Grundzüge der Informatik 1

Vorlesung 23

Überblick Vorlesung

Graphenalgorithmen

- Tiefensuche
  - Klammerstruktur der Tiefensuche
  - Satz vom weißen Weg
- Minimale Spannbäume
  - Definition
  - Die Kreiseigenschaft
  - Algorithmus von Kruskal

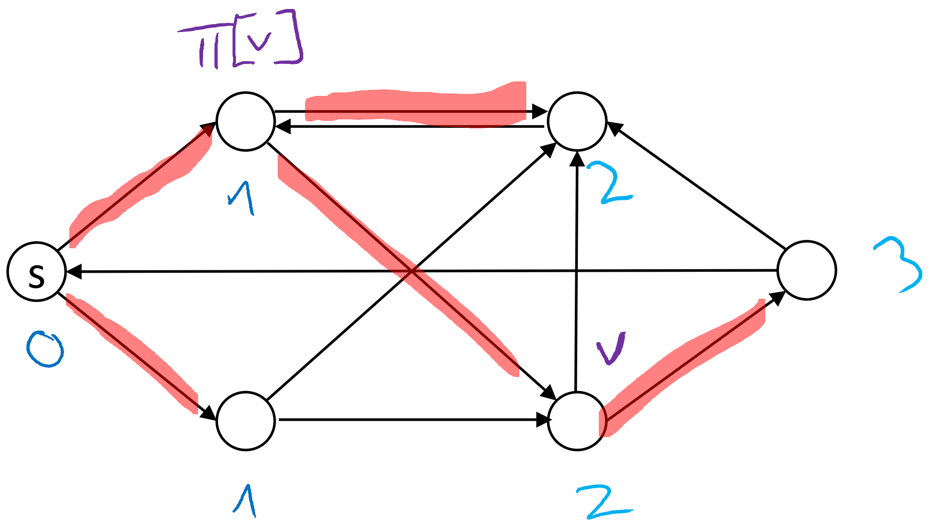
Graphenalgorithmen

Breitensuche

- Durchlauf verbundenen Graph von Startknoten s
- Berechne kürzeste Wege (Anzahl Kanten) von s zu anderen Knoten im Graph
- Eingabegraph in Adjazenzlistendarstellung
- Laufzeit $O(|V|+|E|)$

Graphenalgorithmen

Breitensuche



Graphenalgorithmen

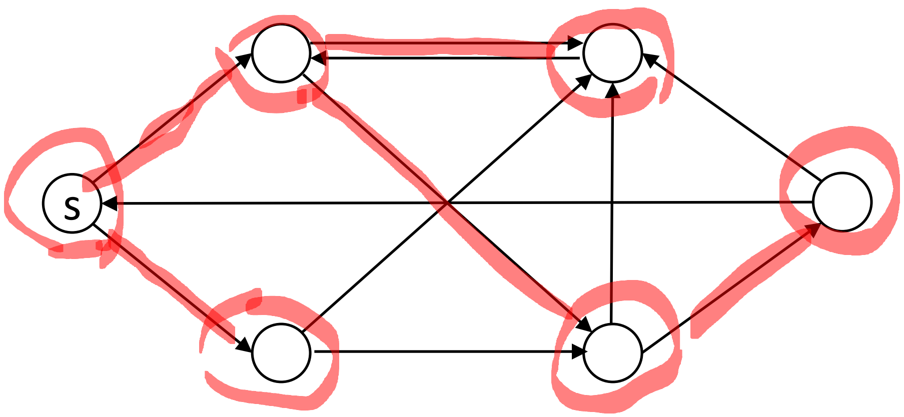
Tiefensuche

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

Universität zu Köln

Graphenalgorithmen

Tiefensuche



Graphenalgorithmen

Invariante Tiefensuche

- Zu Beginn: alle Knoten weiß
- Entdeckte Knoten werden zunächst grau
- Abgearbeitetete Knoten werden schwarz
- Zwei Zeitstempel: d[v] und f[v] (liegen zwischen 1 und 2|V|)
- d[v]: v ist entdeckt
- f[v]: v ist abgearbeitet

Graphenalgorithmen

Zeitstempel der Tiefensuche

- d[v] < f[v]
- Vor d[v] ist v weiß
- Zwischen d[v] und f[v] ist v grau
- Nach f[v] ist v schwarz

Graphenalgorithmen

DFS(G)

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

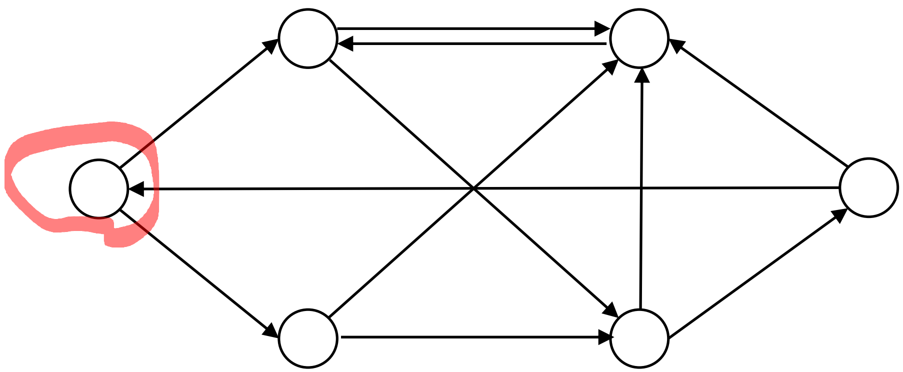
DFS-Visit(u)

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

Graphenalgorithmen

DFS(G)

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



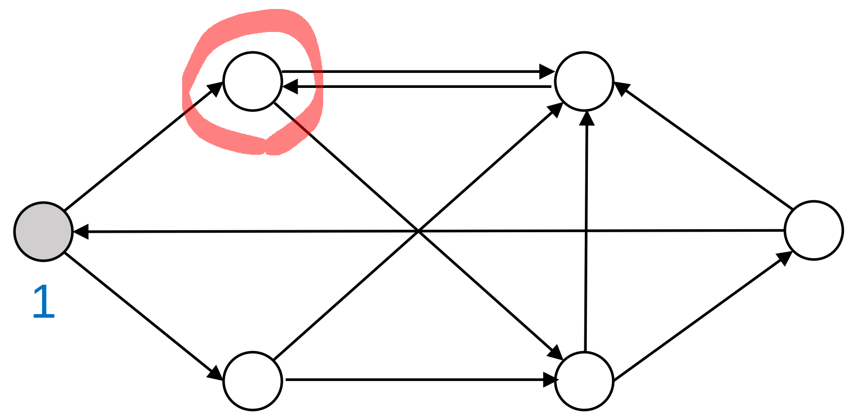
DFS-Visit(u)

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

Graphenalgorithmen

DFS(G)

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



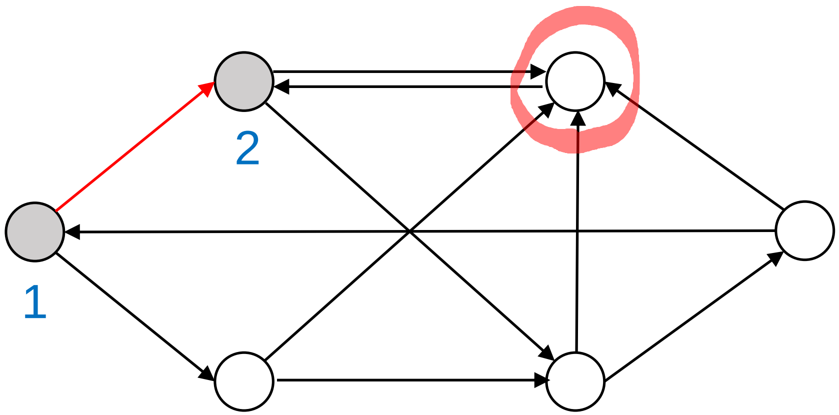
DFS-Visit(u)

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

Graphenalgorithmen

DFS(G)

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



DFS-Visit(u)

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

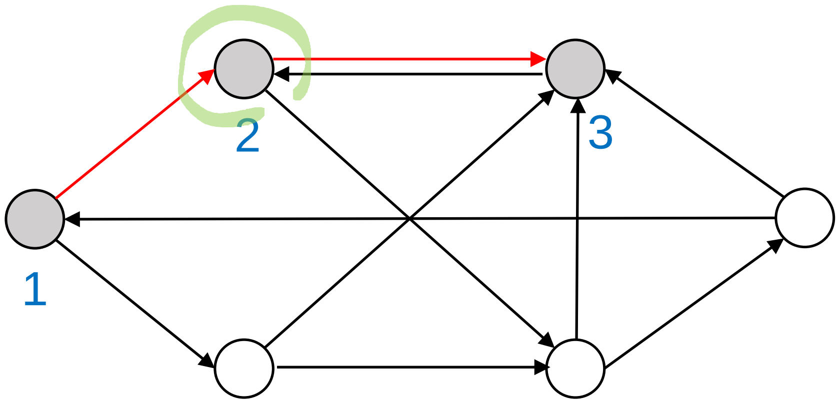
Graphenalgorithmen

DFS(G)

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

DFS-Visit(u)

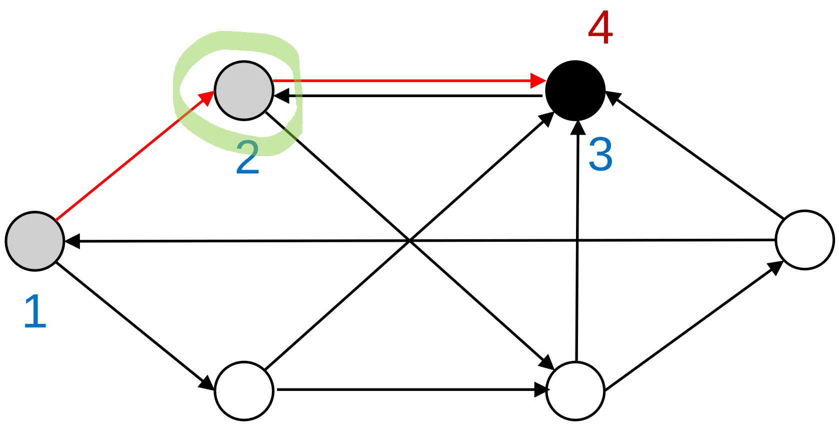
1. color[u] = grau
2. time = time +1; d[u] = time
3. **for each** v∈Adj[u] **do**
4. **if** color[v] = weiß **then** π[v] = u ; DFS-Visit(v)
5. color[u] = schwarz
6. time = time+1; f[u] = time



Graphenalgorithmen

DFS(G)

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



DFS-Visit(u)

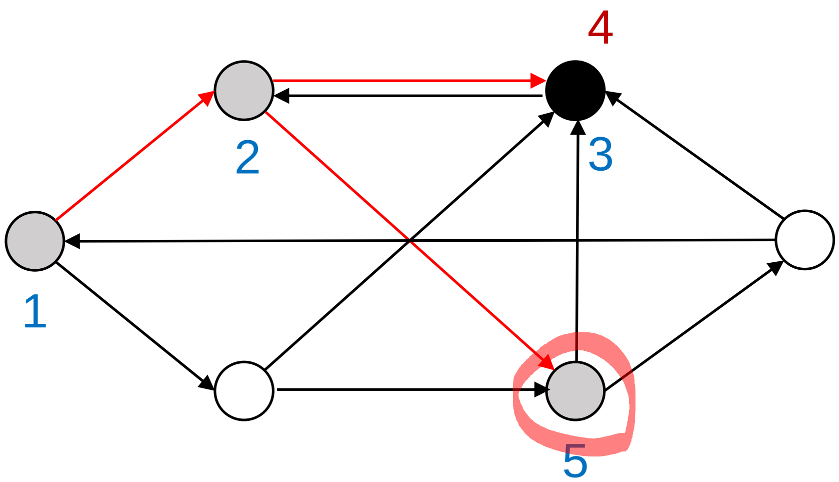
1. color[u] = grau
2. time = time +1; d[u] = time
3. **for each** v∈Adj[u] **do**
4. **if** color[v] = weiß **then** π[v] = u ; DFS-Visit(v)
5. color[u] = schwarz
6. time = time+1; f[u] = time

Universität zu Köln

Graphenalgorithmen

DFS(G)

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



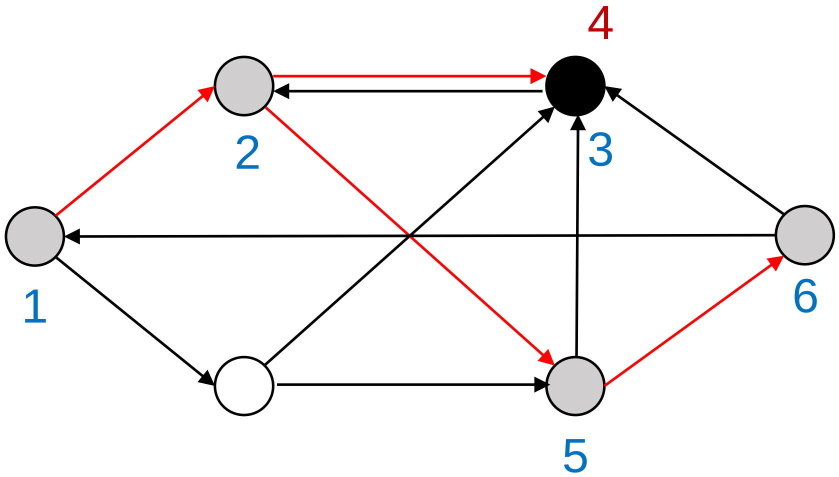
DFS-Visit(u)

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

Graphenalgorithmen

DFS(G)

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



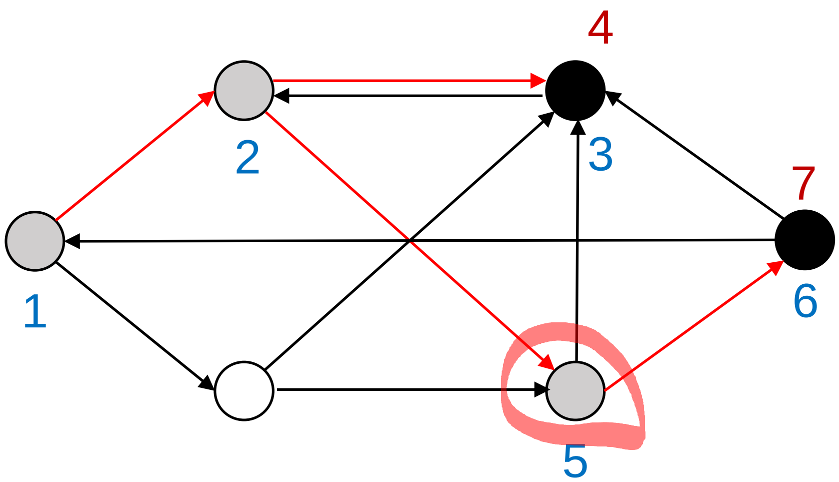
DFS-Visit(u)

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

Graphenalgorithmen

DFS(G)

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



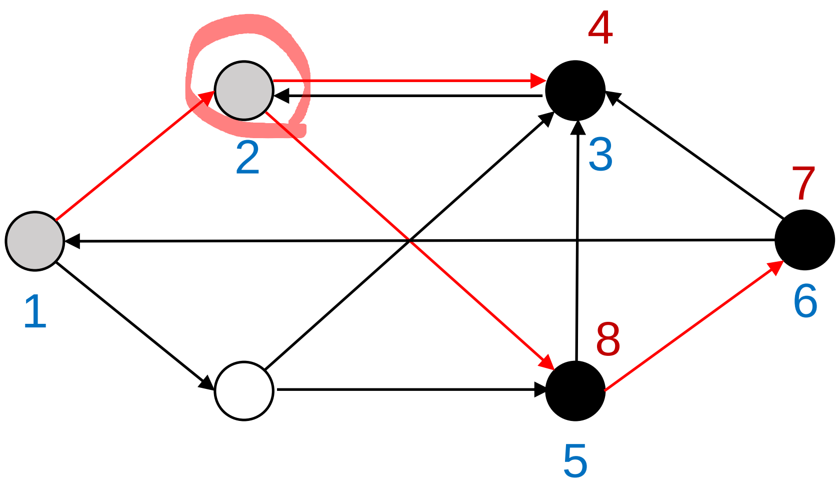
DFS-Visit(u)

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

Graphenalgorithmen

DFS(G)

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



DFS-Visit(u)

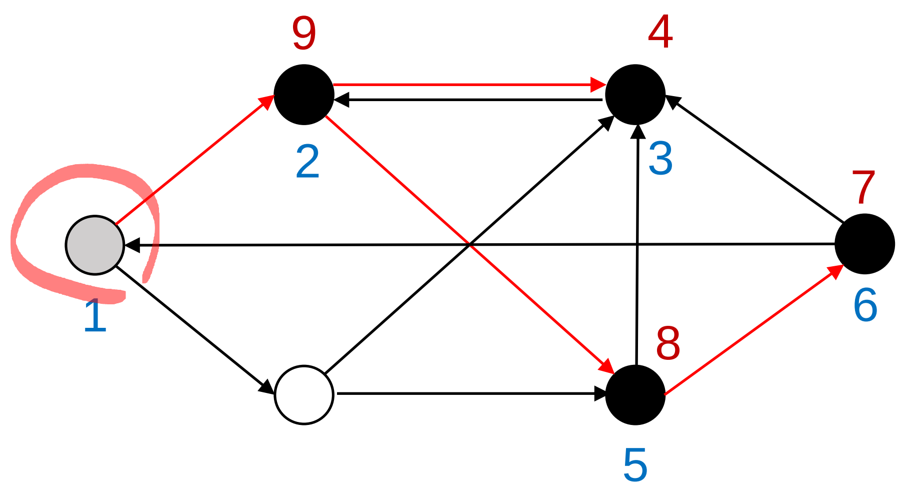
1. color[u] = grau
2. time = time +1; d[u] = time
3. **for each** v$\in$Adj[u] **do**
4.   **if** color[v] = weiß **then** π[v] = u ; DFS-Visit(v)
5. color[u] = schwarz
6. time = time+1; f[u] = time

Universität zu Köln

Graphenalgorithmen

DFS(G)

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



DFS-Visit(u)

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

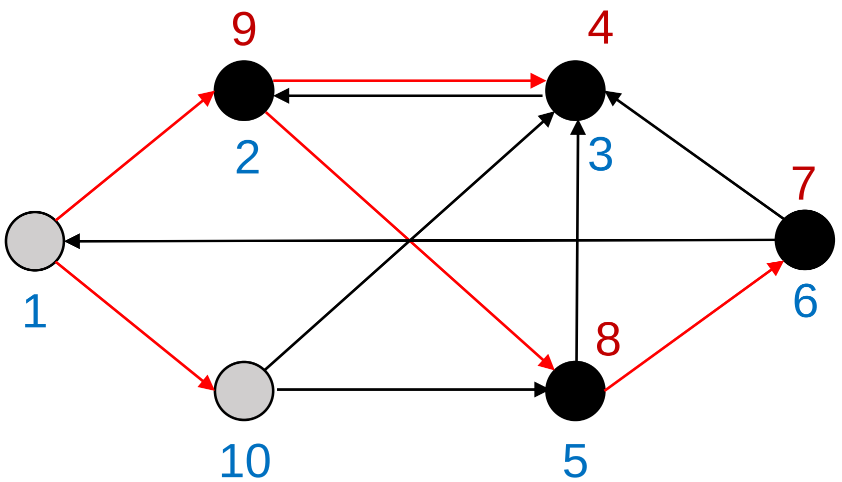
Graphenalgorithmen

DFS(G)

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

DFS-Visit(u)

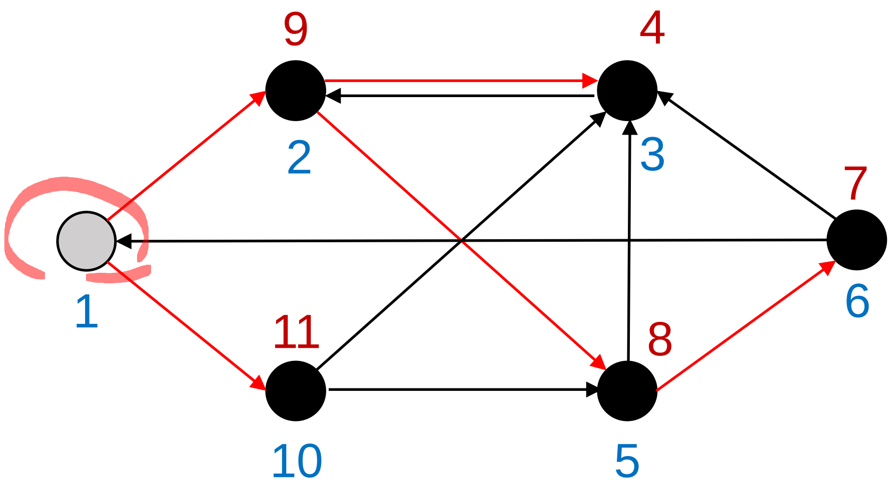
1. color[u] = grau
2. time = time +1; d[u] = time
3. **for each** $v \in$ Adj[u] **do**
4.   **if** color[v] = weiß **then** π[v] = u ; DFS-Visit(v)
5. color[u] = schwarz
6. time = time+1; f[u] = time



Graphenalgorithmen

DFS(G)

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



DFS-Visit(u)

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

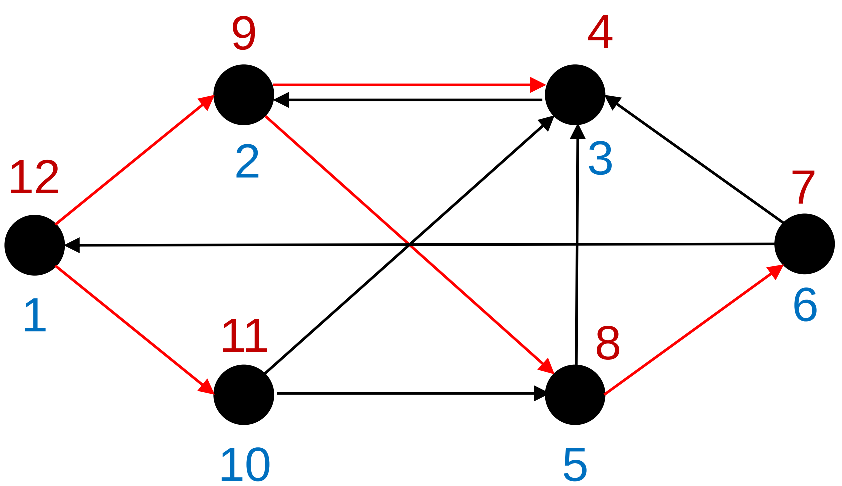
Graphenalgorithmen

DFS(G)

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

DFS-Visit(u)

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



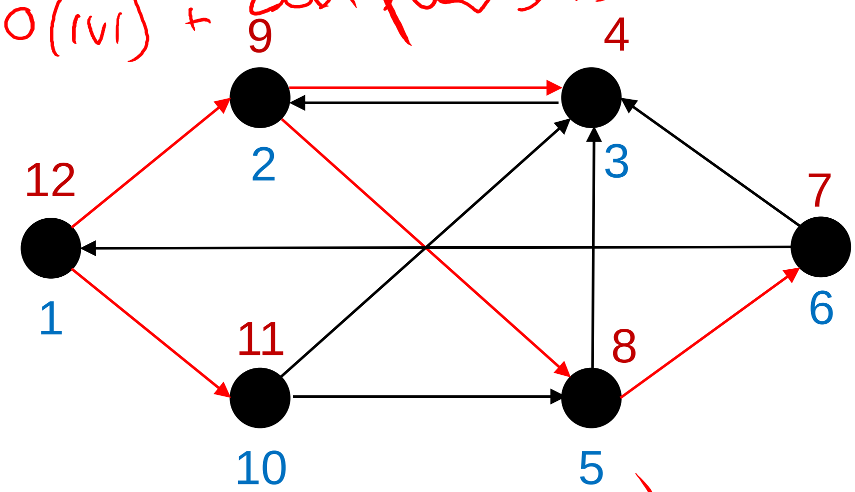
Graphenalgorithmen

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

DFS(G)

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

$\Big\}\; O(|V|)$ + Zeit für DFS-Visit



DFS-Visit(u)

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

$O(1+\deg(u))$

$O\Big(\sum_{u \in V} (1+\deg(u))\Big) = O(|V|+|E|)$

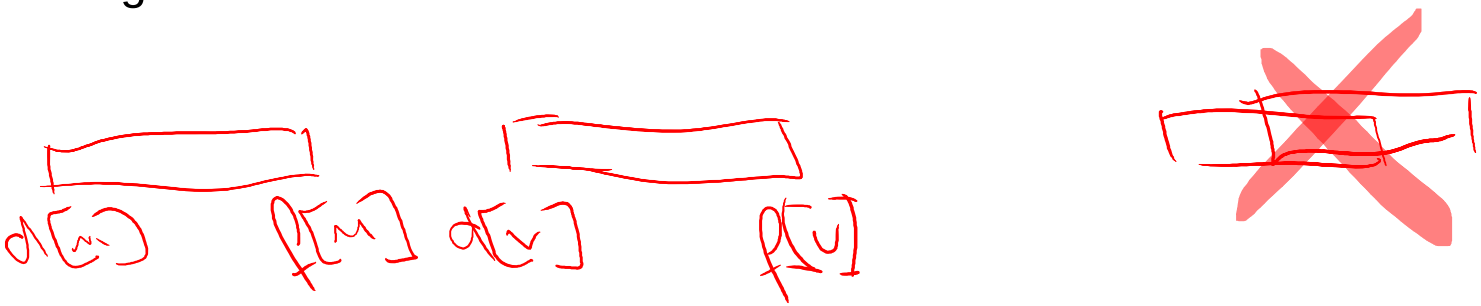
Universität zu Köln

Graphenalgorithmen

Satz 23.1 (Klammersatz zur Tiefensuche)

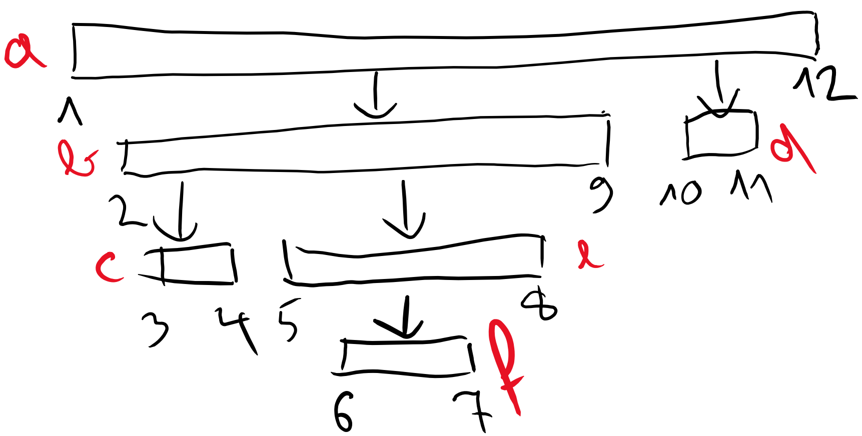
In jeder Tiefensuche eines gerichteten oder ungerichteten Graphen gilt für jeden Knoten u und v genau eine der folgenden drei Bedingungen:

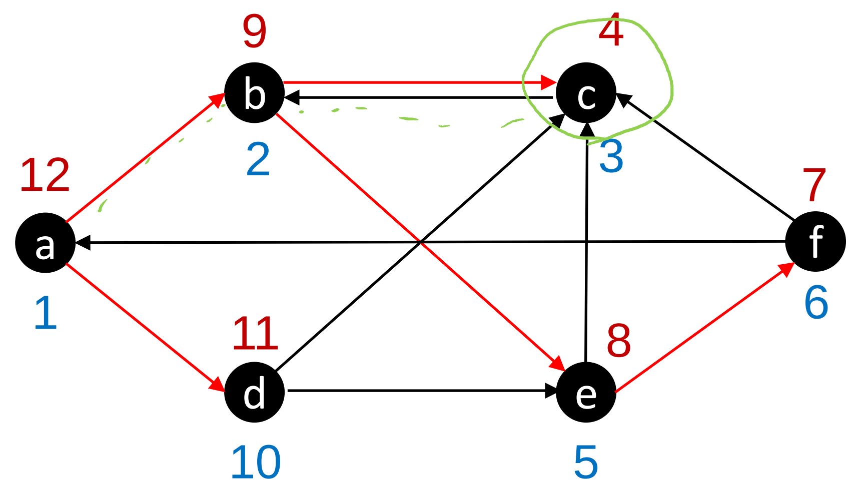
- Die Intervalle [d[u],f[u]] und [d[v],f[v]] sind vollständig disjunkt
- Intervall [d[u],f[u]] ist vollständig im Interval [d[v],f[v]] enthalten und u ist Nachfolger von v im DFS-Wald
- Intervall [d[v],f[v]] ist vollständig im Interval [d[u],f[u]] enthalten und v ist Nachfolger von u im DFS-Wald



Graphenalgorithmen

Beispiel





Korollar 23.2

Knoten v ist echter (u≠v) Nachfolger von Knoten u im DFS-Wald von G, gdw.
d[u]<d[v]<f[v]<f[u].

Graphenalgorithmen

Beweis

- Fall 1: d[u] < d[v].

$d[u] < d[v] < f[v] < f[u]$

- (a) d[v] < f[u]:
- v wurde entdeckt als u noch grau war.
- $\Rightarrow$ v Nachfolger von u
- Da v nach u entdeckt wurde, werden alle seine ausgehenden Kanten entdeckt und wird v abgearbeitet bevor die Suche zu u zurückkehrt und u abarbeitet

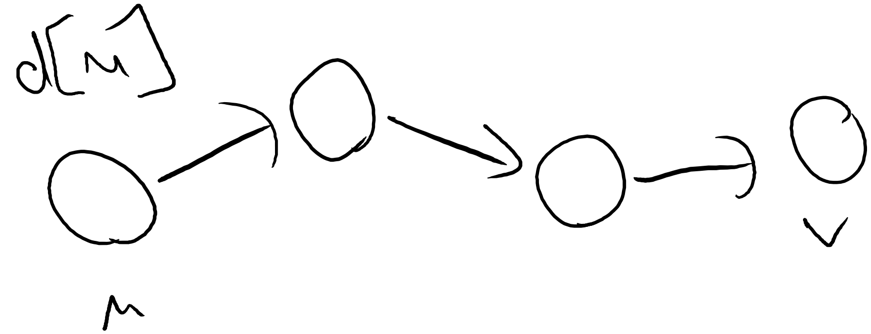
$f[v] < f[u]$

- Daher ist [d[v],f[v]] ist in [d[u],f[u]] enthalten
- (b) f[u] < d[v]: Dann sind die Intervalle disjunkt

$d[u] < f[u] < d[v] < f[v]$

- Fall 2: analog

Graphenalgorithmen



Satz 23.3 (Satz vom weißen Weg)

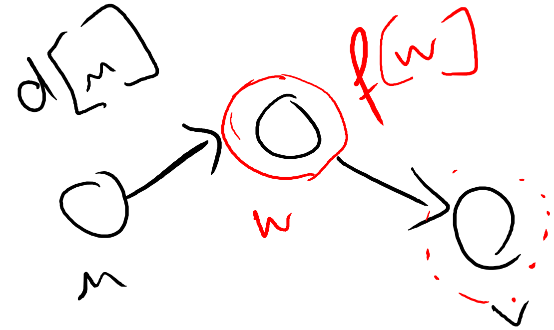
In einem DFS-Wald eines gerichteten oder ungerichteten Graph G ist Knoten v ein Nachfolger von Knoten u, gdw. zum Zeitpunkt d[u] v über einen Weg weißer Knoten erreicht werden kann.

Beweis

„⇒“ Annahme: v Nachfolger von u im DFS Wald

- Sei w beliebiger Knoten auf Weg im DFS Wald von u nach v (ungleich u)
- Damit ist w Nachfolger von u
- Nach Korollar 23.2: d[u] < d[w]. Somit ist w weiß zum Zeitpunkt d[u]

Graphenalgorithmen



Satz 23.3 (Satz vom weißen Weg)

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

Beweis

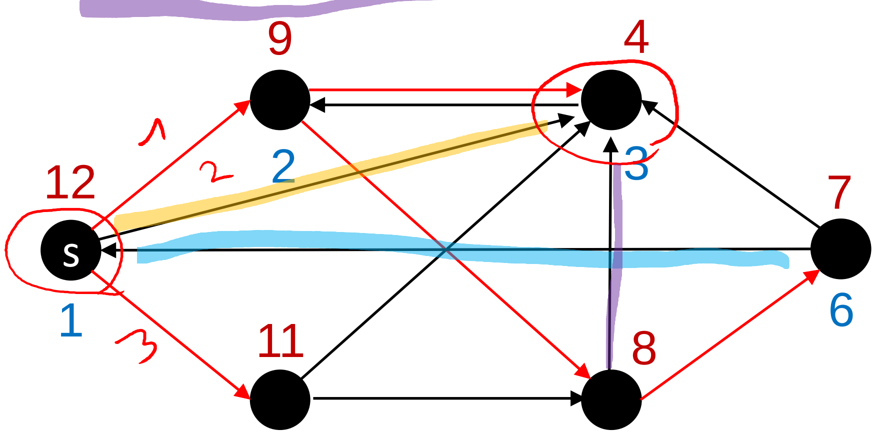
„$\Leftarrow$“ Annahme: v ist erreichbar von u über Weg aus weißen Knoten zum Zeitpunkt d[u], aber v wird nicht Nachfolger von u im DFS Wald

- ObdA. sei v der einzige Knoten auf Weg, der nicht Nachfolger wird
- Sei w der Vorgänger von v auf dem Weg und sei w Nachfolger von u
- Korollar 23.2: $f[w] \le f[u]$
- v muss entdeckt werden, nachdem u entdeckt wurde und bevor w abgearbeitete ist, d.h. $d[u] < d[v] < f[w] \le f[u]$
- Somit ist [d[v],f[v]] in [d[u],f[u]] enthalten (Satz 23.1) und nach Korollar 23.2 ist v Nachfolger von u

Graphenalgorithmen

Klassifikation von Kanten

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



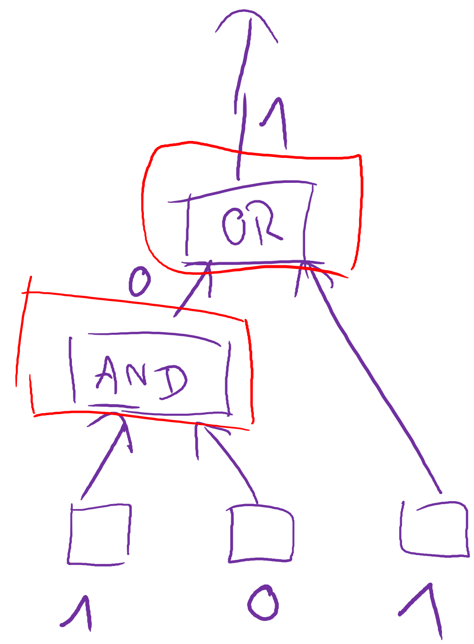
Graphalgorithmen

Zusammenfassung Tiefensuche

- Algorithmus
- Klammerstruktur
- Satz vom weißen Weg

Anwendungen Tiefensuche

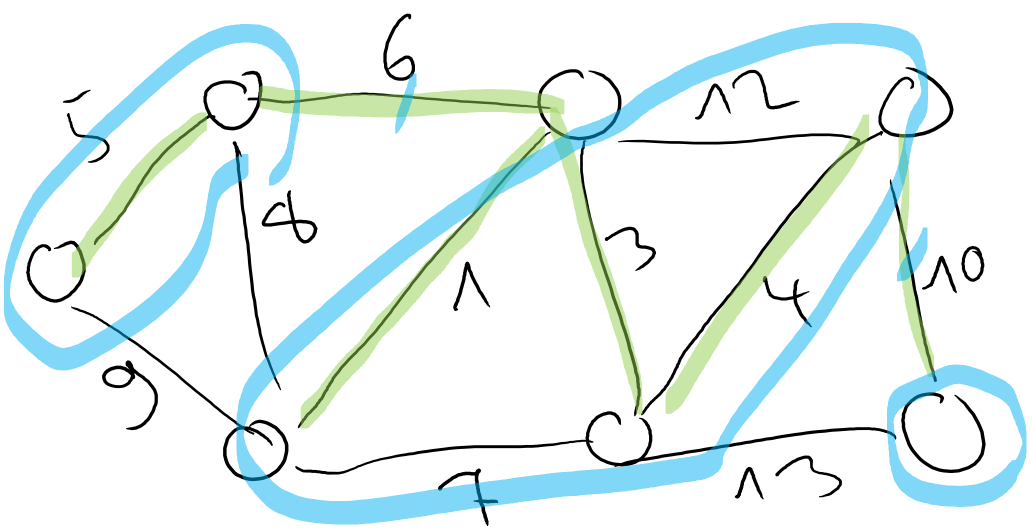
- Finden starker Zusammenhangskomponenten
- Topologisches Sortieren



Graphenalgorithmen

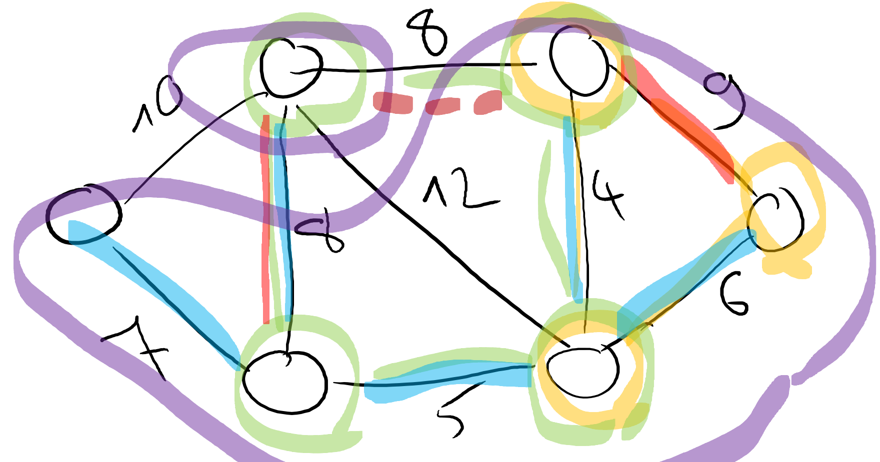
Minimale Spannbäume

- Gegeben: Gewichteter, ungerichteter, zusammenhängender Graph G=(V,E)
- Gesucht: Ein aufspannender Baum mit minimalem Gewicht
  (Summe der Kantengewichte des Baums)
- Aufspannender Baum: Baum mit Knotenmenge V



29

Graphenalgorithmen



Satz 23.4

- Sei G=(V,E) ein ungerichteter, gewichteter, zusammenhängender Graph. Sei $(v_0,..,v_k)$ ein einfacher Kreis in G und sei e eine Kante des Kreises mit maximalem Gewicht. Dann ist der Graph G=(V,E\{e}) zusammenhängend und der minimale Spannbaum hat dasselbe Gewicht wie der minimale Spannbaum von G.

Beweis

- Sei T ein minimaler Spannbaum von G und sei e wie im Satz
- Der Graph G=(V, E\{e}) ist zusammenhängend, da nur eine Kante aus einem Kreis entfernt wurde
- Ist e nicht in T, so folgt der Satz sofort. Sei also e eine Kante von T.
- Entfernen von e aus T lässt T in zwei Zusammenhangskomponenten zerfallen

Graphenalgorithmen

Satz 23.4

- Sei G=(V,E) ein ungerichteter, gewichteter, zusammenhängender Graph. Sei $(v_0,..,v_k)$ ein einfacher Kreis in G und sei e eine Kante des Kreises mit maximalem Gewicht. Dann ist der Graph G=(V,E\{e}) zusammenhängend und der minimale Spannbaum hat dasselbe Gewicht wie der minimale Spannbaum von G.

Beweis

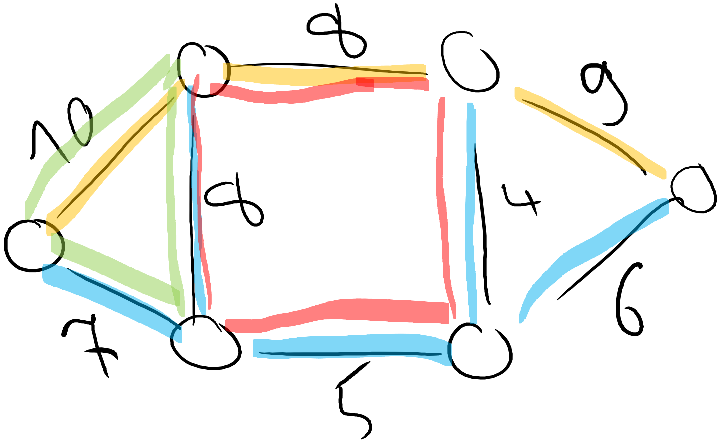
- Das e auf dem Kreis $(v_0,..,v_k)$ lag, gibt es noch eine andere Kante des Kreises f, die die Zusammenhangskomponenten verbindet
- Nach der Wahl von e ist das Gewicht von f höchstens das Gewicht von e
- Somit ist der Baum $T\setminus\{e\}\cup\{f\}$ ein minimaler Spannbaum

Graphenalgorithmen

MST-1(G)

1. T=G
2. **while** T ist kein Baum **do**
3. Finde Kreis in T
4. Entferne die Kante mit maximalem Gewicht aus dem Kreis
5. **return** T

Graphenalgorithmen



Kruskal(G)

1. A =∅
2. Sortiere Kanten nach Gewicht
3. **for each** (u,v)∈E geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in derselben Zusammenhangskomponente in Graph H=(V,A) **then**
5. A = A∪{(u,v)}
6. **return** A

$O(|E| \cdot \log |E|)$

Graphenalgorithmen

Zusammenfassung

- Tiefensuche
  - Klammerstruktur der Tiefensuche
  - Satz vom weißen Weg
- Minimale Spannbäume
  - Definition
  - Die Kreiseigenschaft
  - Algorithmus von Kruskal

Referenzen

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