

# Algorithmen und Datenstrukturen (für ET/IT)

Sommersemester 2018

Dr. Stefanie Demirci

Computer Aided Medical Procedures  
Technische Universität München



Notizen

---

---

---

---

---

---

---

---

---

---

---

---

## Programm heute

- ⑦ Fortgeschrittene Datenstrukturen
  
- ⑧ Such-Algorithmen
  
- ⑨ Graph-Algorithmen
  - Tiefensuche
  - Breitensuche

Notizen

---

---

---

---

---

---

---

---

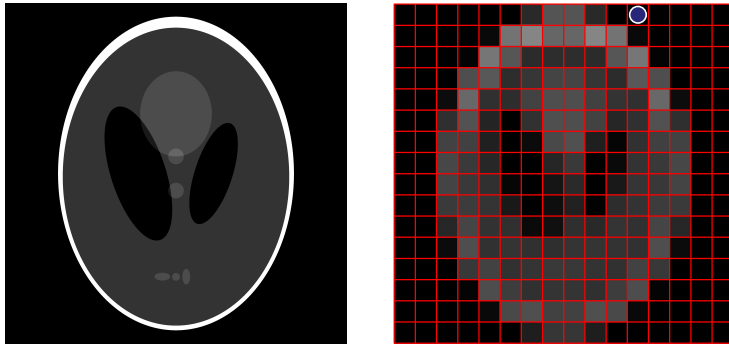
---

---

---

---

## Bild-Repräsentation mit Pixeln



- Bild als **zwei-dimensionales Feld**
- einzelnes Element: **Pixel** (*picture element*)
- hier: jedem Pixel wird ein **Grauwert** zugewiesen
  - z.B. von 0 bis 255

Notizen

---

---

---

---

---

---

---

---

---

---

9

## Segmentierung von Bildern

- **Segmentierung**: Zusammenfassung von inhaltlich zusammenhängenden Regionen gemäß eines Kriteriums



- Beispiel: Segmentierung von Knochen oder Organen

Notizen

---

---

---

---

---

---

---

---

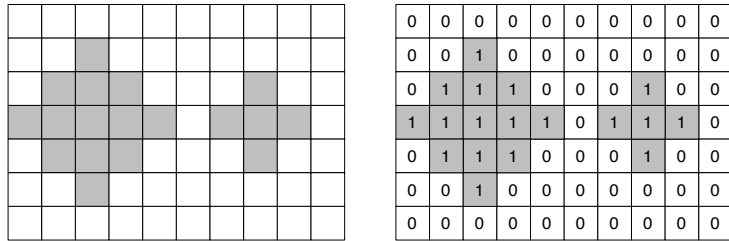
---

---

10

# Binäre Bilder

- **Binäres Bild:** jedes Pixel hat entweder Wert 0 oder 1



- **Segmentierungs-Problem:** identifiziere die beiden "Sterne"  
→ Algorithmus?

Notizen

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

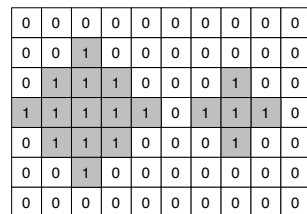
---

---

---

---

# Problem: Segmentierung von "Sternen"



- **Gegeben:** binäres Bild mit 0,1 Werten
- **Gesucht:** Regionen im Bild mit Wert 1
- **Ansatz:**
  - fange an bei beliebigem Pixel mit Wert 1
  - wiederhole: inspiziere unbesuchte Nachbar-Pixel auf Wert 1
  - beende, falls kein weiterer Nachbar mit Wert 1 existiert

Notizen

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

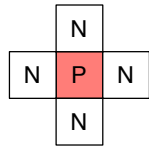
---

---

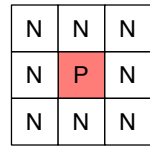
---

---

# Nachbarschaft von Pixeln



4-Nachbarschaft



8-Nachbarschaft

- **4-Nachbarschaft** von Pixel P: alle direkten Nachbar-Pixel, die Kante mit P gemeinsam haben
  - auch genannt: Von-Neumann-Nachbarschaft
- **8-Nachbarschaft** von Pixel P: alle direkten Nachbar-Pixel, die Kante oder Ecke mit P gemeinsam haben
  - auch genannt: Moore-Nachbarschaft

Notizen

---

---

---

---

---

---

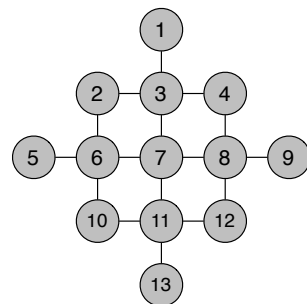
---

---

---

---

# “Stern” als Graph



Segmentierungs-Ansatz von linkem “Stern”:

- 1 starte mit Pixel 1
- 2 besuche Nachbarn (Schema: oben, unten, links, rechts)

Notizen

---

---

---

---

---

---

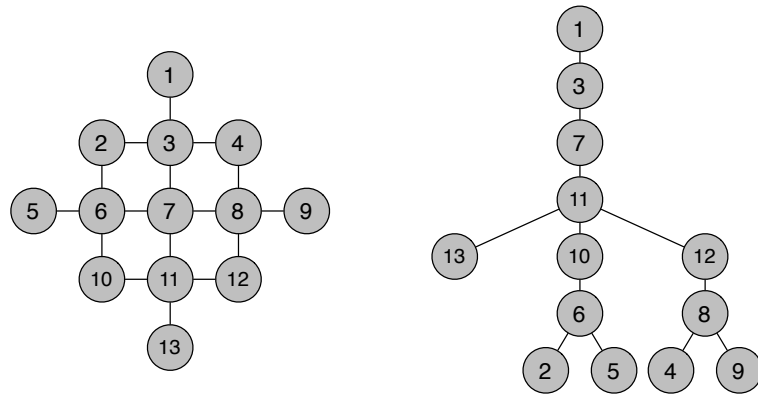
---

---

---

---

# Segmentierungs-Durchlauf als Baum



Ansatz auch genannt: **Tiefensuche** bzw. **Depth-First Search (DFS)**

Notizen

---

---

---

---

---

---

---

---

---

---

# Tiefensuche (DFS)

Sei  $G = (V, E)$  Graph (gerichtet oder ungerichtet).

- Graph repräsentiert mit Adjazenzliste **adj**
- jeder Knoten hat Farb-Markierung **farbe**:
  - **weiss** = noch nicht besucht
  - **grau** = besucht, gerade in Bearbeitung
  - **schwarz** = besucht, fertig bearbeitet
- jeder Knoten hat Vorgänger in Besuch-Reihenfolge **pred**
- (optional) zusätzliche Knoten-Markierungen **d** und **f**
  - Zeitmarke, wann Knoten **entdeckt**: **d** (discovered)
  - Zeitmarke, wann Knoten **fertig bearbeitet**: **f** (finalized)

Notizen

---

---

---

---

---

---

---

---

---

---

# Algorithmus DFS

**Input:** Graph  $G = (V, E)$

**Output:** Vorgänger-Liste `pred`,  
Markierungen `d`, `f`

**DFS( $G$ ):**

```
for each (Knoten  $v \in V$ ) {  
    farbe[v] = weiss;  
    pred[v] = null;  
}  
zeit = 0;  
for each (Knoten  $v \in V$ ) {  
    if (farbe[v] == weiss)  
        DFSvisit(v);  
}
```

**DFSvisit( $v$ ):**

```
farbe[v] = grau; // v war weiss  
zeit = zeit + 1;  
d[v] = zeit;  
for each (Knoten  $u \in \text{adj}[v]$ ) {  
    if (farbe[u] == weiss) {  
        pred[u] = v;  
        DFSvisit(u);  
    }  
}  
farbe[v] = schwarz; // v ist fertig  
zeit = zeit + 1;  
f[v] = zeit;
```

Notizen

---

---

---

---

---

---

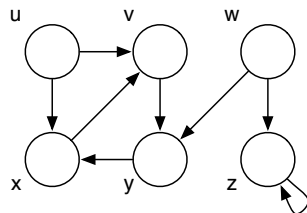
---

---

---

---

## DFS: Beispiel-Ablauf 1



- Knoten `u`, `v`, `w`, `x`, `y`, `z`
- `farbe` direkt im Knoten markiert
- `d/f` im Knoten notiert
- `pred` markiert durch rote Kanten

Notizen

---

---

---

---

---

---

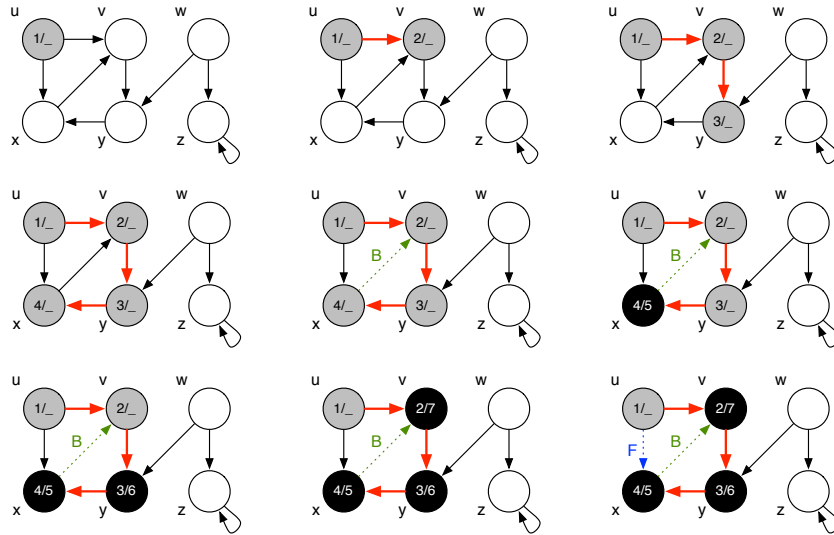
---

---

---

---

## DFS: Beispiel-Ablauf 2



Notizen

---

---

---

---

---

---

---

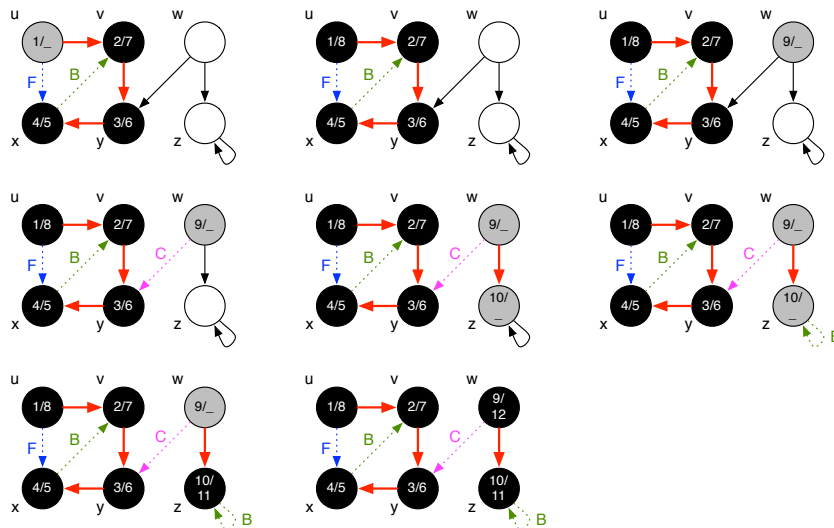
---

---

---

19

## DFS: Beispiel-Ablauf 3



Notizen

---

---

---

---

---

---

---

---

---

---

20

## Spannwald mit DFS

Sei  $G = (V, E)$  Graph.

- **Spannbaum:** Teilgraph  $G' = (V, E')$ , der ein Baum ist und alle Knoten von  $G$  enthält.
  - existiert nur für zusammenhängende Graphen
- **Spannwald:** Teilgraph  $G' = (V, E')$ , dessen Zusammenhangskomponenten jeweils Spann bäume der Zusammenhangskomponenten von  $G$  sind
- DFS erzeugt Spannwald
  - Knoten sind  $V$ , Kanten  $E'$  ergeben sich aus **pred**
  - im Beispiel-Ablauf: Knoten  $V$  sind schwarz, Kanten  $E'$  sind **rot**
- DFS erlaubt Markierung spezieller Kanten:
  - Rückkanten im Spannwald (markiert mit **B**)
  - Vorwärtskanten im Spannwald (markiert mit **F**)
  - Cross-Kanten im Spannwald (markiert mit **C**)

Notizen

---

---

---

---

---

---

---

---

---

---

21

## Komplexität von DFS

- **DFS** erste Schleife wird  $|V|$  mal durchlaufen:  $\Theta(|V|)$
- **DFSvisit** wird für jeden Knoten **genau** einmal aufgerufen (zweite Schleife in **DFS**, rekursiver Aufruf in **DFSvisit**)
- innere Schleife in **DFSvisit** wird  $|\text{adj}(v)|$  mal aufgerufen

$$\sum_{v \in V} |\text{adj}(v)| = \Theta(|E|)$$

- Gesamtlaufzeit:  $\Theta(|V| + |E|)$
- Implementierung gleicher Komplexität auch nicht-rekursiv möglich (mit Stack)

Notizen

---

---

---

---

---

---

---

---

---

---

22



# Anwendungen von DFS

- Test auf **Zusammenhang** von Graphen
  - rufe **DFSvisit** nur für einen Knoten auf (statt für alle)
  - falls nicht alle Knoten dadurch besucht: nicht zusammenhängend!
- Test auf **Zyklenfreiheit** in Graphen
  - Zyklus entdeckt falls Rückkante **B** gefunden
  - Rückkante **B**: in **DFSvisit** Schleife ist Zielknoten bereits grau

Notizen

---

---

---

---

---

---

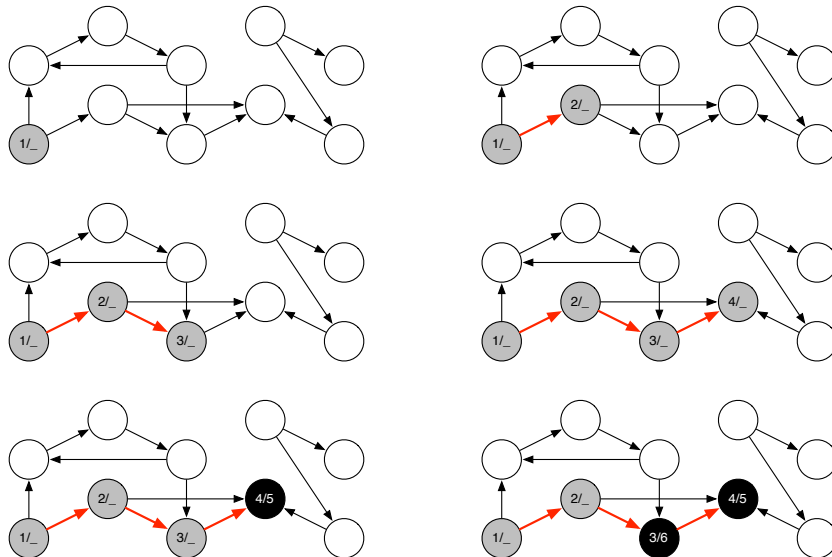
---

---

---

---

## Beispiel DFS 1



Notizen

---

---

---

---

---

---

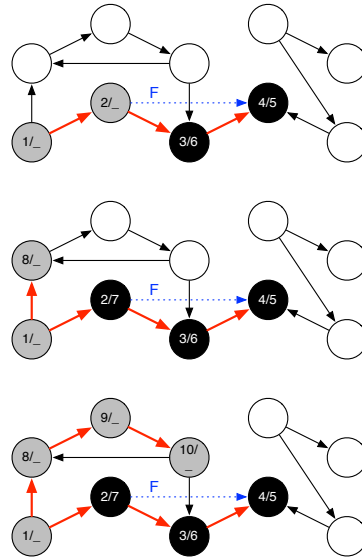
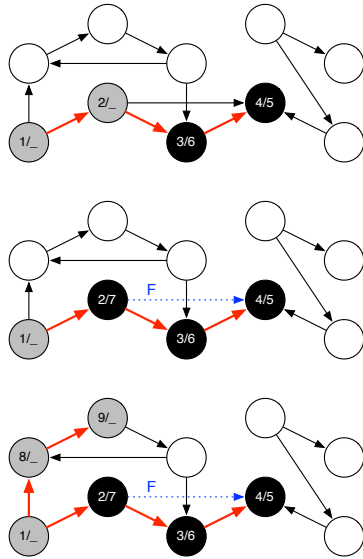
---

---

---

---

## Beispiel DFS 2



25

Notizen

---

---

---

---

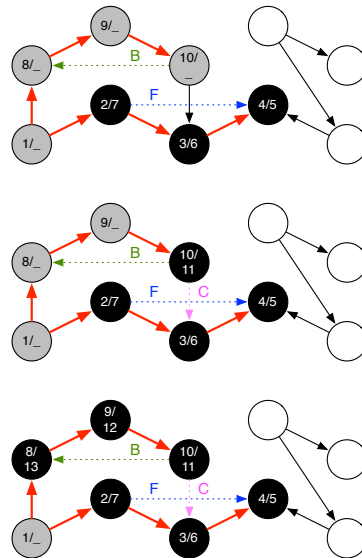
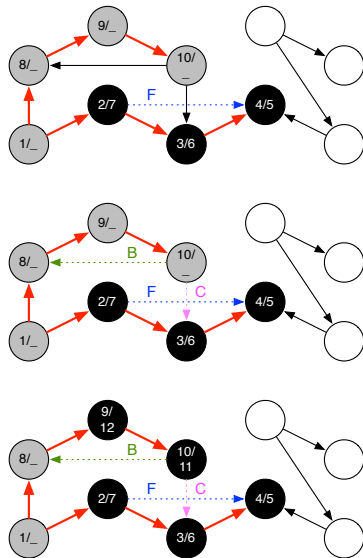
---

---

---

---

## Beispiel DFS 3



26

Notizen

---

---

---

---

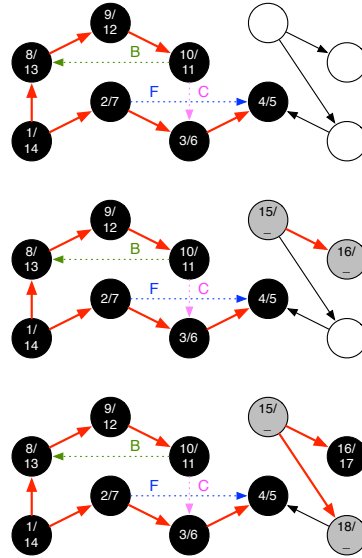
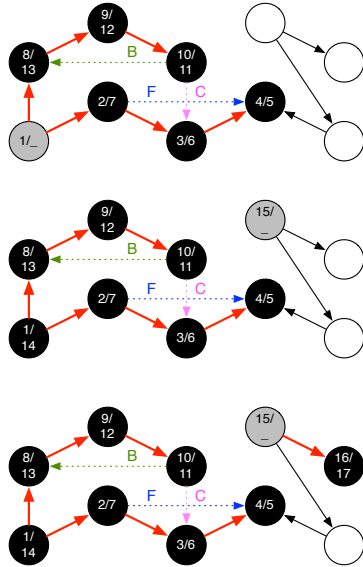
---

---

---

---

## Beispiel DFS 4



27

Notizen

---

---

---

---

---

---

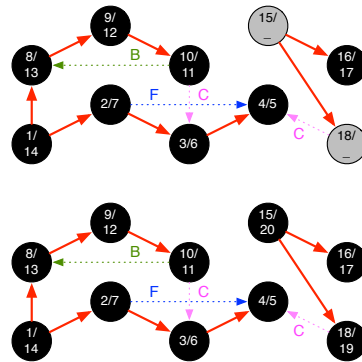
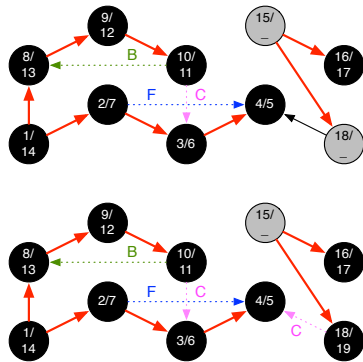
---

---

---

---

## Beispiel DFS 5



28

Notizen

---

---

---

---

---

---

---

---

---

---

# Programm heute

- 7 Fortgeschrittene Datenstrukturen
- 8 Such-Algorithmen
- 9 Graph-Algorithmen
  - Tiefensuche
  - Breitensuche

Notizen

---

---

---

---

---

---

---

---

---

---

## Breitensuche im "Stern"-Beispiel

- Segmentierung von "Sternen"

0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	1	0	0
1	1	1	1	1	0	1	1	1	0
0	1	1	1	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

- neuer Ansatz:
    - anstatt direkt nächsten Nachbar ansteuern, erst alle aktuellen Nachbarn abarbeiten
- Breitensuche

Notizen

---

---

---

---

---

---

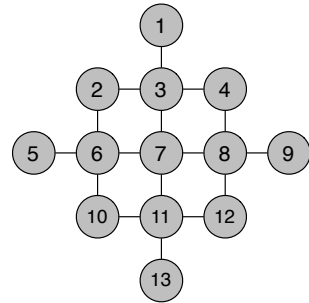
---

---

---

---

# Breitensuche im "Stern"-Beispiel: Ablauf



- 1 starte mit Pixel 1
- 2 besuche alle Nachbarn von aktuellem Pixel (Schema: oben, unten, links, rechts)
- 3 arbeite die Nachbarn der Nachbarn in selber Reihenfolge ab

Notizen

---

---

---

---

---

---

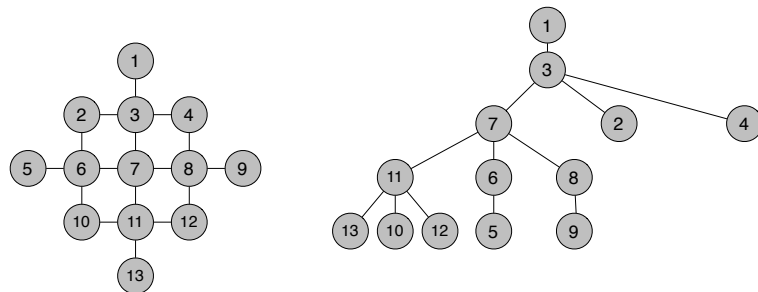
---

---

---

---

# Breitensuche im "Stern"-Beispiel: als Baum



Ansatz: **Breitensuche** bzw. **Breadth-First Search (BFS)**

Notizen

---

---

---

---

---

---

---

---

---

---

## Breitensuche (BFS)

Sei  $G = (V, E)$  zusammenhängender Graph (gerichtet oder ungerichtet).

- Startknoten  $s \in V$
- Graph repräsentiert mit Adjazenzliste  $adj$
- jeder Knoten hat Farb-Markierung  $farbe$ :
  - $weiss$  = noch nicht besucht
  - $grau$  = besucht, gerade in Bearbeitung
  - $schwarz$  = besucht, fertig bearbeitet
- jeder Knoten hat Vorgänger in Besuch-Reihenfolge  $pred$
- (optional) zusätzliche Knoten-Markierung  $d$ 
  - Distanz (in Anzahl von Kanten) von Startknoten  $s$
- Hilfsmittel: Queue  $Q$

Notizen

---

---

---

---

---

---

---

---

---

---

33

## Algorithmus BFS

**Input:** Graph  $G = (V, E)$ , Startknoten  $s \in V$

**Output:** Vorgänger-Liste  $pred$ , Markierung  $d$

**BFS**( $G, s$ ):

```
for each (Knoten  $v \in V$ ) { // Initialisierung
   $farbe[v] = weiss$ ;  $pred[v] = null$ ;  $d[v] = \infty$ ;
}
 $farbe[s] = grau$ ;  $d[s] = 0$ ;
 $Q = initialize()$ ;  $Q.enqueue(s)$ ;
while (  $!Q.isEmpty()$  ) {
   $u = Q.dequeue()$ ;
  for each ( $v \in adj[u]$ ) { // besuche alle Nachbarn
    if ( $farbe[v] == weiss$ ) {
       $farbe[v] = grau$ ;  $d[v] = d[u] + 1$ ;  $pred[v] = u$ ;
       $Q.enqueue(v)$ ;
    }
  }
   $farbe[u] = schwarz$ ; //  $u$  ist erledigt
}
```

Notizen

---

---

---

---

---

---

---

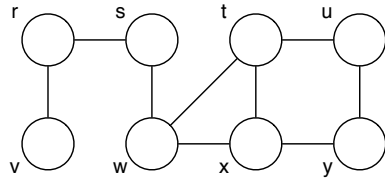
---

---

---

34

## BFS: Beispiel-Ablauf 1



- Knoten r, s, t, u, v, w, x, y
- farbe direkt im Knoten markiert
- d im Knoten notiert
- pred markiert durch rote Kanten
- Startknoten s

Notizen

---

---

---

---

---

---

---

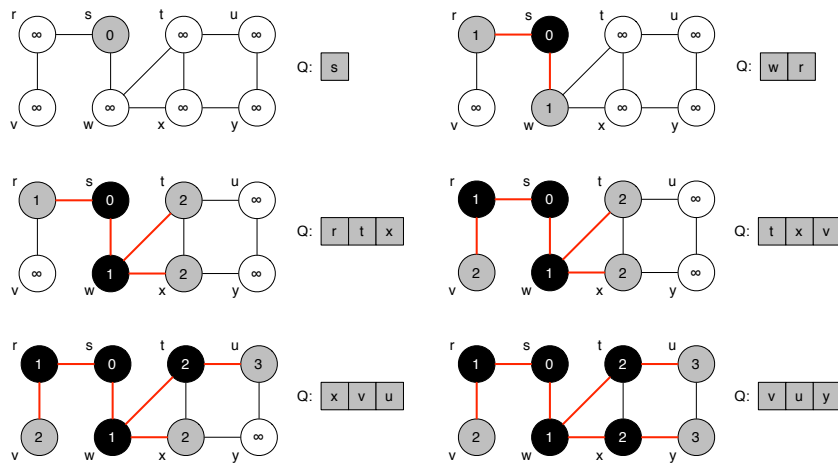
---

---

---

35

## BFS: Beispiel-Ablauf 2



Notizen

---

---

---

---

---

---

---

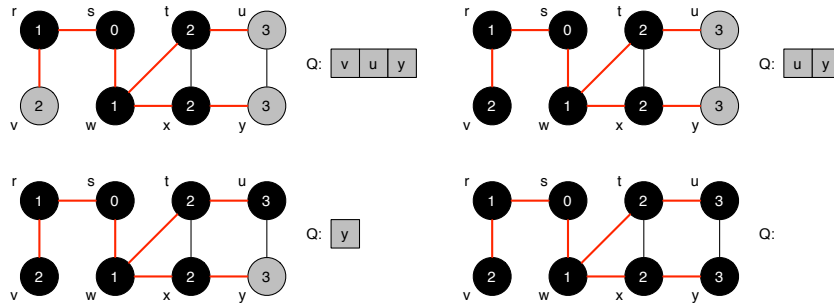
---

---

---

36

## BFS: Beispiel-Ablauf 3



Notizen

---

---

---

---

---

---

---

---

---

---

---

---

37

## BFS Eigenschaften

Sei  $G = (V, E)$  Graph.

- BFS erzeugt Spannbaum  $G' = (V, E')$ 
  - Knoten sind  $V$ , Kanten  $E'$  ergeben sich aus **pred**
  - im Beispiel-Ablauf: Knoten  $V$  sind schwarz, Kanten  $E'$  sind **rot**
  
- ausgehend von Startknoten  $s$  wird nur Zusammenhangskomponente von Graph durchsucht!
- falls  $G$  nicht zusammenhängend, wird **kein** Spannwald berechnet!
  
- falls Stack statt Queue in BFS: DFS-artiger Algorithmus
  - Unterschied: nur in Zusammenhangskomponente

Notizen

---

---

---

---

---

---

---

---

---

---

---

---

38



## Komplexität von BFS

- **BFS** erste Schleife (for) wird  $|V|$  mal durchlaufen:  $\Theta(|V|)$
- zweite geschachtelte Schleife (while und for) wird im schlechtesten Fall je einmal für jeden Knoten in Adjazenzliste aufgerufen

$$\sum_{v \in V} |\text{adj}(v)| = \Theta(|E|)$$

- Gesamtlaufzeit:  $\Theta(|V| + |E|)$

Notizen

---

---

---

---

---

---

---

---

---

---

39

## Anwendungen von BFS

- Besuche alle Knoten in Zusammenhangskomponente von Graph
  - ausgehend von Startknoten  $s$
- Berechne Länge der kürzesten Pfade (d.h. Anzahl von Kanten)
  - ausgehend von Startknoten  $s$
  - kürzeste Pfade selbst erfordern Modifikation des Algorithmus

Notizen

---

---

---

---

---

---

---

---

---

---

40

# Zusammenfassung

7 Fortgeschrittene Datenstrukturen

8 Such-Algorithmen

9 Graph-Algorithmen

Tiefensuche

Breitensuche

Notizen

---

---

---

---

---

---

---

---

---

---

---

---

Notizen

---

---

---

---

---

---

---

---

---

---

---

---