Informatik II - SS 2018

(Algorithmen & Datenstrukturen)

Vorlesung 15a (13.6.2018)

Graphenalgorithmen IV

UNI FREIBURG

Fabian Kuhn

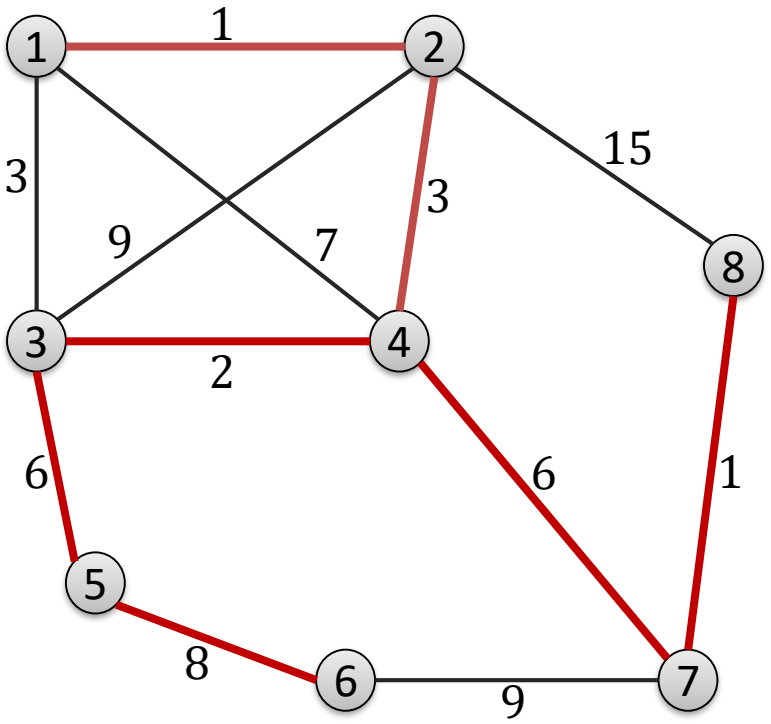
Algorithmen und Komplexität

Minimaler Spannbaum

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

Minimaler Spannbaum $T = (V, E_T)$:

- Spannbaum mit kleinstem Gesamtgewicht



MST Eindeutigkeit

UNI FREIBURG

Satz: Bei paarweise verschiedenen Kantengewichten ist der MST eindeutig.

Sichere Kanten

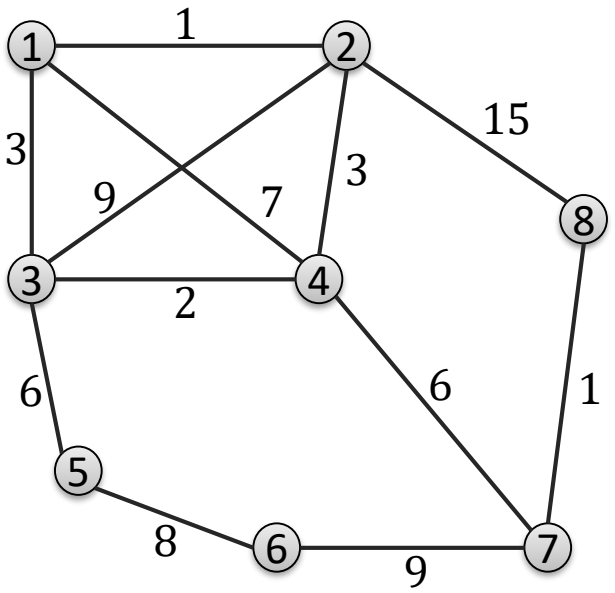
UNI FREIBURG

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

$A = \emptyset$
while A ist kein Spannbaum **do**
    $e = \{u, v\}$ ist Kante mit kleinstem Gewicht, so dass $A \cup \{e\}$ keinen Zyklus enthält
    $A = A \cup \{e\}$

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



Kruskal mit Union-Find

UNI FREIBURG

Kruskals Algorithmus

1. $A = \emptyset$
2. U = create new **Union Find** data structure
3. **for all** $u \in V$ **do**
4. U.makeSet(u)
5. Sort edges by weight (ascending)
6. **for all** $e = \{u,v\} \in E$ (in sorted order) **do**
7. S_u = U.find(u); S_v = U.find(v)
8. **if** $S_u \neq S_v$ **then**
9. $A = A \cup \{e\}$
10. U.union(S_u,S_v)

Kruskal: Laufzeitanalyse

UNI FREIBURG

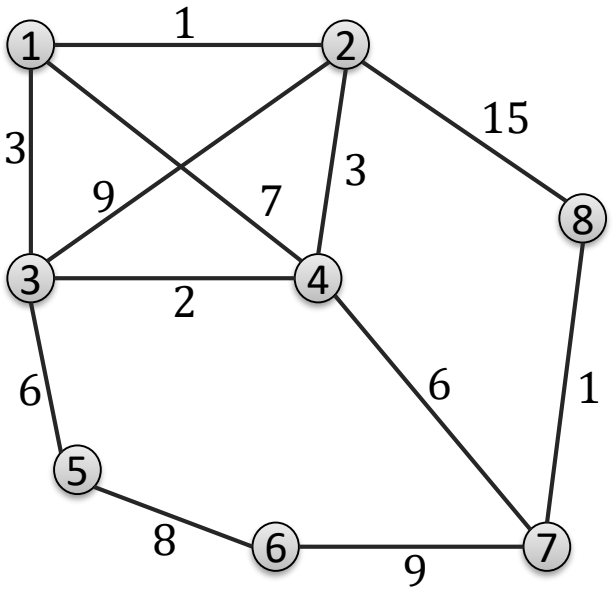
Laufzeit Kruskal

- 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

Prims MST-Algorithmus

$A = \{s\}$
while A ist kein Spannbaum **do**
    $e = \{u, v\}$ ist Kante mit kleinstem Gewicht,
    so dass $u \in A$ und $v \notin A$
    $A = A \cup \{e\}$

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



Implementierung von Prims Algorithmus

UNI FREIBURG

```
H = create(); A = ∅
for all u ∈ V \ {s} do
    H.insert(u, ∞); α(u) = NULL
H.insert(s, 0)

while H is not empty do
    u = H.deleteMin()
    for all unmarked neighbors v of u do
        if w({u,v}) < d(v) then
            H.decreaseKey(v, w({u,v}))
            α(v) = u
    if u ≠ s then A = A ∪ {{u, α(u)}}
```

Laufzeit: $O\big(n \cdot \big(T(\texttt{insert}) + T(\texttt{deleteMin})\big) + m \cdot T(\texttt{decreaseKey})\big)$