Informatik II - SS 2016

(Algorithmen & Datenstrukturen)

Vorlesung 15 (15.6.2016)

Graphenalgorithmen III (MST)

Fabian Kuhn

Algorithmen und Komplexität

UNI FREIBURG

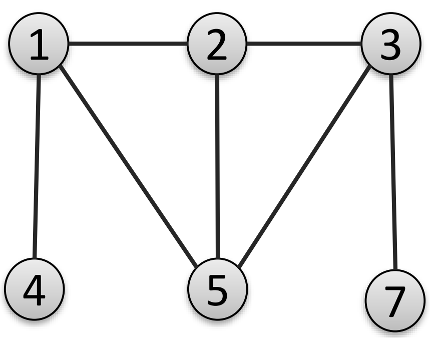
DFS- “Klammer”-Theorem

Wir definieren für jeden Knoten v die folgenden zwei Zeitpunkte

- $t_{v,1}$: Zeitpunkt, wenn v in der DFS-Suche grau gefärbt wird
- $t_{v,2}$: Zeitpunkt, wenn v in der DFS-Suche schwarz gefärbt wird

Theorem: Im DFS-Baum ist ein Knoten v genau dann im Teilbaum eines Knoten u, falls das Intervall $\left[t_{v,1}, t_{v,2}\right]$ vollständig im Intervall $\left[t_{u,1}, t_{u,2}\right]$ enthalten ist. Zudem sind zwei Intervalle entweder disjunkt, oder das eine ist komplett im anderen enthalten.

Beispiel:



Klassifizierung der Kanten (bei DFS-Suche)

Baumkanten:

- (u, v) ist eine Baumkante, falls v von u aus entdeckt wird

Rückwärtskanten:

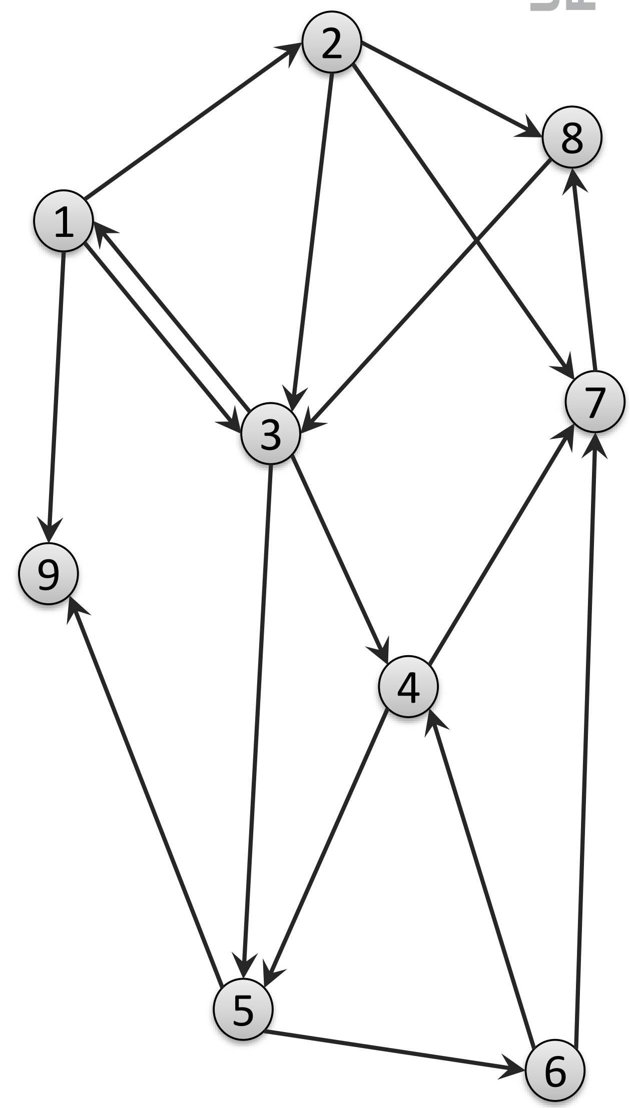
- (u, v) ist eine Rückwärtskante, falls v ein Vorgängerknoten von u ist

Vorwärtskanten:

- (u, v) ist eine Vorwärtskante, falls v ein Nachfolgerknoten von u ist

Querkanten:

- Alle übrigen Kanten



DFS – Eigenschaften

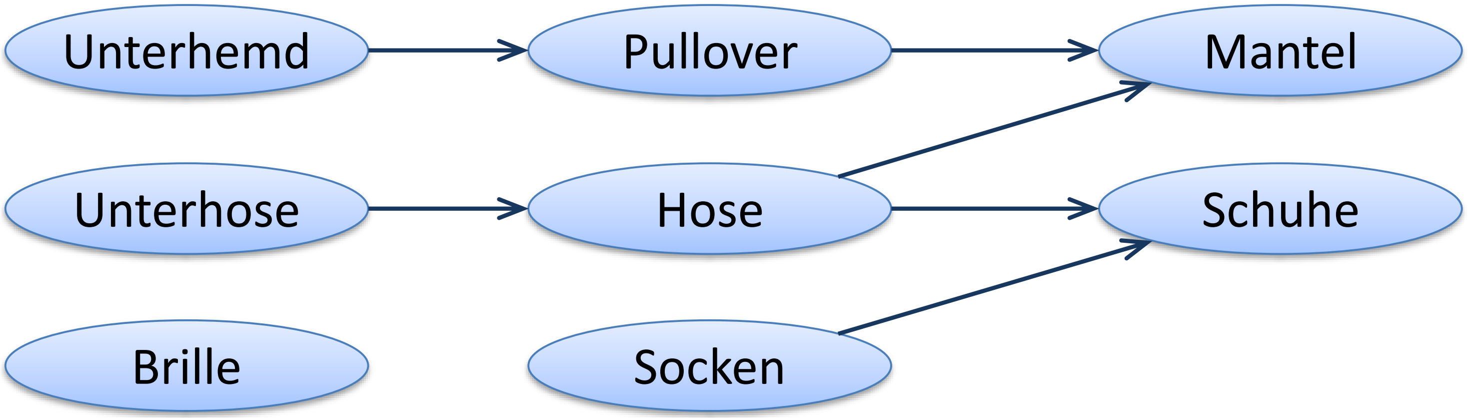
Theorem: Bei einer DFS-Suche in ungerichteten Graphen ist jede Kante entweder eine Baumkante oder eine Rückwärtskante.

Theorem: Ein gerichteter Graph hat genau dann keine Zyklen, falls es bei der DFS-Suche keine Rückwärtskanten gibt.

Anwendung: Topologische Sortierung

Zyklenfreie, gerichtete Graphen:

- **DAG**: directed acyclic graph
- Modellieren z.B. zeitliche Abhängigkeiten von Aufgaben
- Beispiel: Anziehen von Kleidungsstücken



Topologische Sortierung:

- Sortiere die Knoten eines DAGs so, dass u vor v erscheint, falls ein gerichteter Pfad von u nach v existiert
- Im Beispiel: Finde eine mögliche Anziehreihenfolge

Topologische Sortierung: Etwas formaler…

Zyklenfreie, gerichtete Graphen:

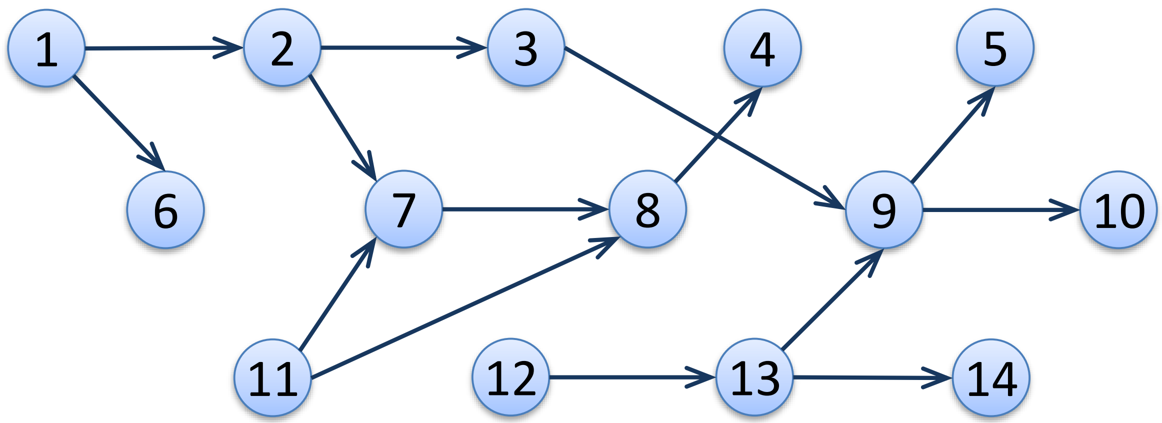
- repräsentieren partielle Ordnungsrelationen
  - asymmetrisch: $a < b \implies \neg(b < a)$
    transitiv: $a < b \,\wedge\, b < c \implies a < c$
  - partielle Ordnung: nicht alle Paare müssen vergleichbar sein
- Beispiel: Teilmengenrelation bei Mengen

Topologische Sortierung:

- Sortiere die Knoten eines DAGs so, dass u vor v erscheint, falls ein gerichteter Pfad von u nach v existiert
- Erweitere eine partielle Ordnung zu einer totalen Ordnung

Topologische Sortierung: Algorithmus

Führe DFS aus...



$t_{1,1}, t_{2,1}, t_{3,1}, t_{9,1}, t_{10,1}, \boldsymbol{t_{10,2}}, t_{5,1}, \boldsymbol{t_{5,2}}, \boldsymbol{t_{9,2}}, \boldsymbol{t_{3,2}}, t_{7,1}, t_{8,1}, t_{4,1}, \boldsymbol{t_{4,2}}, \boldsymbol{t_{8,2}},$
$\boldsymbol{t_{7,2}}, \boldsymbol{t_{2,2}}, t_{6,1}, \boldsymbol{t_{6,2}}, \boldsymbol{t_{1,2}}, t_{11,1}, \boldsymbol{t_{11,2}}, t_{12,1}, t_{13,1}, t_{14,1}, \boldsymbol{t_{14,2}}, \boldsymbol{t_{13,2}}, \boldsymbol{t_{12,2}}$

Beobachtung:

- Knoten ohne Nachfolger werden als erstes besucht (schwarz gef.)
- Besuchreihenfolge ist umgekehrte topologische Sortierung

Topologische Sortierung: Algorithmus

Theorem: Umgekehrte “Visit”-Reihenfolge (schwarz färben) der Knoten bei DFS-Traversierung ergibt topologische Sortierung

DFS-Traversierung: Weitere Anwendungen

Stark zusammenhängende Komponenten

- Stark zus.-hängende Komponente eines gerichteten Graphen: "Maximale Knoten-Teilmenge, so dass jeder jeden erreichen kann"

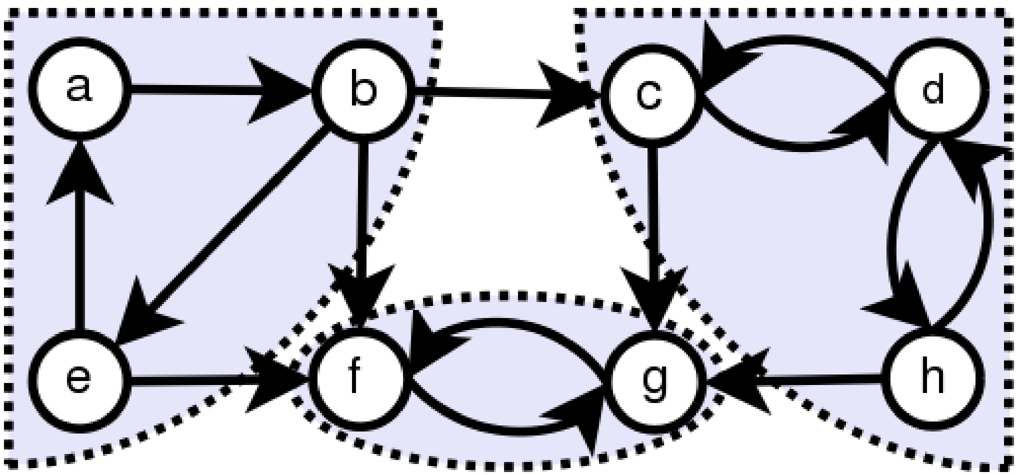


Bild: Wikipedia

- Kann man mit 2 DFS-Traversierungen finden (Zeit $\in O(m+n)$)
  - auf G und auf G^T (alle Kanten umgedreht)
  - G und G^T haben die gleichen stark zus.-hängenden Komponenten
- Details z.B. in [CLRS]

DFS-Traversierung: Weitere Anwendungen

Artikulationsknoten, Brücken, Biconnected Components

- Annahme: ungerichteter Graph
- **Artikulationsknoten v:**
  v entfernen vergrössert die Anzahl Komponenten

- **Brücke e:**
  Kante e entfernen vergrössert die Anzahl Komponenten

DFS-Traversierung: Weitere Anwendungen

Biconnected Components

- Komponenten, welche übrig bleiben, wenn man alle Brücken entfernt

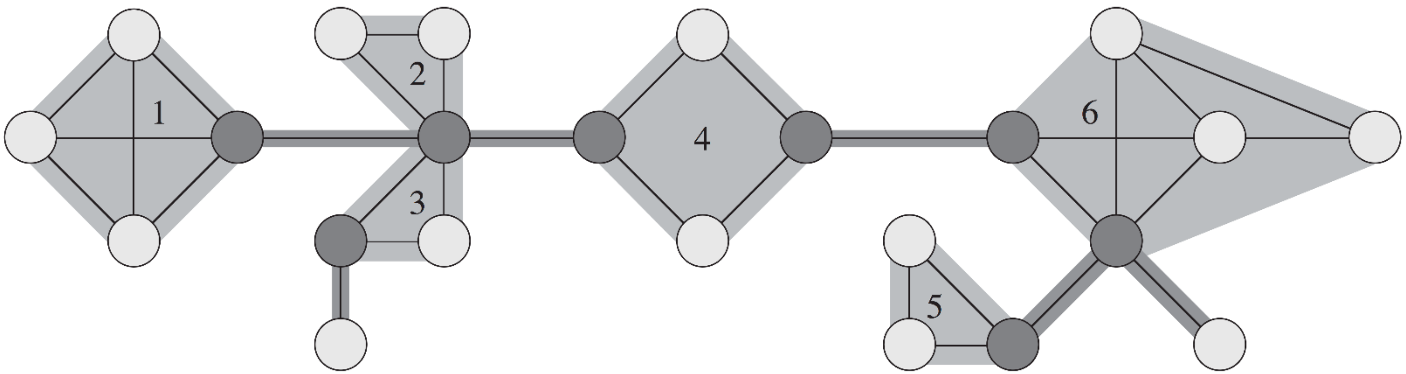


Bild: [CLRS]

- Artikulationsknoten und Brücken können mit einer DFS-Traversierung in $O(m+n)$ Zeit gefunden werden
  - Algorithmus von Hopcroft, Tarjan (1973)
- Zerlegung in Biconnected Components daher in der gleichen Zeit

Bäume

- Als ungerichtete Graphen (mit n Knoten) betrachtet…

Baum:

- Zusammenhängender ungerichteter Graph, ohne Zyklen
  - Ein nicht zus.-hängender zyklenfreier (unger.) Graph heisst Wald
  - Anzahl Kanten: $n - 1$ (jede Kante reduziert die #Komponenten um 1)

Äquivalente Definitionen:

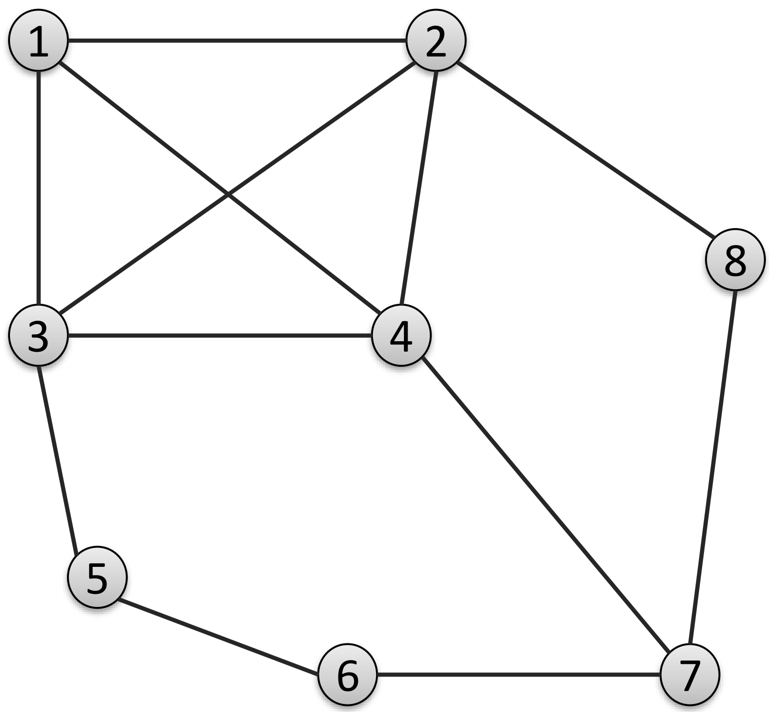
- Minimaler zusammenhängender Graph
- Maximaler zyklenfreier Graph
- Eindeutiger Pfad zwischen jedem Knotenpaar
- Zusammenhängender Graph mit $n - 1$ Kanten

Spannbaum

Gegeben: Zusammenhängender, ungerichteter Graph $G = (V, E)$

Spannbaum $\boldsymbol{T = (V, E_T)}$: Teilgraph ($E_T \subseteq E$)

- T ist ein Baum ist, welcher alle Knoten von G enthält
- Alternativ: T ist ein Baum mit $n - 1$ Kanten aus E

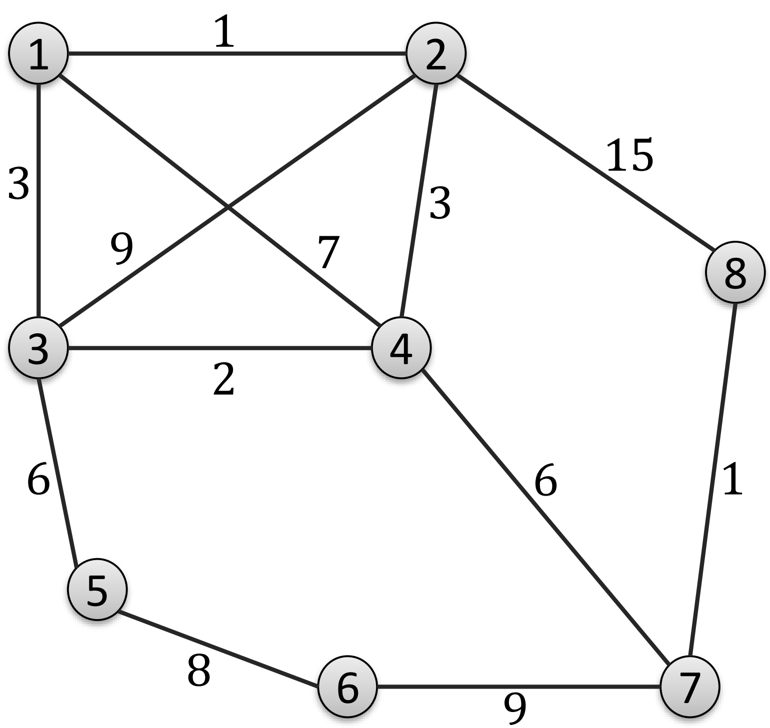


Minimaler Spannbaum

Gegeben: Zus.-hängender, ungerichteter Graph $G = (V, E, w)$ mit Kantengewichten $w : E \to \mathbb{R}$

Minimaler Spannbaum $\boldsymbol{T = (V, E_T)}$:

- Spannbaum mit kleinstem Gesamtgewicht



Minimale Spannbäume

Ziel: Gegeben ein gewichteter, ungerichteter Graph G, finde einen Spannbaum mit minimalem Gesamtgewicht.

- **Minimaler Spannbaum = Minimum Spanning Tree = MST**
- Ein grundlegendes Optimierungsproblem auf Graphen
  - eines von sehr vielen Optimierungsproblemen auf Graphen
- kommt oft als Teilproblem vor
- ist aber auch interessant an sich

Basis-MST-Algorithmus

Idee: Starte mit leerer Kantenmenge und füge die Kanten schrittweise hinzu, bis es ein Spannbaum ist

Invariante:
Algorithmus hat zu jeder Zeit eine Kantenmenge A, so dass A Teilmenge eines minimalen Spannbaums ist.

- Am Anfang ist $A = \emptyset$
- Danach wird jeweils eine Kante hinzugefügt, ohne die Invariante zu verletzen
- Wir nennen eine Kante, für welche wir sicher sein können, dass wir sie zu A hinzufügen können eine sichere Kante für A
- Wie man sichere Kanten findet, werden wir sehen…

Basis-MST-Algorithmus

Invariante:
Algorithmus hat zu jeder Zeit eine Kantenmenge A, so dass A Teilmenge eines minimalen Spannbaums ist.

Basis-MST-Algorithmus:

A = $\emptyset$
while A ist kein Spannbaum **do**
    Finde sichere Kante $\{u, v\}$ für A
    A = $A \cup \big\{\{u, v\}\big\}$
return A

- Invariante ist eine gültige Schleifeninvariante
- Invariante + Abbruchbedingung $\Longrightarrow$ A ist ein MST!

Wie findet man sichere Kanten?

- Invariante → es gibt immer mindestens eine sicher Kante
  - A ist Teilmenge eines MST und kann daher zu einem MST erweitert werden

- Zuerst benötigen wir ein paar Begriffe...

Schnitt $(S, V \setminus S)$:

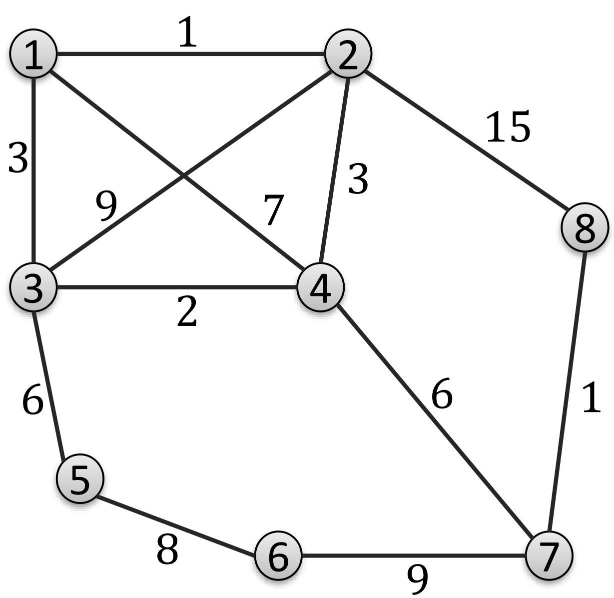
- Kante $\{u, v\} \in E$ ist eine Schnittkante bezüglich $(S, V \setminus S)$, falls ein Ende in S und ein Ende in $V \setminus S$ ist.
- Wir nennen Kante $\{u, v\}$ eine **leichte Schnittkante** bez. $(S, V \setminus S)$, falls sie das kleinste Gewicht von allen Schnittkanten hat

Sichere Kanten

Annahmen:

- $G = (V, E, w)$ ist zus.-h., unger. Graph mit Kantengewichten $w(e)$
- A ist Teilmenge (Teilgraph) eines MST

Theorem: Sei $(S, V \setminus S)$ ein Schnitt, so dass A keine Schnittkanten enthält und sei $\{u, v\}, u \in S, v \in V \setminus S$ eine leichte Schnittkante bezüglich $(S, V \setminus S)$. Dann ist $\{u, v\}$ eine sichere Kante für A.



Sichere Kanten

Theorem: Sei $(S, V \setminus S)$ ein Schnitt, so dass A keine Schnittkanten enthält und sei $\{u, v\}$, $u \in S$, $v \in V \setminus S$ eine leichte Schnittkante bezüglich $(S, V \setminus S)$. Dann ist $\{u, v\}$ eine sichere Kante für A.

Sichere Kanten

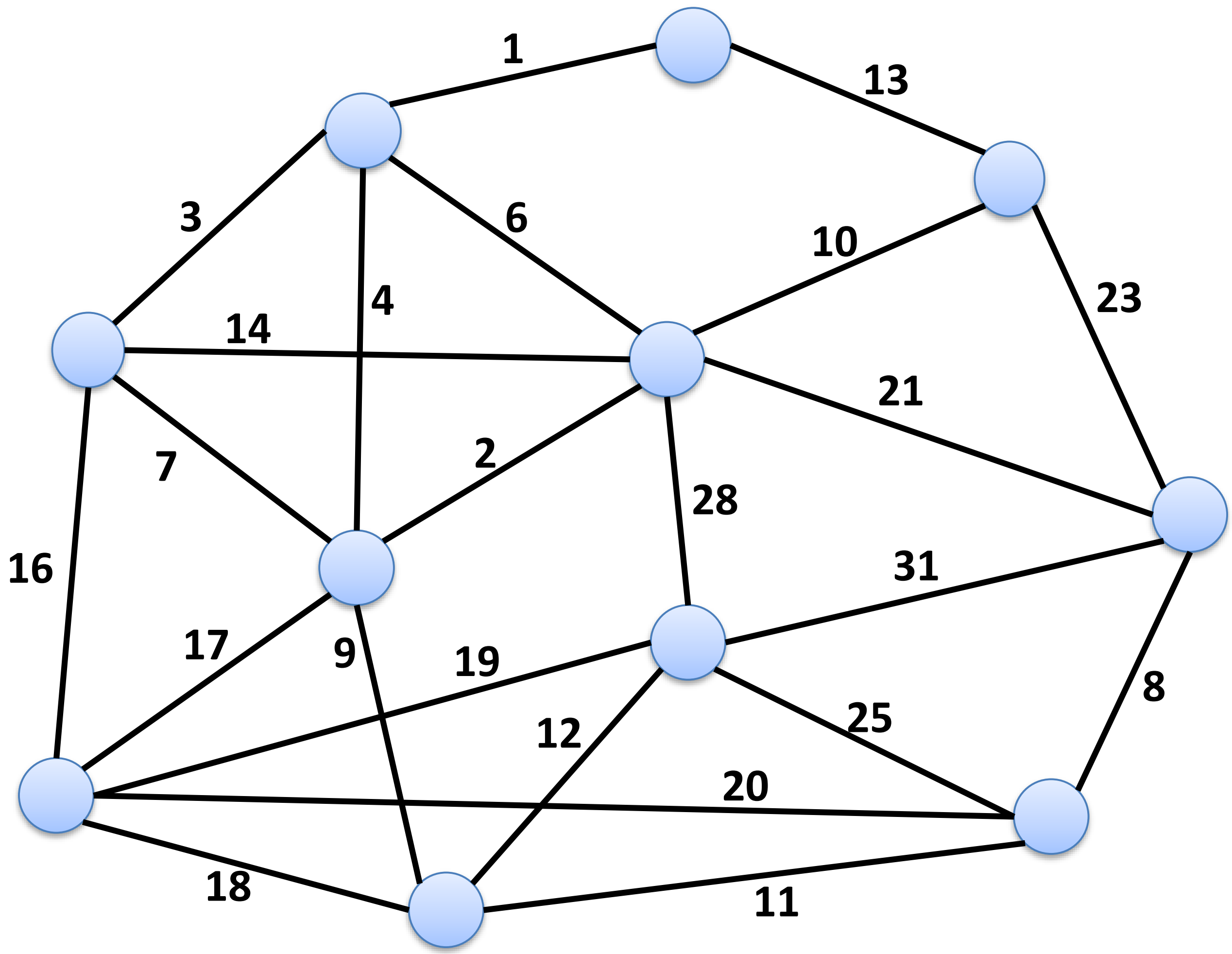
Theorem: Sei $(S, V \setminus S)$ ein Schnitt, so dass A keine Schnittkanten enthält und sei $\{u, v\}$, $u \in S$, $v \in V \setminus S$ eine leichte Schnittkante bezüglich $(S, V \setminus S)$. Dann ist $\{u, v\}$ eine sichere Kante für A.

Kruskal's MST-Algorithmus

```
A = ∅
while A ist kein Spannbaum do
    e = {u,v} ist Kante mit kleinstem Gewicht,
    so dass A ∪ {{u,v}} keinen Zyklus enthält
    A = A ∪ {{u,v}}
```

- Wir müssen zeigen, dass e eine sichere Kante für A ist

Kruskal's MST-Algorithmus: Beispiel



Kruskal's MST-Algorithmus: Bemerkungen

Kruskal's Algorithmus berechnet minimalen Spannbaum

- Die hinzugefügte Kante ist in jedem Schritt sicher
- Wir haben gesehen, dass der Basis-Algorithmus korrekt ist

Kruskal's Algorithmus ist ein typisches Beispiel eines sogenannten

Greedy Algorithmus

- Wir beginnen mit einer leeren Kantenmenge
- In jedem Schritt wird die im Moment beste Kante hinzugenommen
- Eine gewählte Kante wird nie wieder verworfen

- Wir werden noch kurz besprechen, wie man den Algorithmus effizient implementieren kann...

Implementierung Kruskals Algorithmus

Kruskals Algorithmus (Pseudo-Code)

$A = \emptyset$

Sortiere Kanten aufsteigend nach Kantengewicht

for $e = \{u, v\} \in E$ (in sorted order) **do**

    if u and v are in different components **then**

        $A = A \cup \{e\}$

Implementierung Kruskals Algorithmus

Kruskals Algorithmus (Pseudo-Code)

```
1.  A = ∅
2.  Sortiere Kanten aufsteigend nach Kantengewicht
3.  for e = {u,v} ∈ E (in sorted order) do
4.      if u and v are in different components then
5.          A = A ∪ {e}
```

- Müssen Komponenten des durch A bestimmten Graphen effizient verwalten können
- **Laufzeit:** $O(m \log n)$ für's Sortieren, sowie die Gesamtzeit, um die Komponenten zu verwalten…

Union-Find / Disjoint Sets Datentyp

Verwaltet eine Partition von Elementen

Operationen:

- *create* : erzeugt eine leere Union-Find-DS
- *U.makeSet(x)* : fügt Menge $\{x\}$ zur Partition hinzu
- *U.find(x)* : gibt Menge mit Element x zurück
- *U.union(S1, S2)* : vereinigt die Mengen $S1$ und $S2$

- Details dazu werden in der Algorithmentheorie besprochen…

Kruskal mit Union-Find

Kruskals Algorithmus

$A = \emptyset$
U = create new
for all $u \in V$ **do**
    U.makeSet(u)
Sortiere Kanten aufsteigend nach Kantengewicht
for all $e = \{u, v\} \in E$ (in sorted order) **do**
    S_u = U.find(u); S_v = U.find(v)
    if $S_u \neq S_v$ **then**
        $A = A \cup \{e\}$
        U.union(S_u, S_v)

Kruskal: Laufzeitanalyse

Beste Union-Find Datenstruktur

- Laufzeit für m Union-Find-Operationen auf n Elementen (n makeSet-Operationen):

$$\boldsymbol{O\big(m \cdot \alpha(m,n)\big)}$$

- $\alpha(m,n)$ ist die Inverse der Ackermannfunktion und wächst extrem langsam (für alle halbwegs vernünftigen m, n, $\alpha(m,n) \leq 5$)

Laufzeit Kruskal

- Kanten sortieren: $O(m \cdot \log n)$
- Union-Find-Operationen: $O\big(m \cdot \alpha(m,n)\big)$
- Insgesamt: $O(m \cdot \log n)$
  - besser, falls Kantengewichte schneller sortiert werden können

MST Eindeutigkeit

- Im Allgemeinen ist der MST nicht eindeutig

Satz: Bei paarweise verschiedenen Kantengewichten ist der MST eindeutig.