichbar von  $s$ ):
- Sei  $V_k = \{v \in V : \delta(s, v) = k\}$

## 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

## 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

- 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.

## 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  $V_{k-1}$



## 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.

## 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

## 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 47 sind die  $d$ -Werte monoton steigend, wenn Knoten in die Schlange eingefügt werden

## 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 47 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 46) und (I.V.): wenn  $v$  entdeckt wird, dann erst nachdem alle Knoten aus  $V_{k-1}$  in die Schlange eingefügt wurden

## Graphalgorithmen

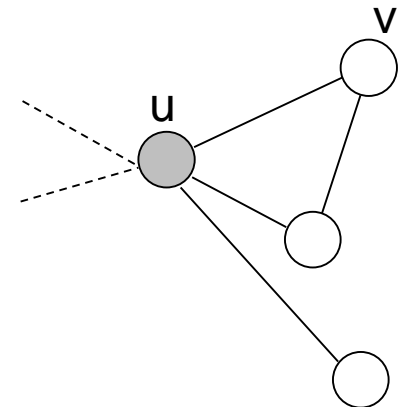
### Beweis

- (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 47 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 46) und (I.V.): wenn  $v$  entdeckt wird, dann erst nachdem alle Knoten aus  $V_{k-1}$  in die Schlange eingefügt wurden

## Graphalgorithmen

### Beweis

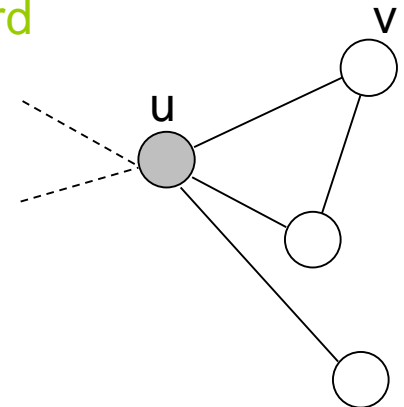
- Da  $\delta(s, v) = k$  gibt es Pfad mit  $k$  Kanten von  $s$  nach  $v$  und Knoten  $u$  mit  $(u, v) \in E$  und  $u \in V_{k-1}$



## Graphalgorithmen

### Beweis

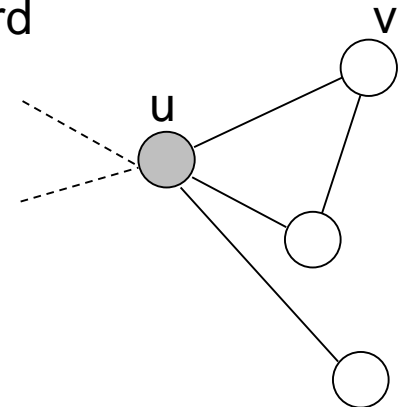
- Da  $\delta(s, v) = k$  gibt es Pfad mit  $k$  Kanten von  $s$  nach  $v$  und Knoten  $u$  mit  $(u, v) \in E$  und  $u \in V_{k-1}$
- OBdA. Sei  $u$  der erste solche Knoten, der grau gefärbt wird (existiert wegen I.V.)



## 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

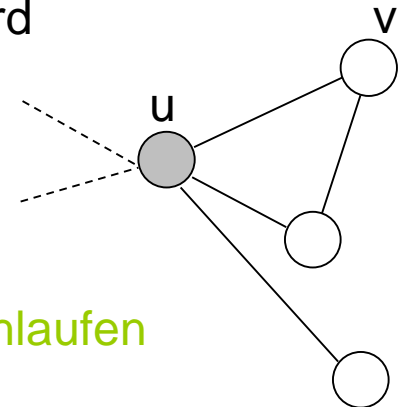




## Graphalgorithmen

### Beweis

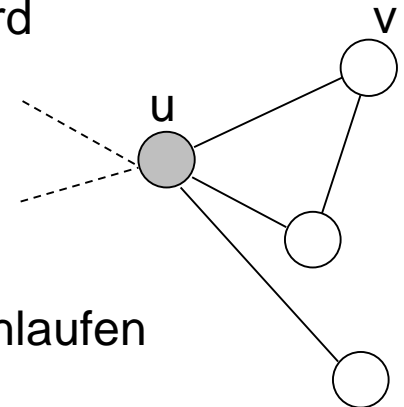
- Da  $\delta(s, v) = k$  gibt es Pfad mit  $k$  Kanten von  $s$  nach  $v$  und Knoten  $u$  mit  $(u, v) \in E$  und  $u \in V_{k-1}$
- OBdA. Sei  $u$  der erste solche Knoten, der grau gefärbt wird (existiert wegen I.V.)
- Wird Knoten grau gefärbt, so wird er auch in Schlange eingefügt und muss irgendwann als Kopf der Schlange auftauchen
- Ist  $u$  Kopf der Schlange, so wird seine Adjazenzliste durchlaufen und  $v$  in Zeile 4 entdeckt



## Graphalgorithmen

### Beweis

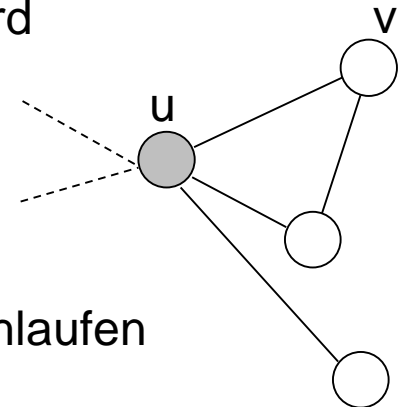
- Da  $\delta(s, v) = k$  gibt es Pfad mit  $k$  Kanten von  $s$  nach  $v$  und Knoten  $u$  mit  $(u, v) \in E$  und  $u \in V_{k-1}$
- OBdA. Sei  $u$  der erste solche Knoten, der grau gefärbt wird (existiert wegen I.V.)
- Wird Knoten grau gefärbt, so wird er auch in Schlange eingefügt und muss irgendwann 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  $\pi[v] = u$ .



## Graphalgorithmen

### Beweis

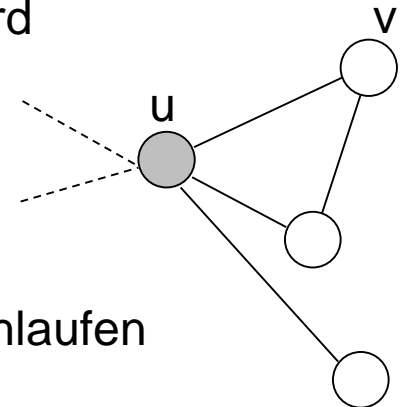
- Da  $\delta(s, v) = k$  gibt es Pfad mit  $k$  Kanten von  $s$  nach  $v$  und Knoten  $u$  mit  $(u, v) \in E$  und  $u \in V_{k-1}$
- OBdA. Sei  $u$  der erste solche Knoten, der grau gefärbt wird (existiert wegen I.V.)
- Wird Knoten grau gefärbt, so wird er auch in Schlange eingefügt und muss irgendwann 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



## Graphalgorithmen

### Beweis

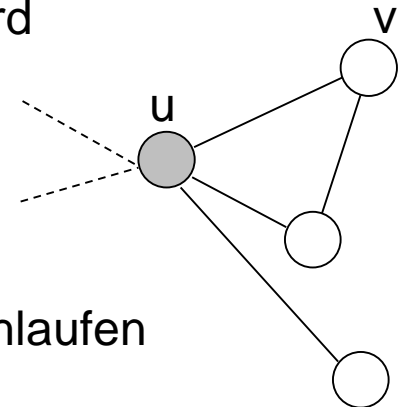
- Da  $\delta(s, v) = k$  gibt es Pfad mit  $k$  Kanten von  $s$  nach  $v$  und Knoten  $u$  mit  $(u, v) \in E$  und  $u \in V_{k-1}$
- OBdA. Sei  $u$  der erste solche Knoten, der grau gefärbt wird (existiert wegen I.V.)
- Wird Knoten grau gefärbt, so wird er auch in Schlange eingefügt und muss irgendwann 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

- 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  $V_{k-1}$

## 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

### *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  $\mathbf{O}(|V| + |E|)$