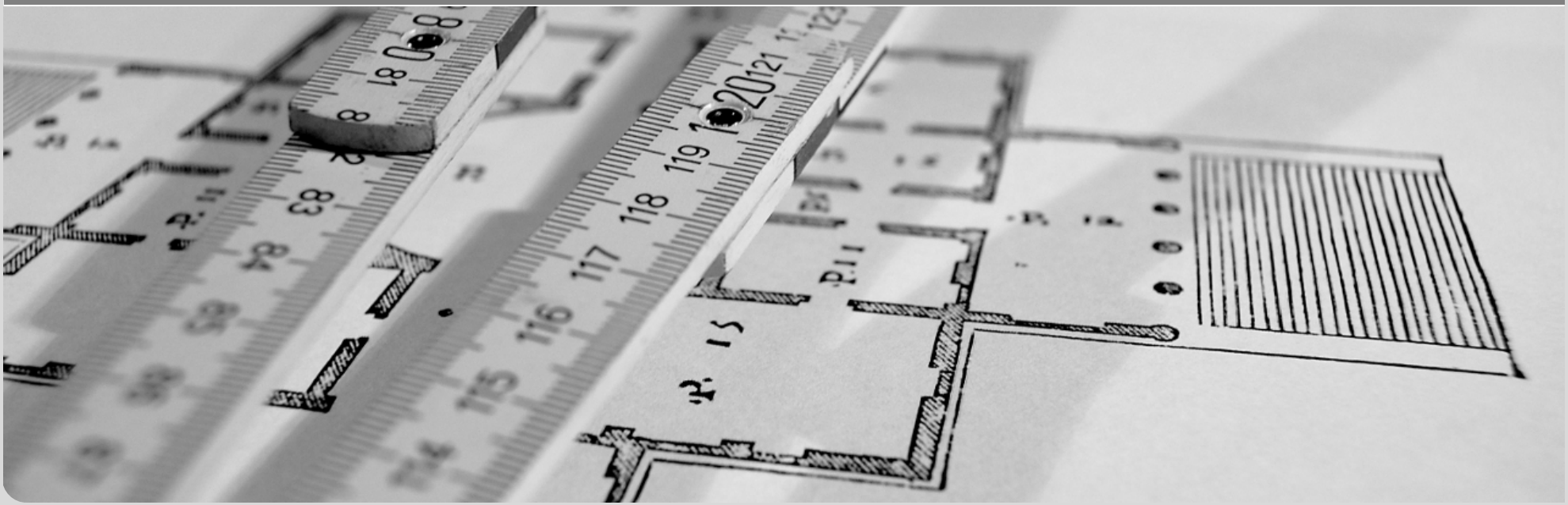
Karlsruher Institut für Technologie

Graphen II

Morris Kurz, Moritz Grauer, Michael Schrempp, Frederic Tausch | 1. Juni 2016

ICPC-PRAKTIKUM - KIT



Gliederung

Wiederholung: Graphen

- Graph G = (V,E) mit Knotenmenge V und Kantenmenge E
- wichtig heute: Gewichtete, gerichtete und zusammenhängende Graphen (Weigthed, directed and connected graph)
- negativer Zyklus: Es existiert ein Zyklus, wobei die Summe der Kantengewichte negativ ist
- Shortest Path: Minimale Distanz bzgl. der Gewichtung zwischen zwei Knoten
- Beispiel: Navigationssystem

Wiederholung: Graphen

- Graph G = (V,E) mit Knotenmenge V und Kantenmenge E
- wichtig heute: Gewichtete, gerichtete und zusammenhängende Graphen (Weigthed, directed and connected graph)
- negativer Zyklus: Es existiert ein Zyklus, wobei die Summe der Kantengewichte negativ ist
- Shortest Path: Minimale Distanz bzgl. der Gewichtung zwischen zwei Knoten
- Beispiel: Navigationssystem

Wiederholung: Graphen

- Graph G = (V,E) mit Knotenmenge V und Kantenmenge E
- wichtig heute: Gewichtete, gerichtete und zusammenhängende Graphen (Weigthed, directed and connected graph)
- negativer Zyklus: Es existiert ein Zyklus, wobei die Summe der Kantengewichte negativ ist
- Shortest Path: Minimale Distanz bzgl. der Gewichtung zwischen zwei Knoten
- Beispiel: Navigationssystem

Wiederholung: Graphen

- Graph G = (V,E) mit Knotenmenge V und Kantenmenge E
- wichtig heute: Gewichtete, gerichtete und zusammenhängende Graphen (Weigthed, directed and connected graph)
- negativer Zyklus: Es existiert ein Zyklus, wobei die Summe der Kantengewichte negativ ist
- Shortest Path: Minimale Distanz bzgl. der Gewichtung zwischen zwei Knoten
- Beispiel: Navigationssystem

Bellman-Ford-Algorithmus

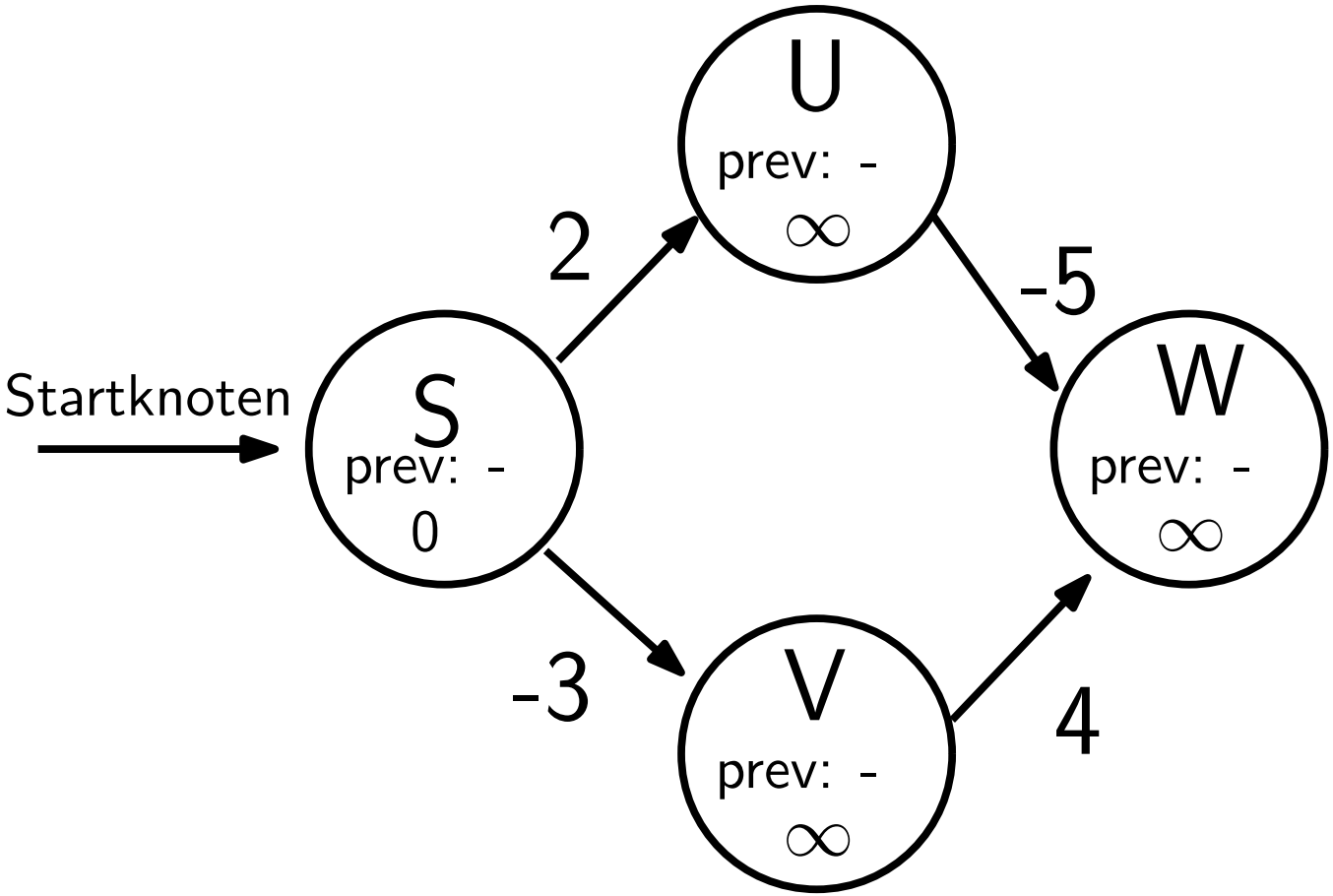
- Gegeben: Gerichteter und gewichteter Graph G=(V,E) mit $|V| = n$ und $|E| = m$
- Ziel: Berechnung des kürzesten Weges von einem Startknoten s zu allen anderen Knoten

Bellman-Ford-Algorithmus

- Gegeben: Gerichteter und gewichteter Graph G=(V,E) mit $|V|$ = n und $|E| = m$
- Ziel: Berechnung des kürzesten Weges von einem Startknoten s zu allen anderen Knoten

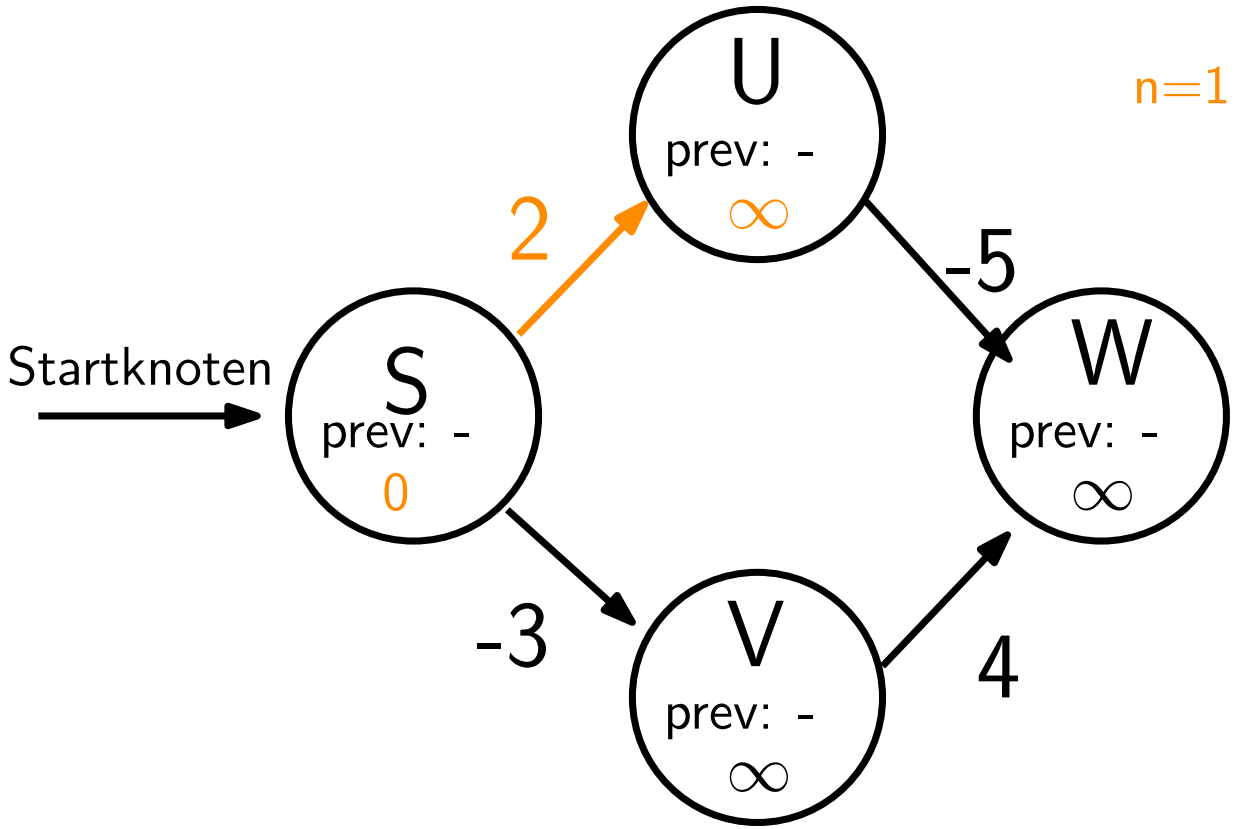
Schritt 1

- Weise Startknoten S Distanz 0 zu, den anderen Knoten Distanz ∞ und keinen Vorgänger



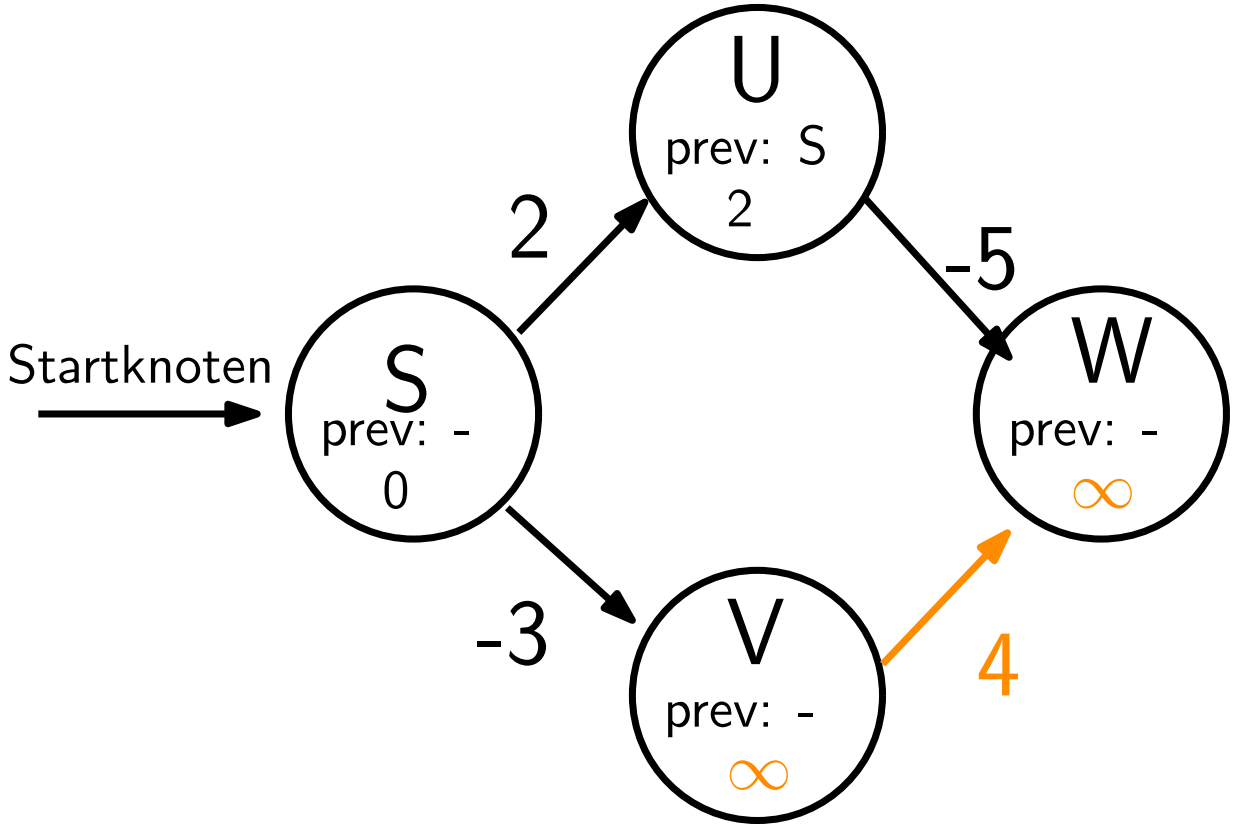
Schritt 2

- Wiederhole n-1 mal:
  - Gehe alle Kanten (u,v) aus E durch, Reihenfolge beliebig
  - wenn Distanz(u) + Gewicht(u,v) $<$ Distanz(v) dann setze Distanz(v) auf den größeren Wert und Vorgänger(v) auf u



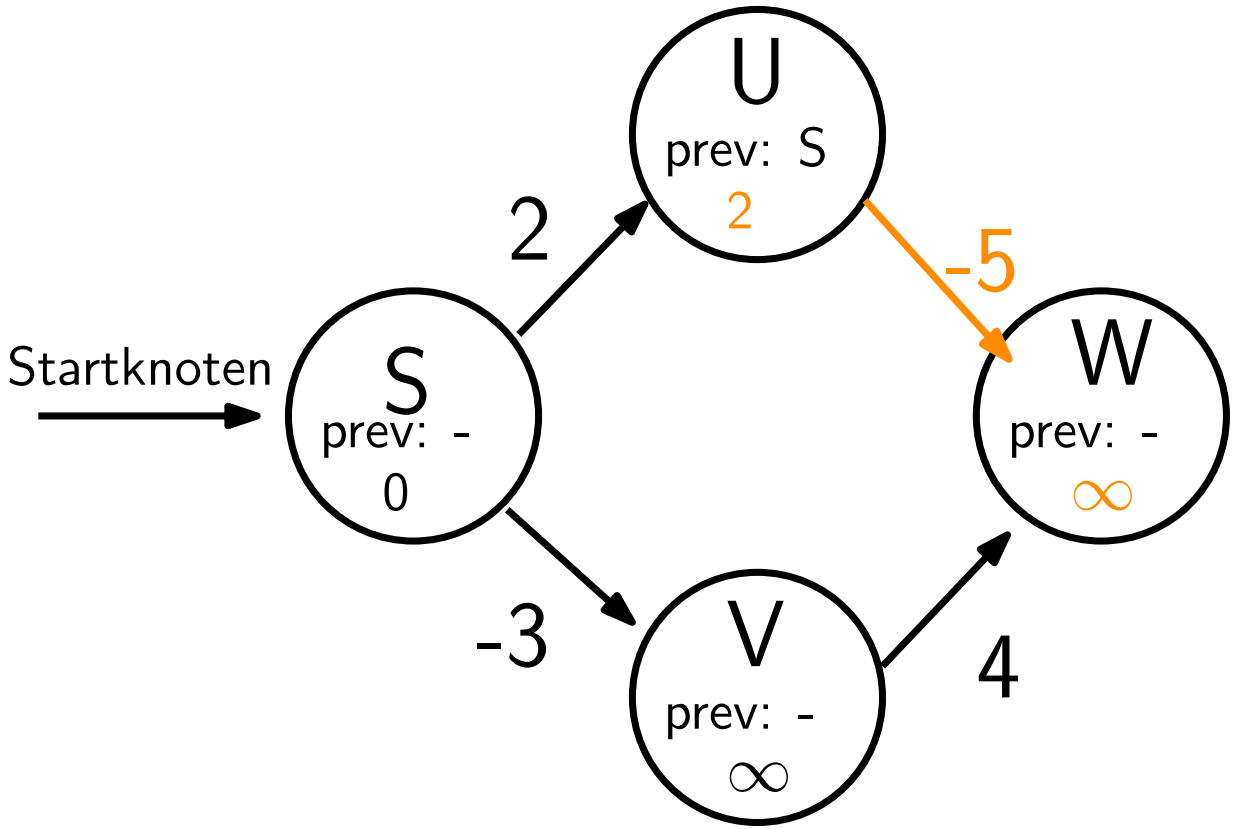
Schritt 2

- Wiederhole n-1 mal:
  - Gehe alle Kanten (u,v) aus E durch, Reihenfolge beliebig
  - wenn Distanz(u) + Gewicht(u,v) $<$ Distanz(v) dann setze Distanz(v) auf den größeren Wert und Vorgänger(v) auf u



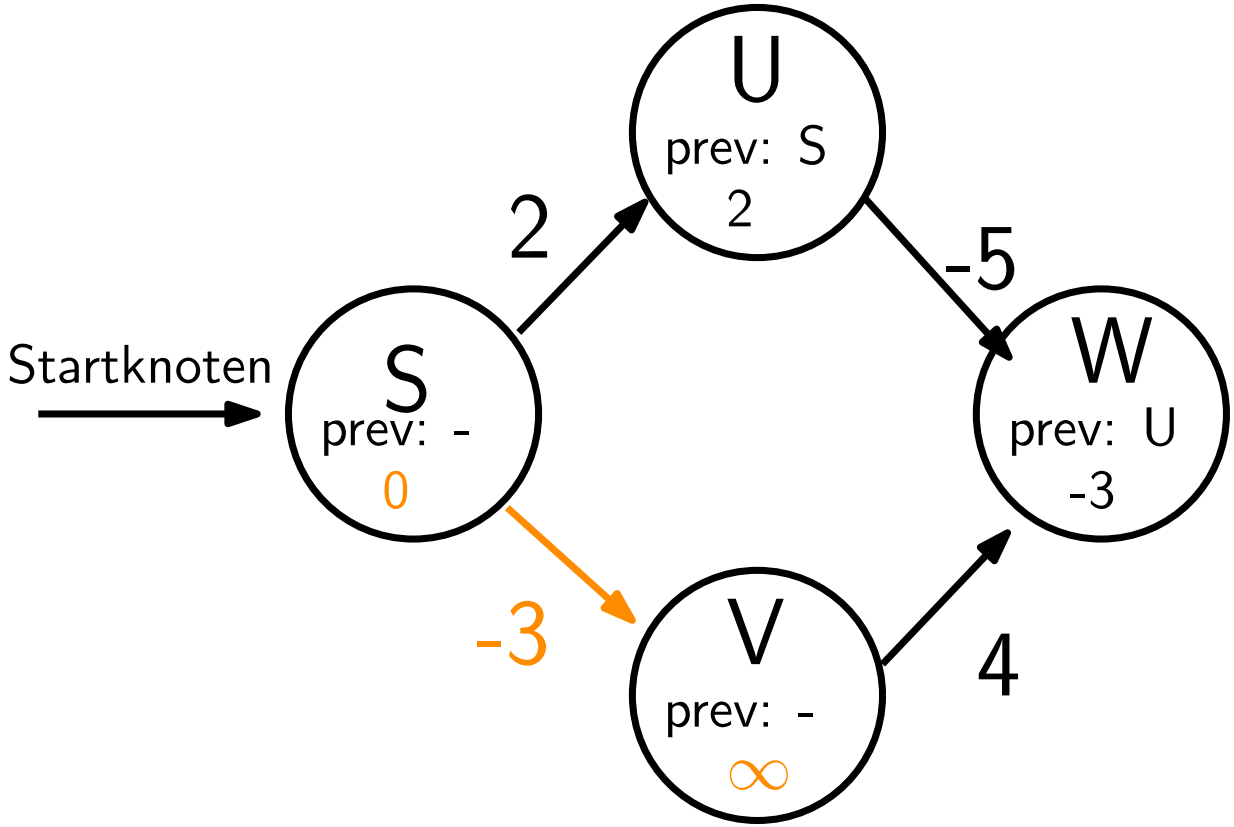
Schritt 2

- Wiederhole n-1 mal:
  - Gehe alle Kanten (u,v) aus E durch, Reihenfolge beliebig
  - wenn Distanz(u) + Gewicht(u,v) $<$ Distanz(v) dann setze Distanz(v) auf den größeren Wert und Vorgänger(v) auf u



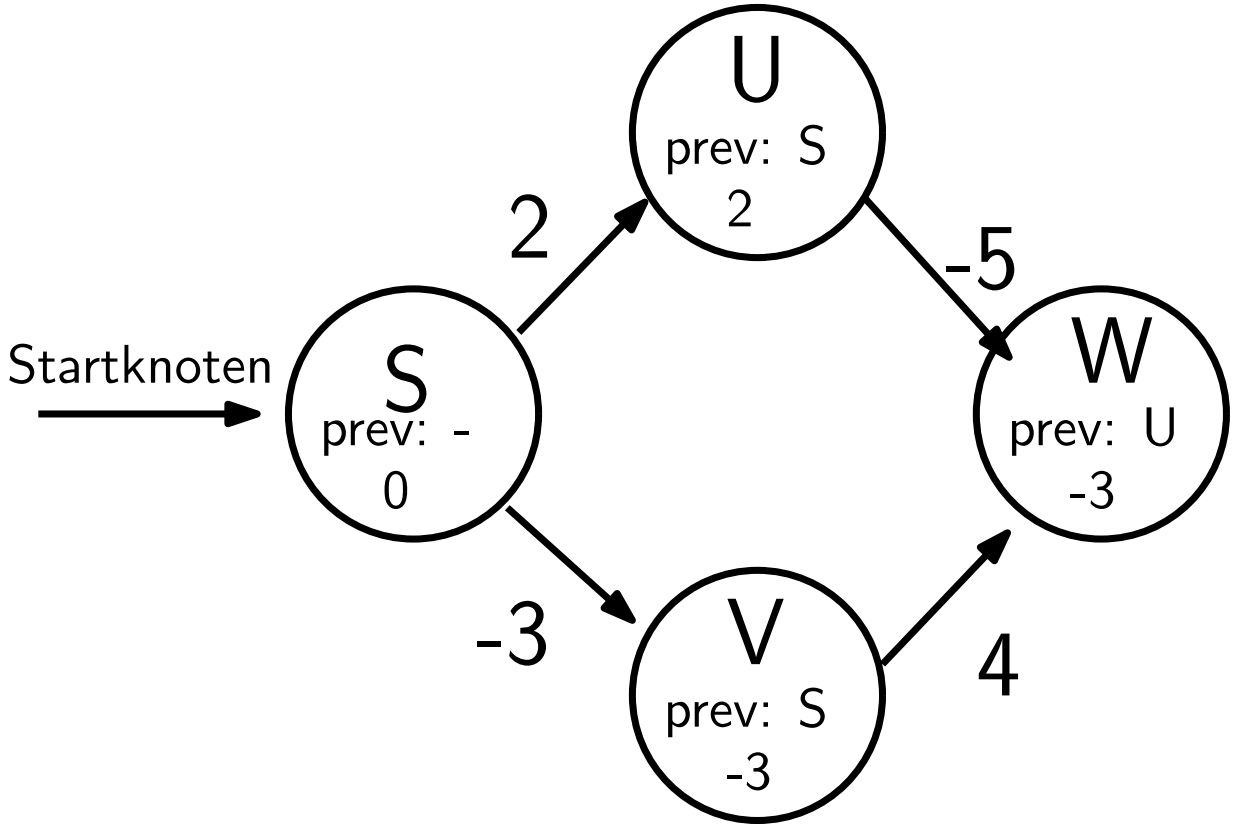
Schritt 2

- Wiederhole n-1 mal:
  - Gehe alle Kanten (u,v) aus E durch, Reihenfolge beliebig
  - wenn Distanz(u) + Gewicht(u,v) $<$ Distanz(v) dann setze Distanz(v) auf den größeren Wert und Vorgänger(v) auf u



Schritt 2

- Wiederhole n-1 mal:
  - Gehe alle Kanten (u,v) aus E durch, Reihenfolge beliebig
  - wenn Distanz(u) + Gewicht(u,v) $<$ Distanz(v) dann setze Distanz(v) auf den größeren Wert und Vorgänger(v) auf u



Bellman-Ford-Algorithmus

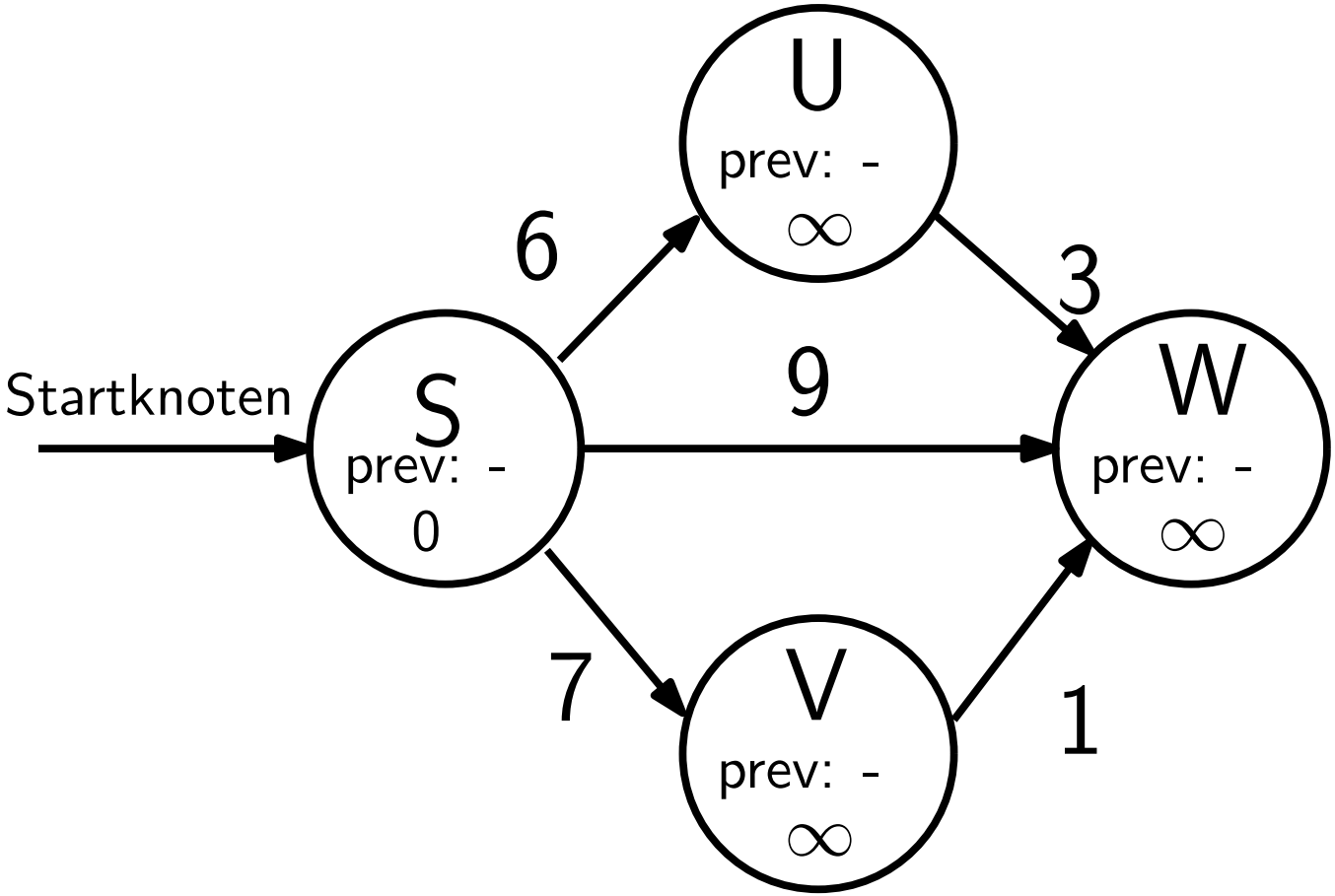
- Zum Schluss wird jede Kante noch einmal durchgegangen
  - Bei Veränderung: Negativer Zyklus vorhanden
- Laufzeit: O(n*m)

Schneller: Dijkstra-Algorithmus

- Gleiche Vorbedingungen und Zielsetzung wie bei Bellman-Ford
- Vorbemerkung: Negative Kantengewichte nicht erlaubt

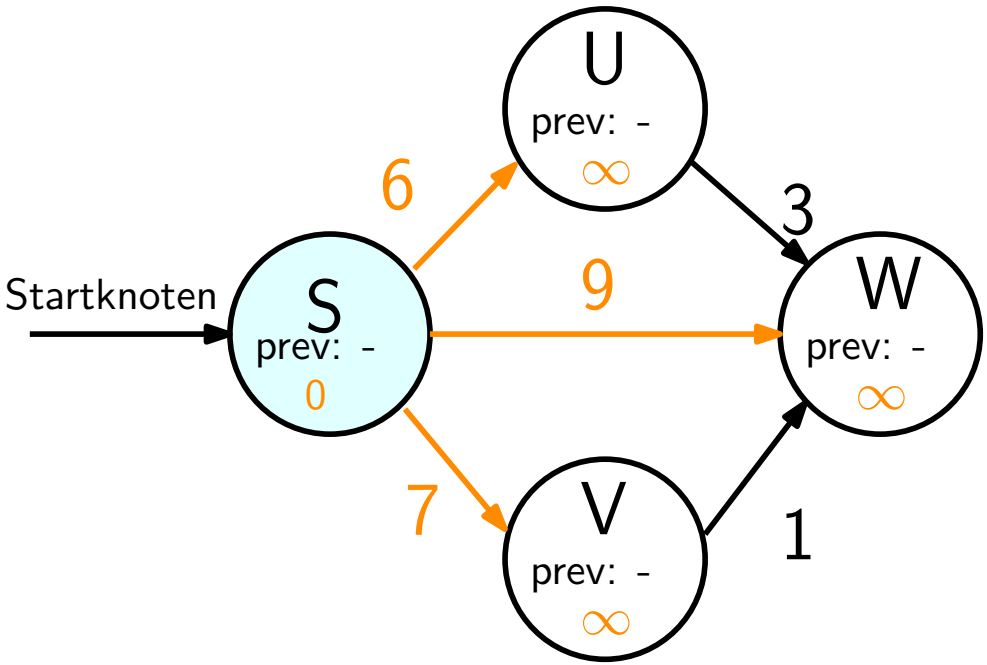
Schritt 1

- Weise Startknoten S Distanz 0 zu, den anderen Knoten Distanz ∞ und keinen Vorgänger



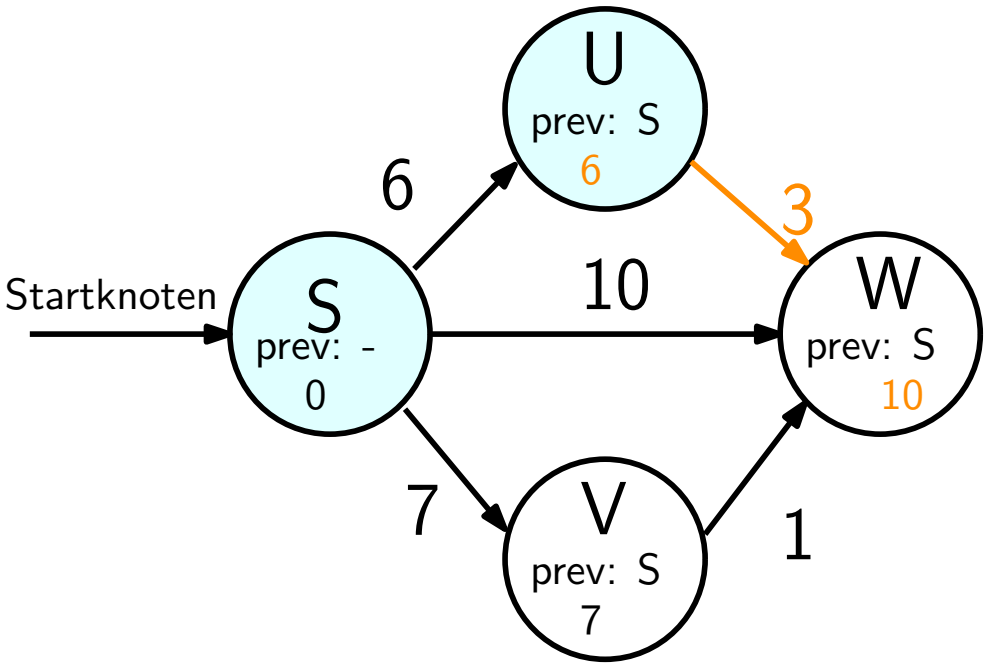
Schritt 2

- Wiederhole solange es unbesuchte Knoten gibt:
  - Betrachte den Knoten mit minimaler Distanz als nächstes
  - Setze Besucht - Flag auf true (hier blau)
  - Berechne für alle unbesuchten Nachbarsknoten u: Distanz(v) + Gewicht(u,v)
  - **Relaxieren** Falls Distanz(v) + Gewicht(u,v) ¡ Distanz(u) setze die Distanz von u auf das kleinere und Vorgänger(u) = v



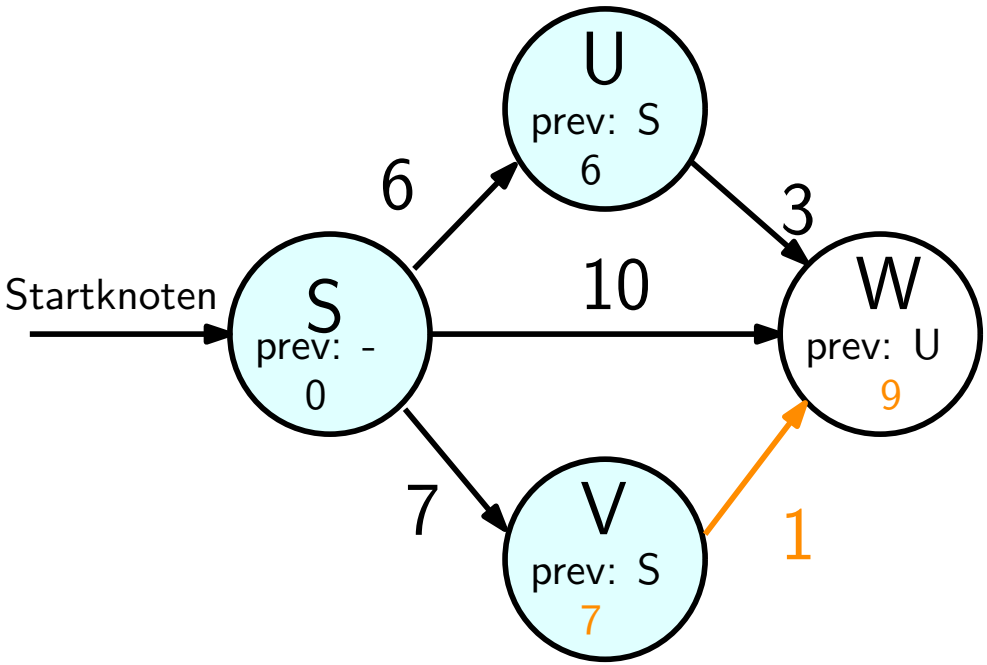
Schritt 2

- Wiederhole solange es unbesuchte Knoten gibt:
  - Betrachte den Knoten mit minimaler Distanz als nächstes
  - Setze Besucht - Flag auf true (hier blau)
  - Berechne für alle unbesuchten Nachbarsknoten u: Distanz(v) + Gewicht(u,v)
  - **Relaxieren** Falls Distanz(v) + Gewicht(u,v) ¡ Distanz(u) setze die Distanz von u auf das kleinere und Vorgänger(u) = v



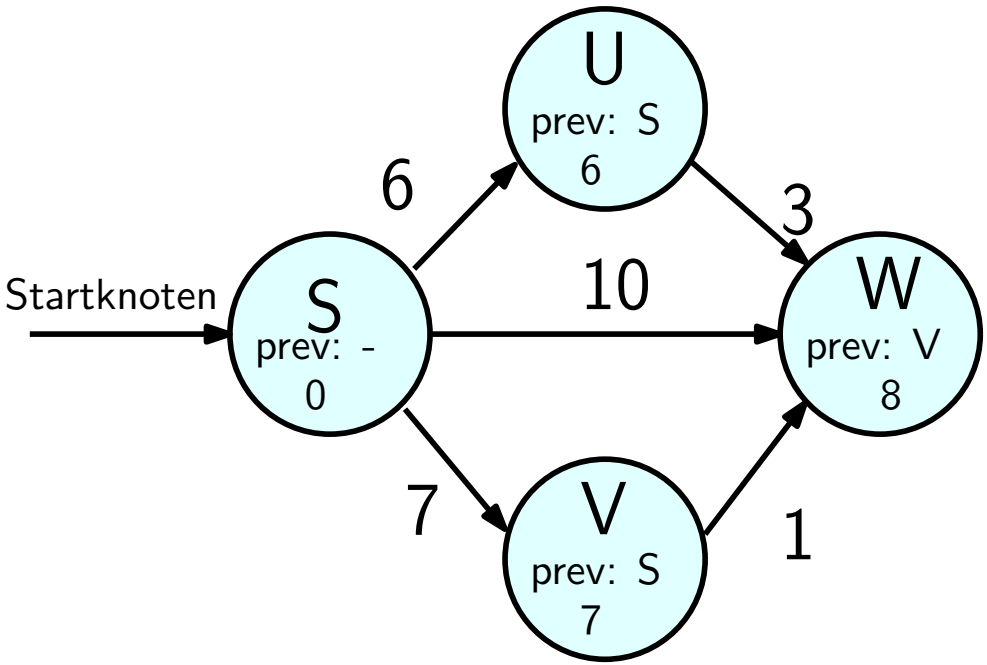
Schritt 2

- Wiederhole solange es unbesuchte Knoten gibt:
  - Betrachte den Knoten mit minimaler Distanz als nächstes
  - Setze Besucht - Flag auf true (hier blau)
  - Berechne für alle unbesuchten Nachbarsknoten u: Distanz(v) + Gewicht(u,v)
  - **Relaxieren** Falls Distanz(v) + Gewicht(u,v) ¡ Distanz(u) setze die Distanz von u auf das kleinere und Vorgänger(u) = v



Schritt 2

- Wiederhole solange es unbesuchte Knoten gibt:
  - Betrachte den Knoten mit minimaler Distanz als nächstes
  - Setze Besucht - Flag auf true (hier blau)
  - Berechne für alle unbesuchten Nachbarsknoten u: Distanz(v) + Gewicht(u,v)
  - **Relaxieren** Falls Distanz(v) + Gewicht(u,v) ¡ Distanz(u) setze die Distanz von u auf das kleinere und Vorgänger(u) = v



Schneller: Dijkstra-Algorithmus

- Problem: negative Kantengewichte sind nicht erlaubt
- Laufzeit: optimiert $O(m + n \log n)$

Schneller: Dijkstra-Algorithmus

- Problem: negative Kantengewichte sind nicht erlaubt
- Laufzeit: optimiert O(m + n log n)

Ungerichtete Bäume

- Ungerichteter Baum
  - Ungerichteter Graph
  - Zusammenhängend
  - Zyklenfrei
- Spannbaum
  - Teilgraph
  - Enthält alle Knoten
- Minimaler Spannbaum
  - Betrachte ungerichteten Baum mit Kantengewichten
  - Spannbaum mit minimalem Gewicht

Ungerichtete Bäume

- Ungerichteter Baum
  - Ungerichteter Graph
  - Zusammenhängend
  - Zyklenfrei
- Spannbaum
  - Teilgraph
  - Enthält alle Knoten
- Minimaler Spannbaum
  - Betrachte ungerichteten Baum mit Kantengewichten
  - Spannbaum mit minimalem Gewicht

Ungerichtete Bäume

- Ungerichteter Baum
  - Ungerichteter Graph
  - Zusammenhängend
  - Zyklenfrei
- Spannbaum
  - Teilgraph
  - Enthält alle Knoten
- Minimaler Spannbaum
  - Betrachte ungerichteten Baum mit Kantengewichten
  - Spannbaum mit minimalem Gewicht

Problemstellung

- Gegeben:
  - Zusammenhängender, ungerichteter, kantengewichteter Graph
- Gesucht:
  - Minimaler Spannbaum
- Lösung:
  - Algorithmus von Prim

Problemstellung

- Gegeben:
  - Zusammenhängender, ungerichteter, kantengewichteter Graph
- Gesucht:
  - Minimaler Spannbaum
- Lösung:
  - Algorithmus von Prim

Problemstellung

- Gegeben:
  - Zusammenhängender, ungerichteter, kantengewichteter Graph
- Gesucht:
  - Minimaler Spannbaum
- Lösung:
  - Algorithmus von Prim

Algorithmus von Prim

Beschreibung

- Ermittelt minimalen Spannbaum
- Idee:
- Versuche, möglichst „billige“ Kanten zu wählen und „teure“ zu umgehen
  $\rightarrow$ *Greedy*
- Algorithmus:
  - Gegeben: Ungerichteter Baum B = (V,E)
  - Wähle beliebigen Startknoten als Startgraph T
  - Solange T nicht alle Knoten aus V enthält:
    - Wähle Kante e mit minimalem Gewicht von T nach V $\setminus$ T
    - Füge e und Knoten zum Graph T hinzu

Algorithmus von Prim

Beschreibung

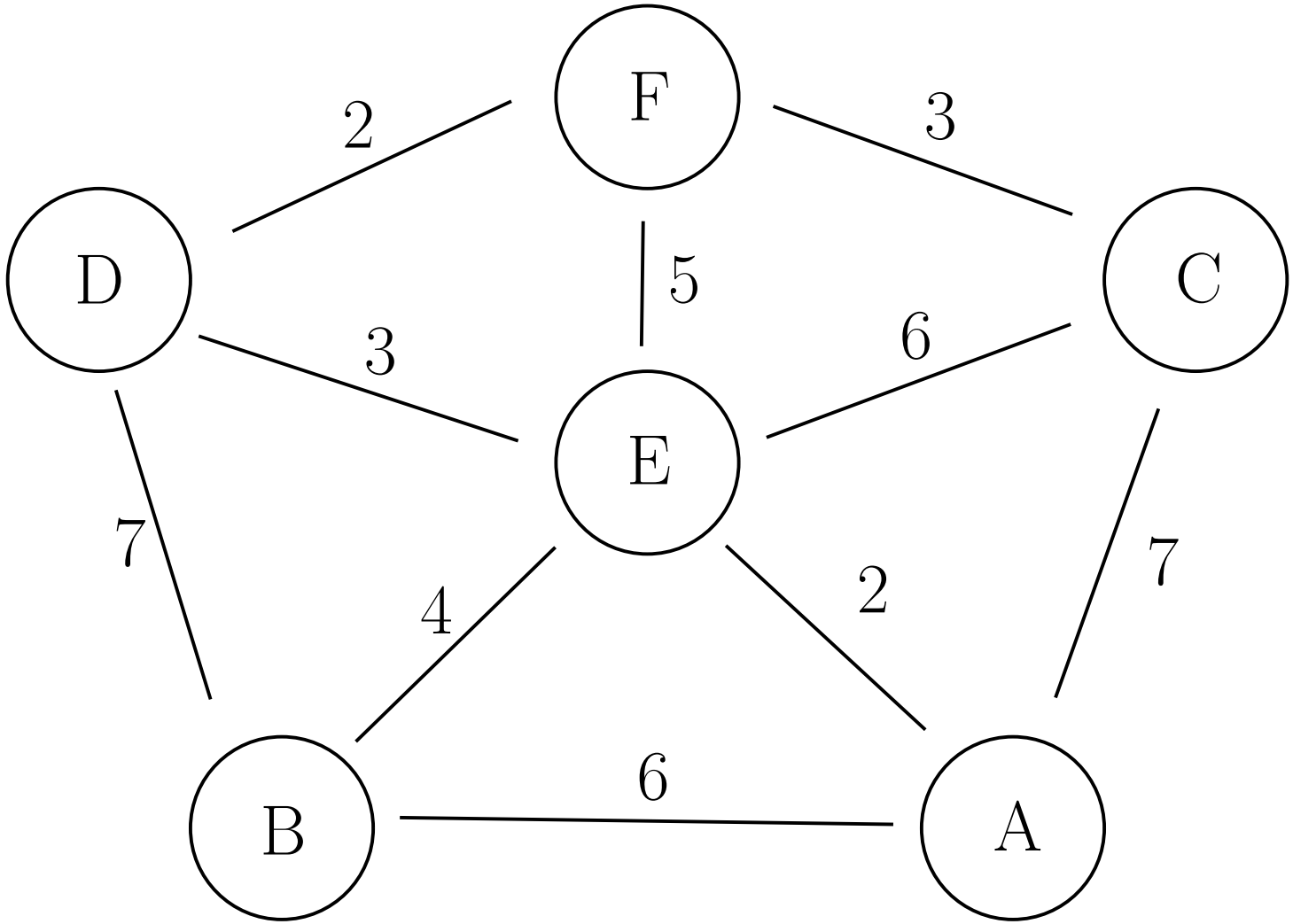
- Ermittelt minimalen Spannbaum
- Idee:
- Versuche, möglichst „billige“ Kanten zu wählen und „teure“ zu umgehen
  $\rightarrow$ *Greedy*
- Algorithmus:
  - Gegeben: Ungerichteter Baum B = (V,E)
  - Wähle beliebigen Startknoten als Startgraph T
  - Solange T nicht alle Knoten aus V enthält:
    - Wähle Kante e mit minimalem Gewicht von T nach V $\setminus$ T
    - Füge e und Knoten zum Graph T hinzu

Algorithmus von Prim

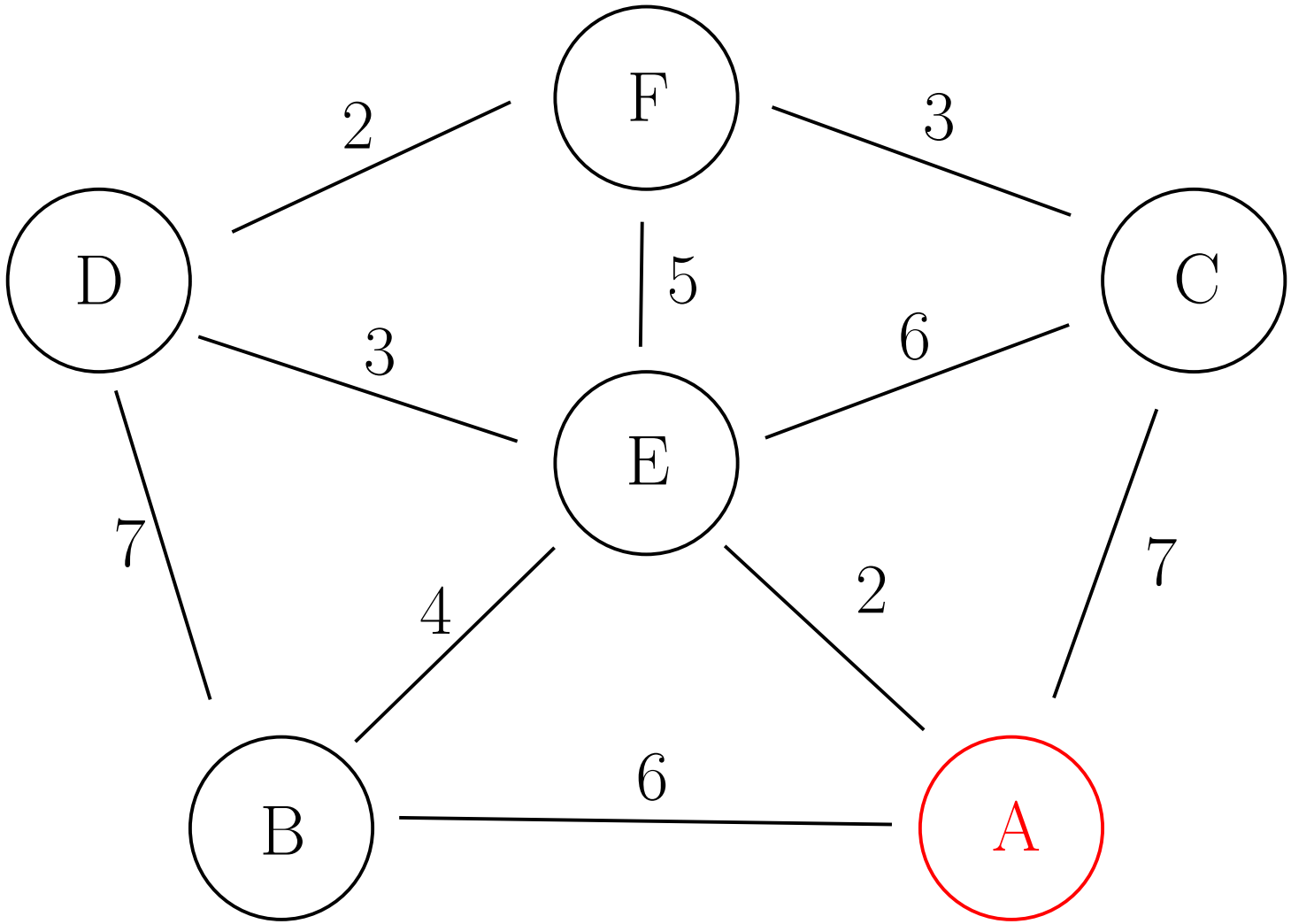
Beschreibung

- Ermittelt minimalen Spannbaum
- Idee:
- Versuche, möglichst „billige“ Kanten zu wählen und „teure“ zu umgehen
  $\rightarrow$ *Greedy*
- Algorithmus:
  - Gegeben: Ungerichteter Baum B = (V,E)
  - Wähle beliebigen Startknoten als Startgraph T
  - Solange T nicht alle Knoten aus V enthält:
    - Wähle Kante e mit minimalem Gewicht von T nach V $\setminus$ T
    - Füge e und Knoten zum Graph T hinzu

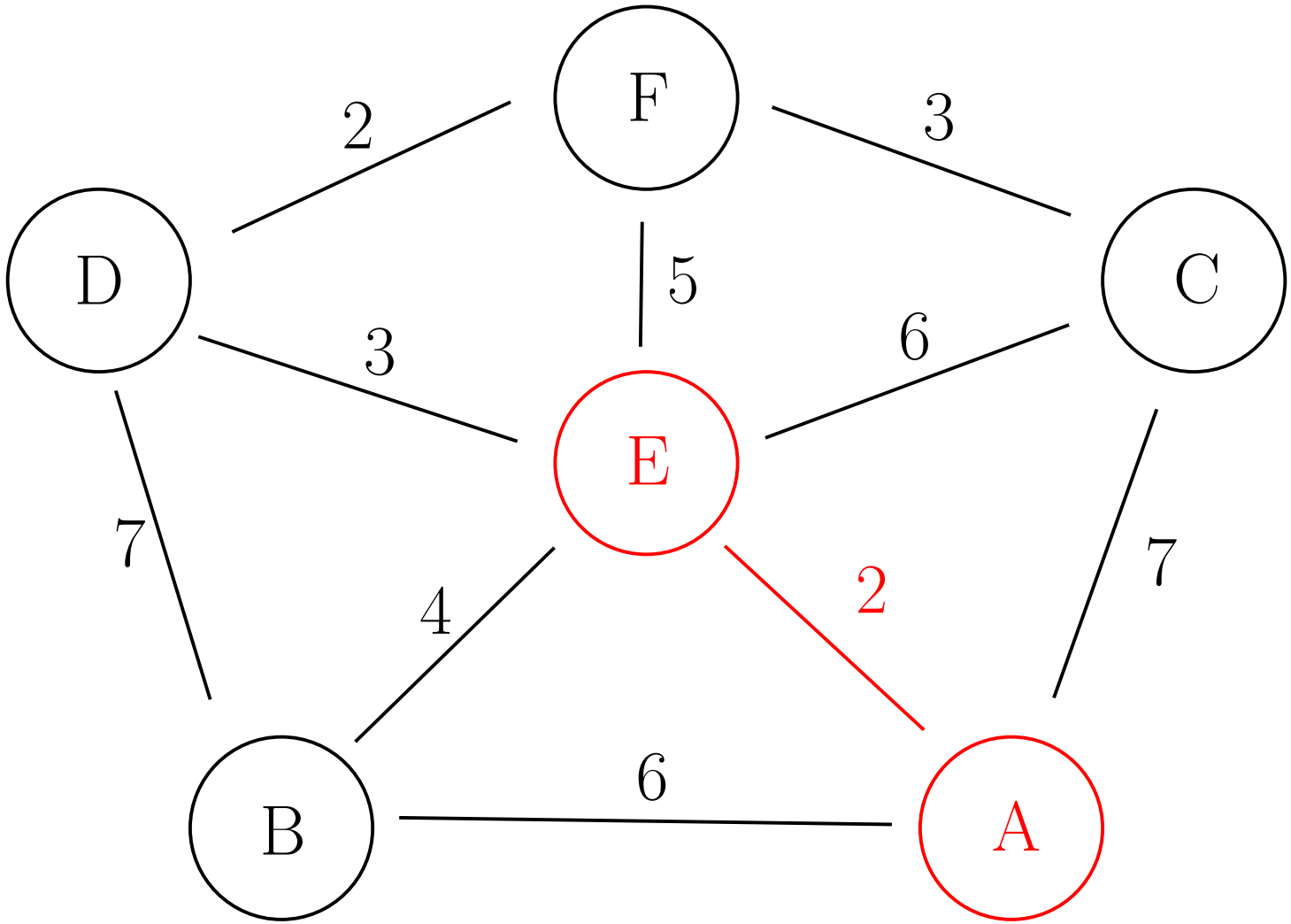
Durchführung



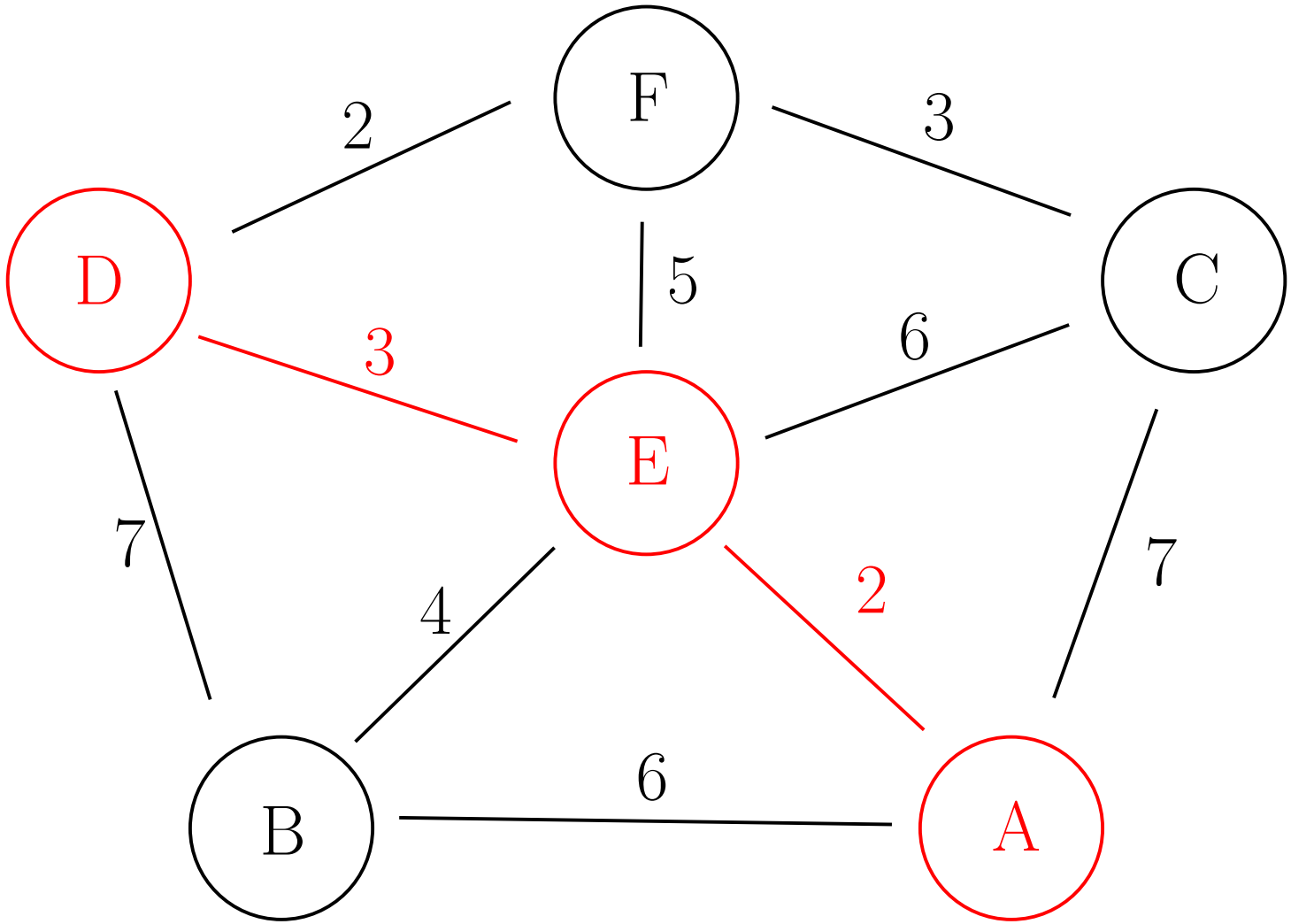
Durchführung



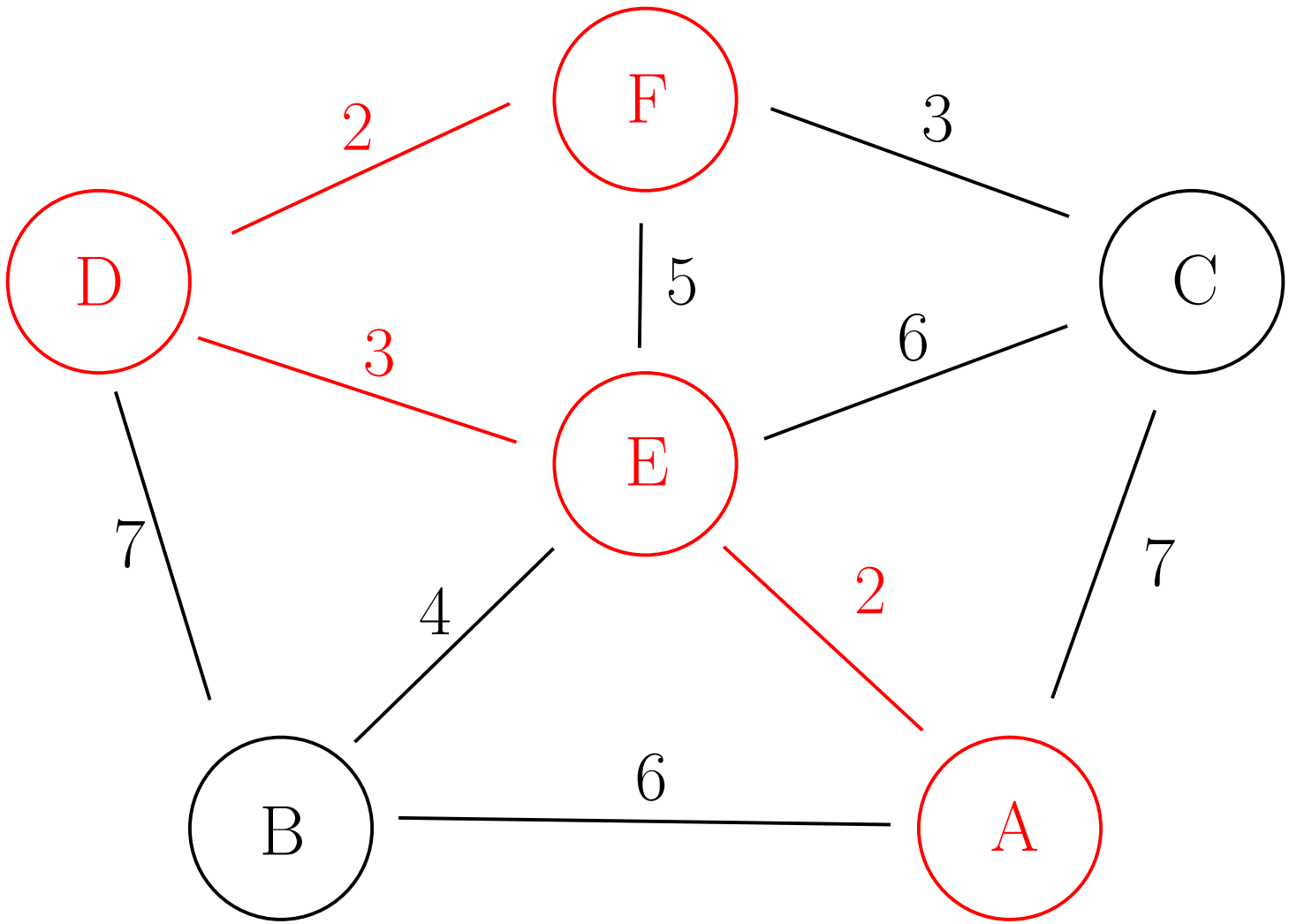
Durchführung



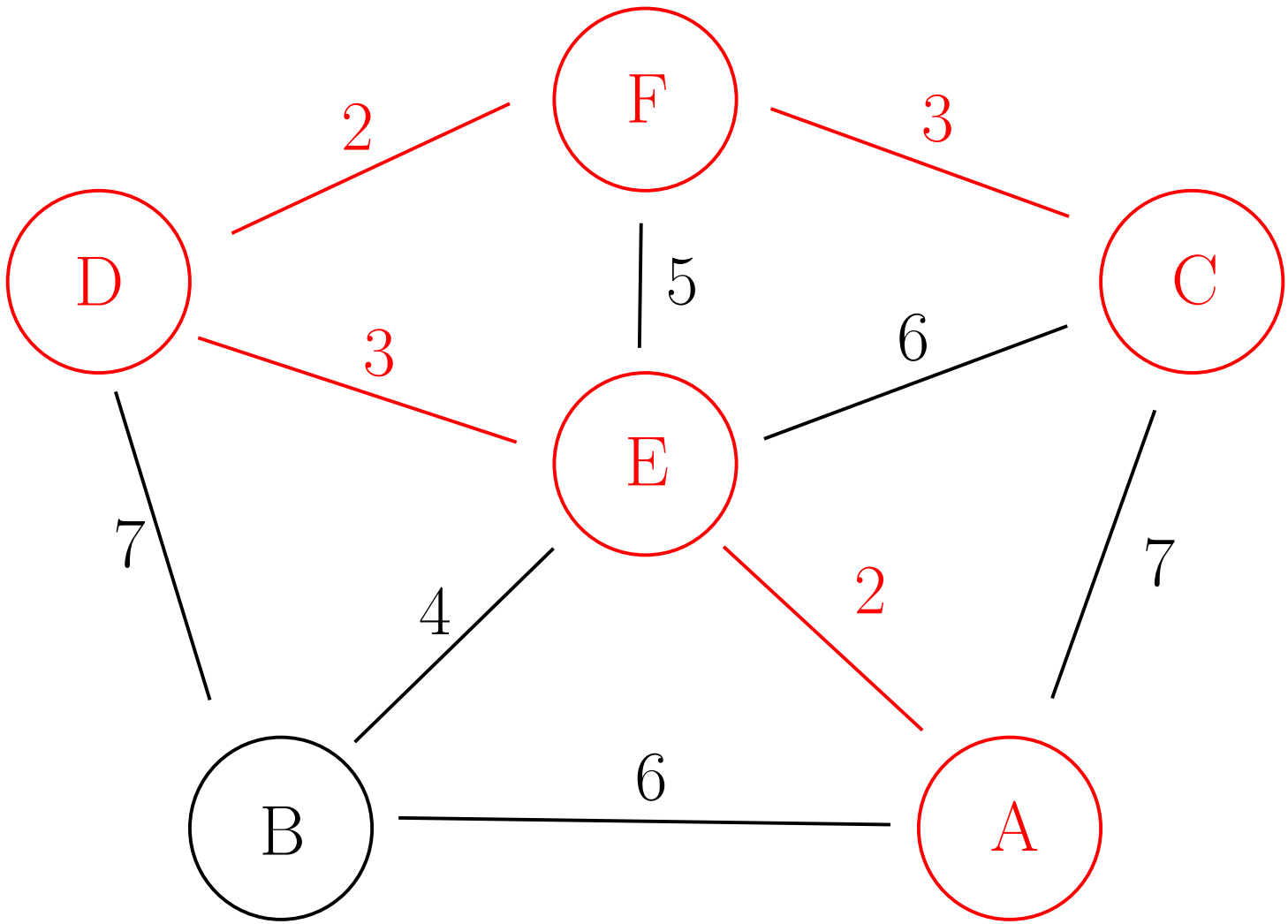
Durchführung



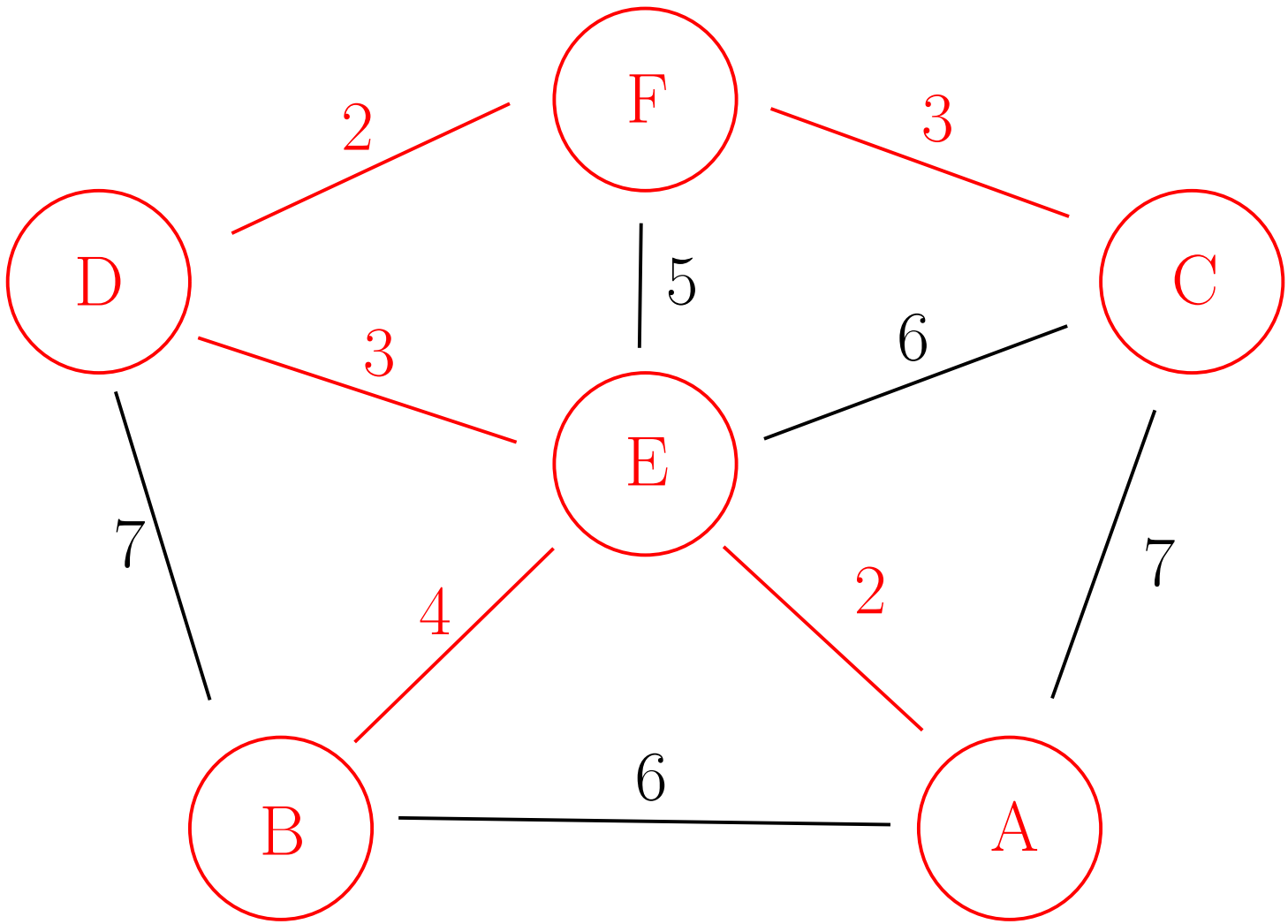
Durchführung



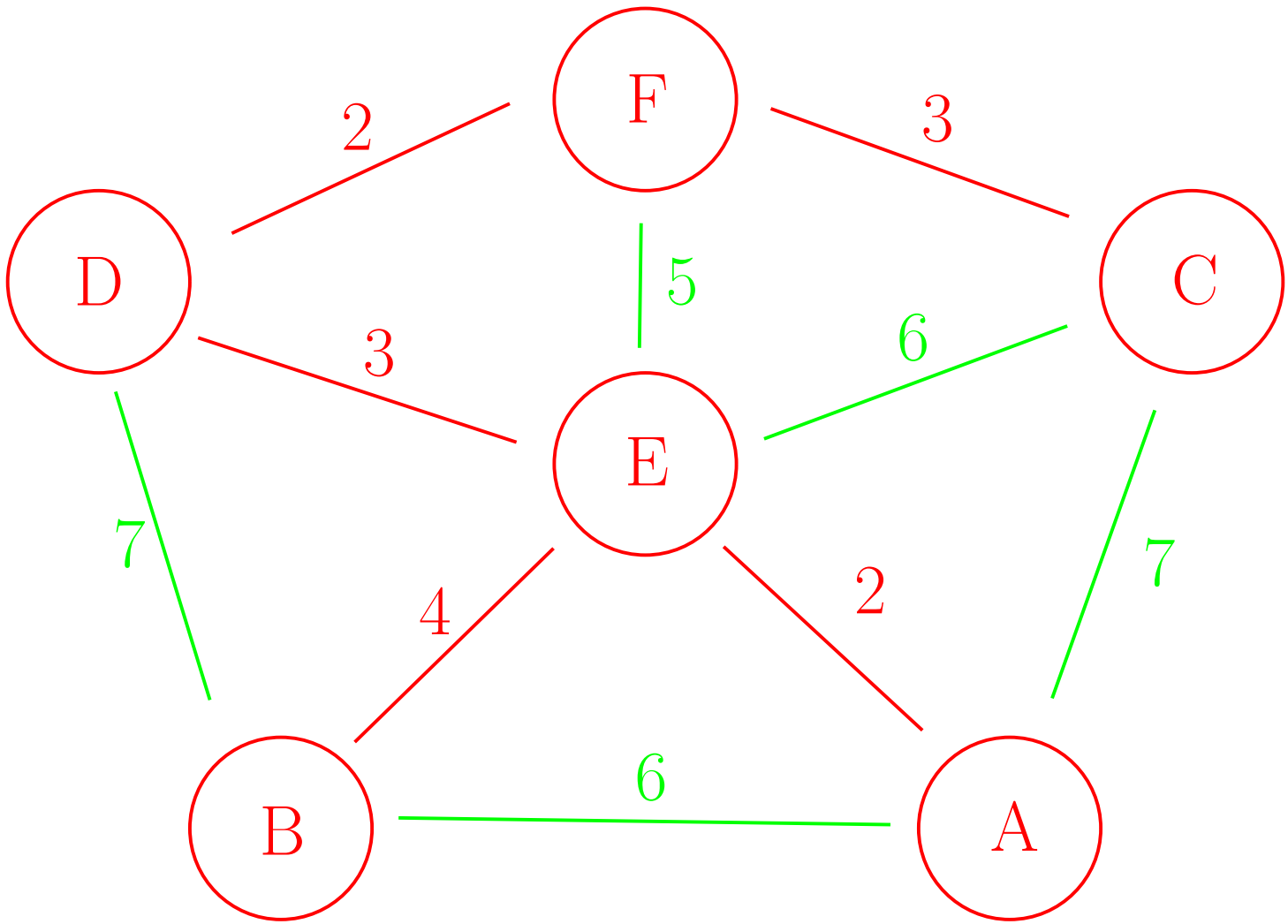
Durchführung



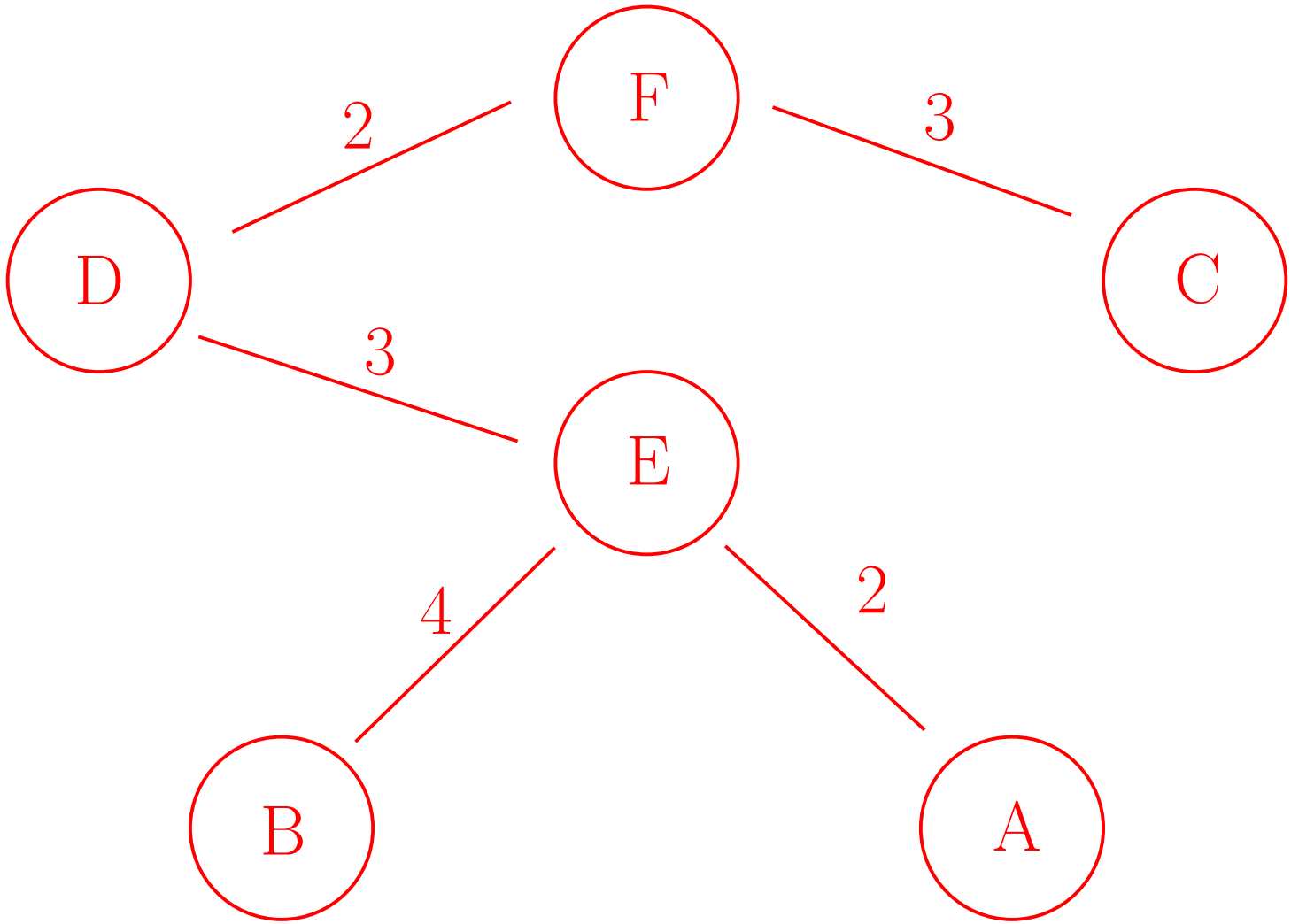
Durchführung



Durchführung



Durchführung



Operationen auf disjunkte Mengen

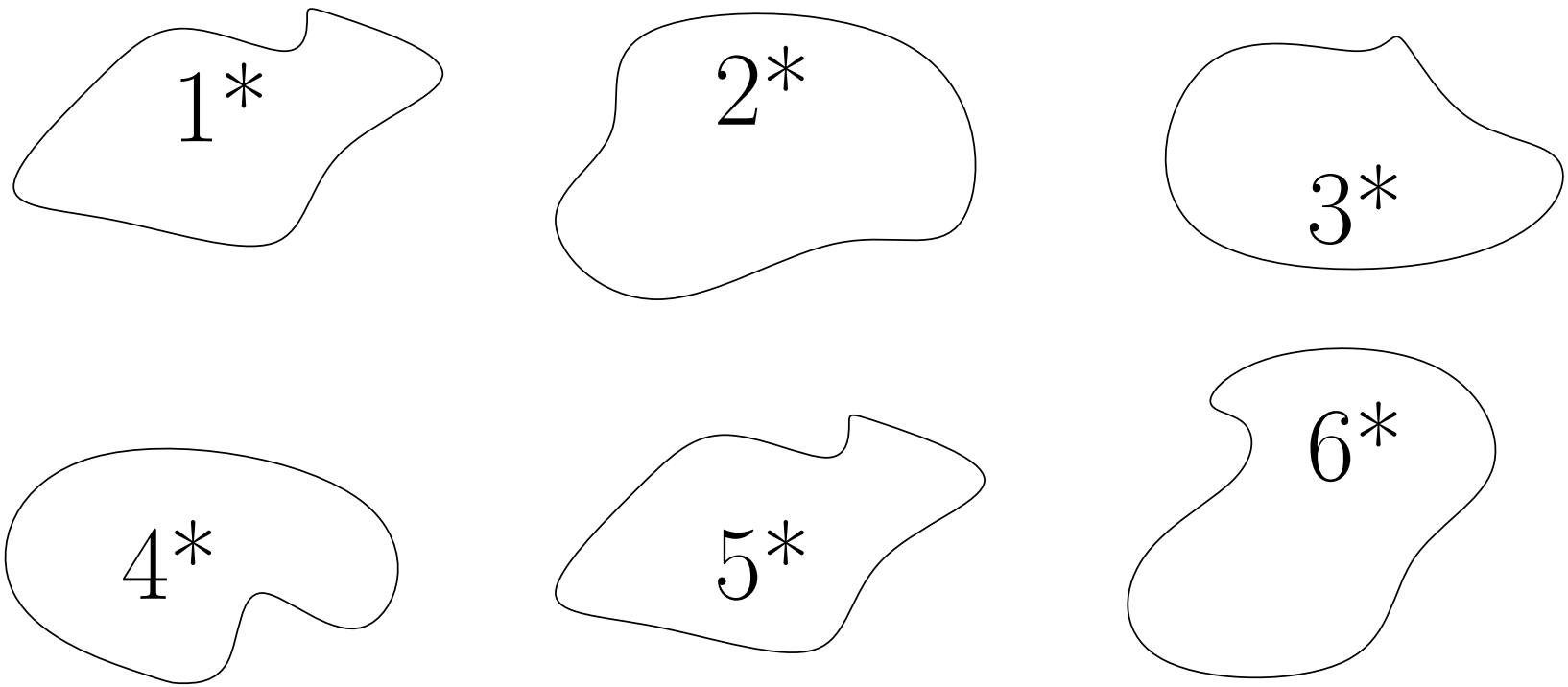
Operationen auf disjunkte Mengen

- **makeSet(Object)** Erstelle ein neue Menge A := $\{$Object$\}$
- **union(Object1, Object2)** Sei Object1 $\in$ A, Object2 $\in$ B $\Rightarrow$ A $\cup$ B.
- **findSet(Object)** Für Object $\in$ A, gib Repräsentanten von A zurück.

Beispiel

Operationen auf disjunkte Mengen

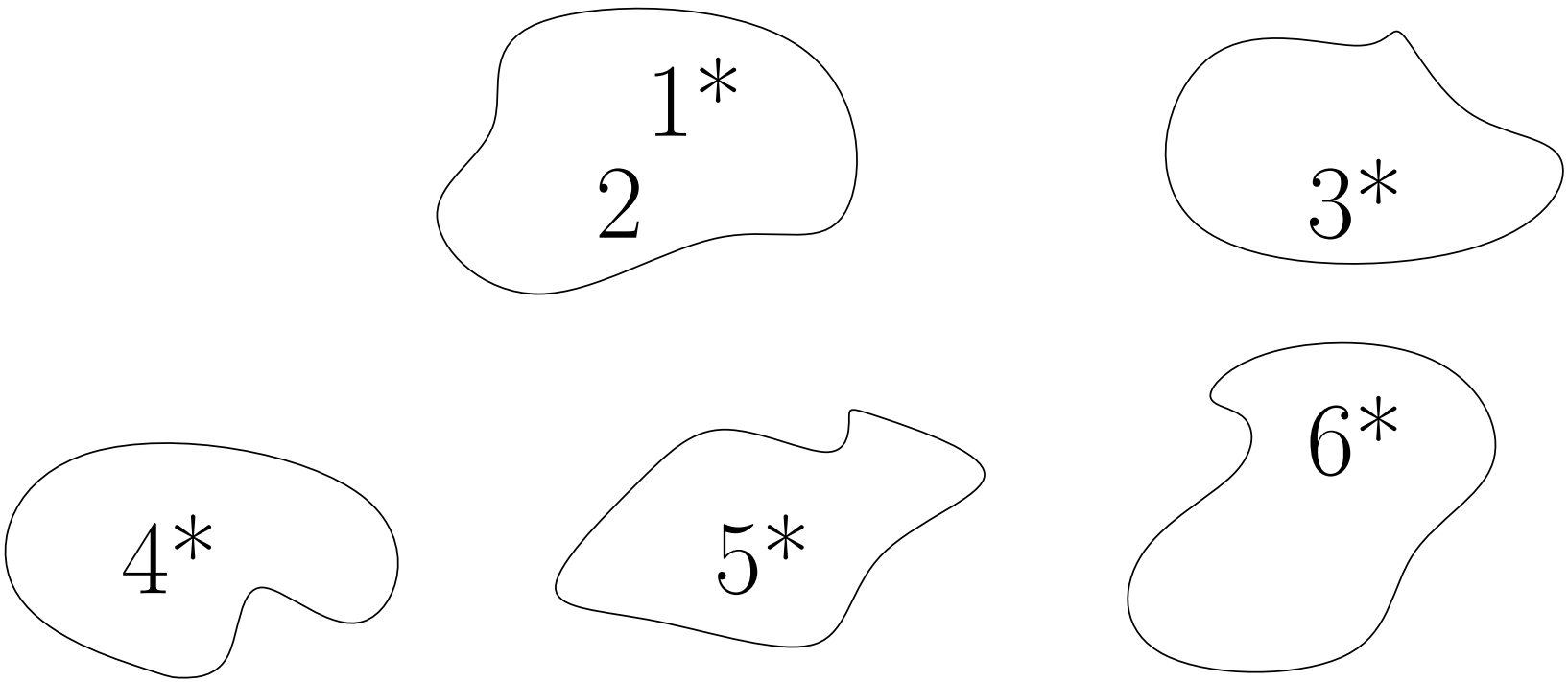
- makeSet(i) für $1 \leq i \leq 6$
- union(1,2)



Beispiel

Operationen auf disjunkte Mengen

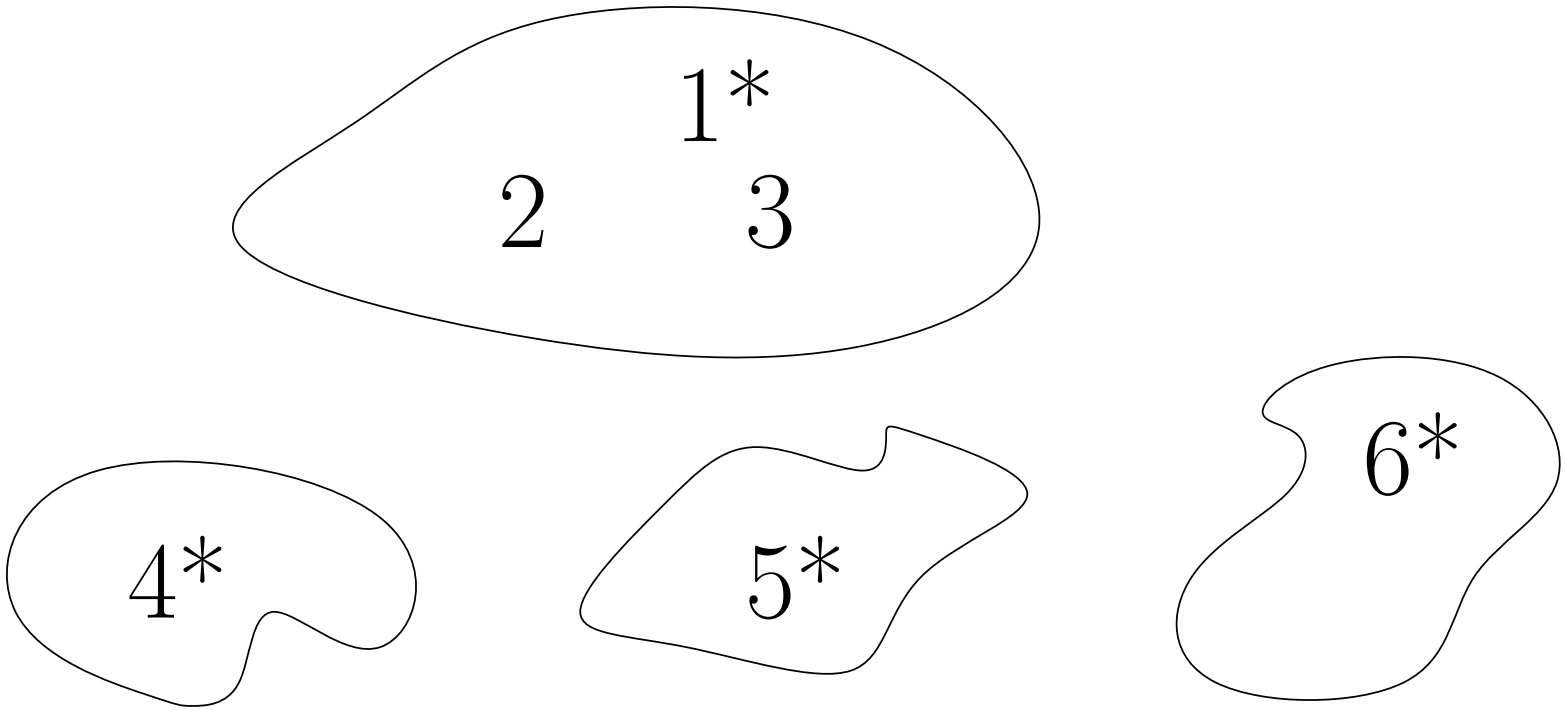
- union(1,2)
- union(2,3)



Beispiel

Operationen auf disjunkte Mengen

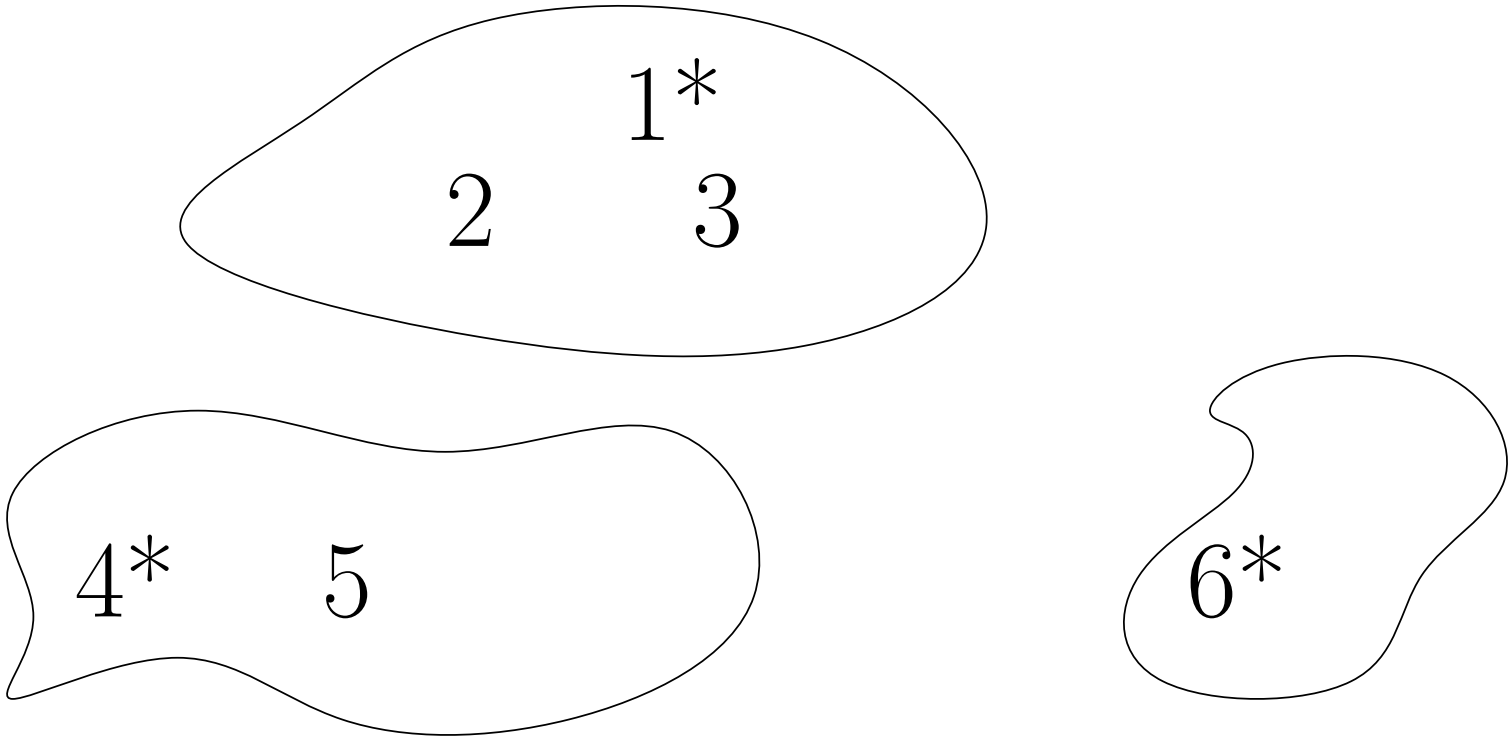
- union(2,3)
- union(4,5)



Beispiel

Operationen auf disjunkte Mengen

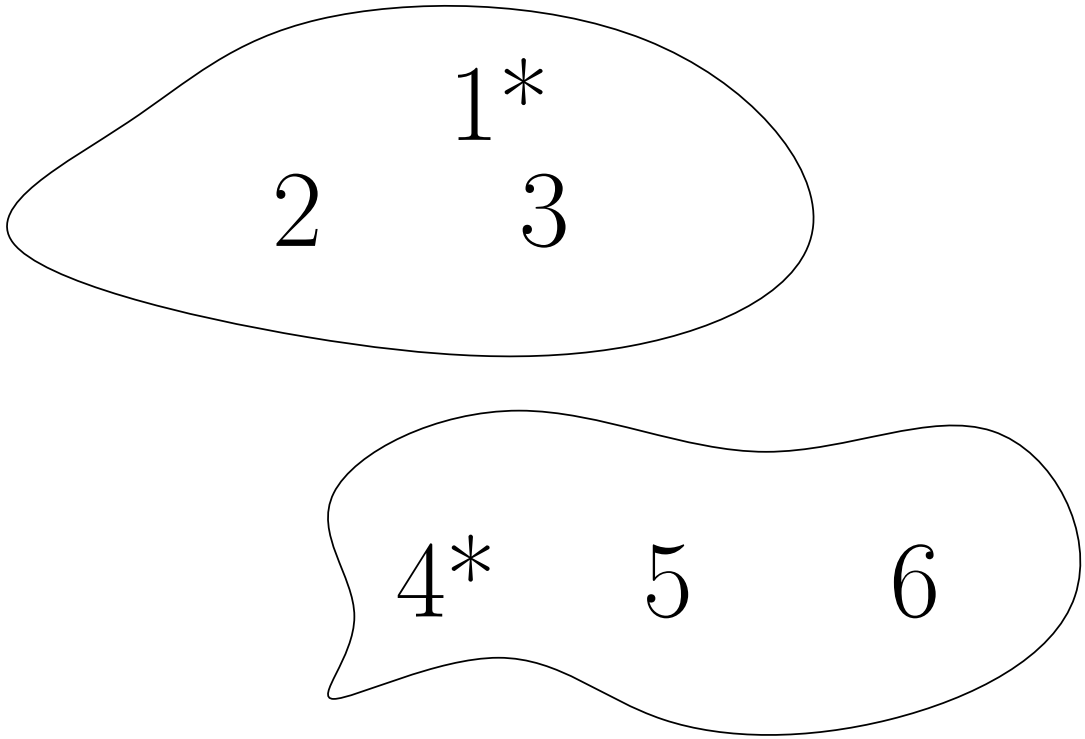
- union(4,5)
- union(5,6)



Beispiel

Operationen auf disjunkte Mengen

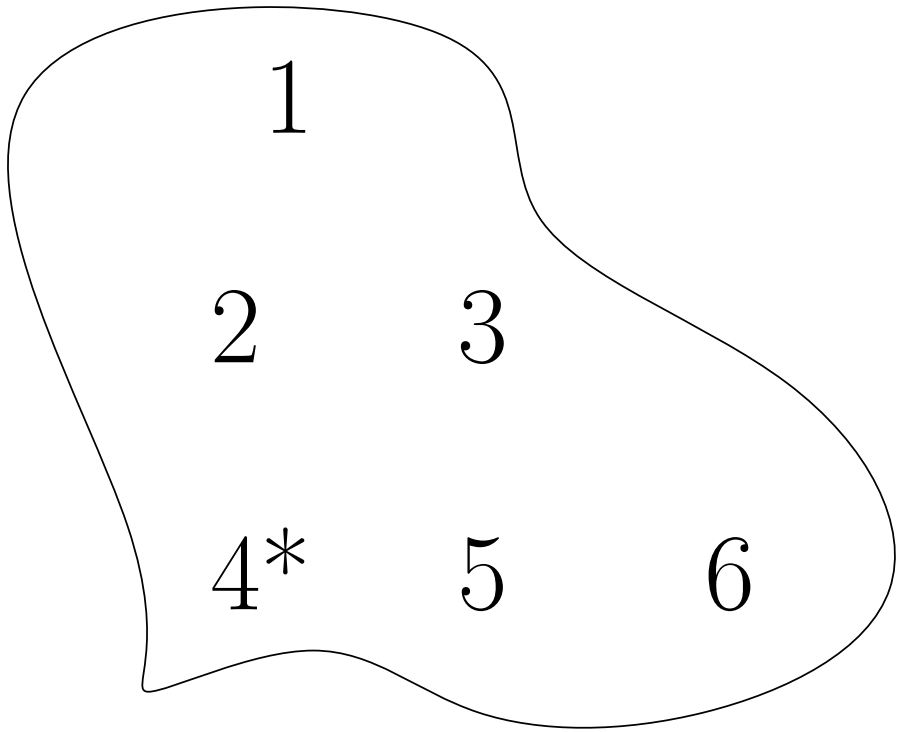
- union(5,6)
- union(5,3)



Beispiel

Operationen auf disjunkte Mengen

- union(5,3)



Implementieren der Union Find Struktur

Karlsruher Institut für Technologie

Idee:

- Jede Menge wird durch ein Baum beschrieben
- Der Repräsentanten ist die Wurzel des Baumes

Node:

- Object data;
- Node parent;

Implementieren der Union Find Struktur

Idee:

- Jede Menge wird durch ein Baum beschrieben
- Der Repräsentanten ist die Wurzel des Baumes

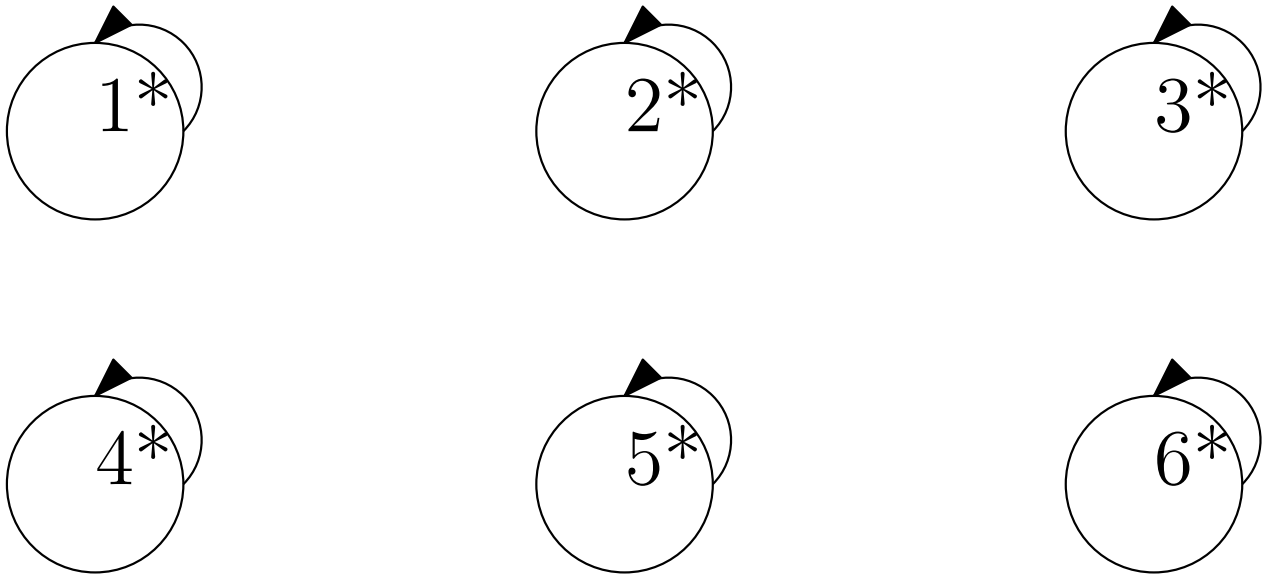
Node:

- Object data;
- Node parent;

Beispiel

Implementieren der Union Find Struktur

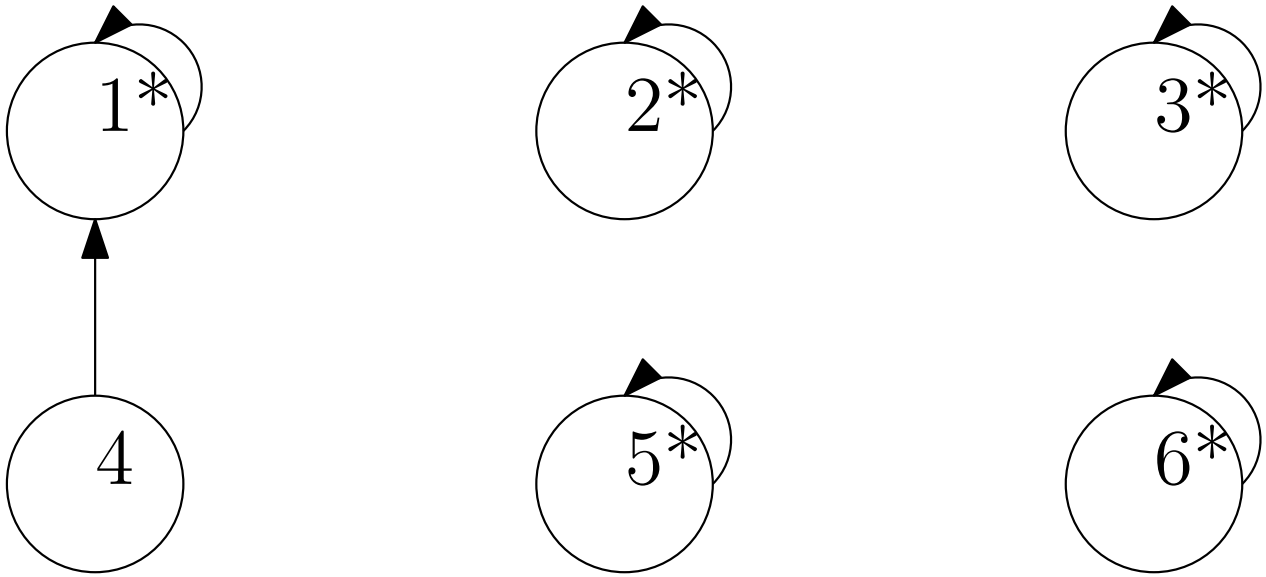
- makeSet(i) für $1 \leq i \leq 6$
- union(1,4)



Beispiel

Implementieren der Union Find Struktur

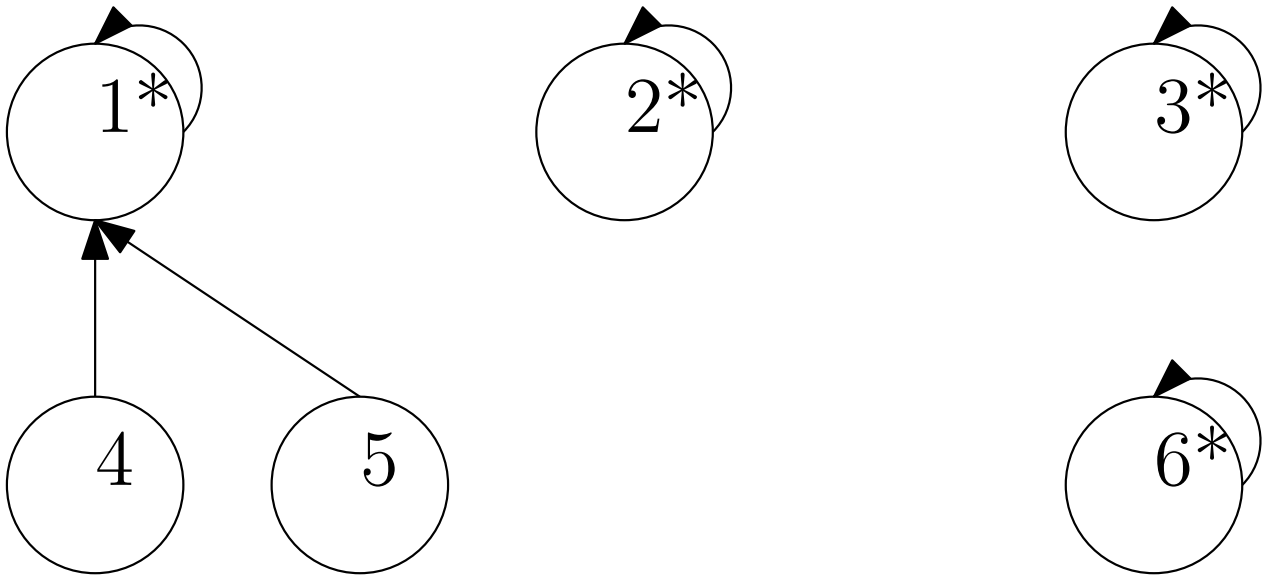
- union(1,4)
- union(1,5)



Beispiel

Implementieren der Union Find Struktur

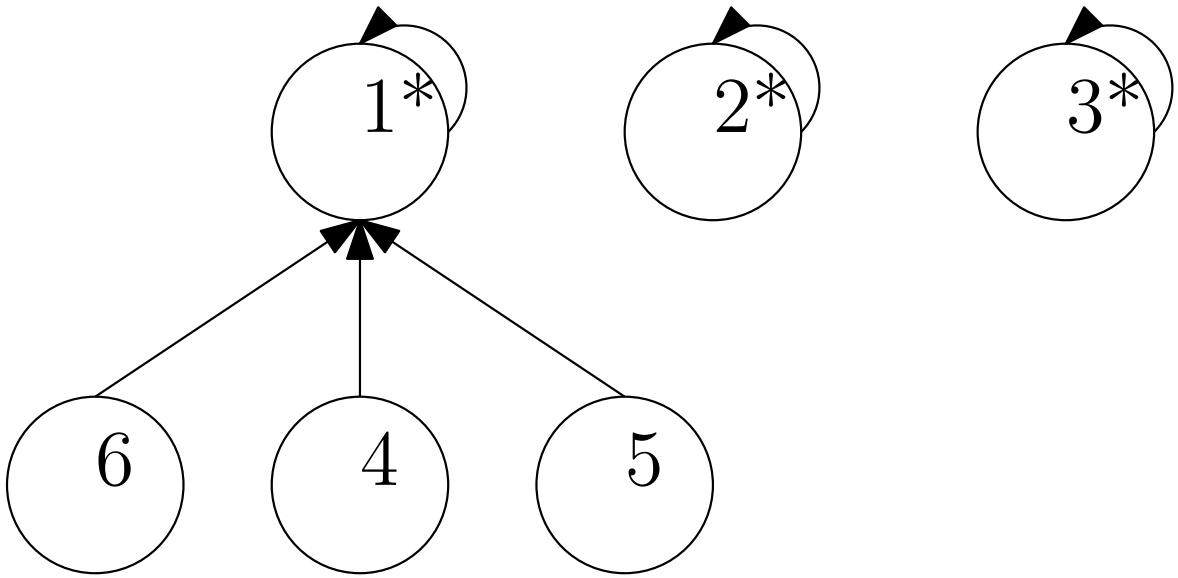
- union(1,5)
- union(1,6)



Beispiel

Implementieren der Union Find Struktur

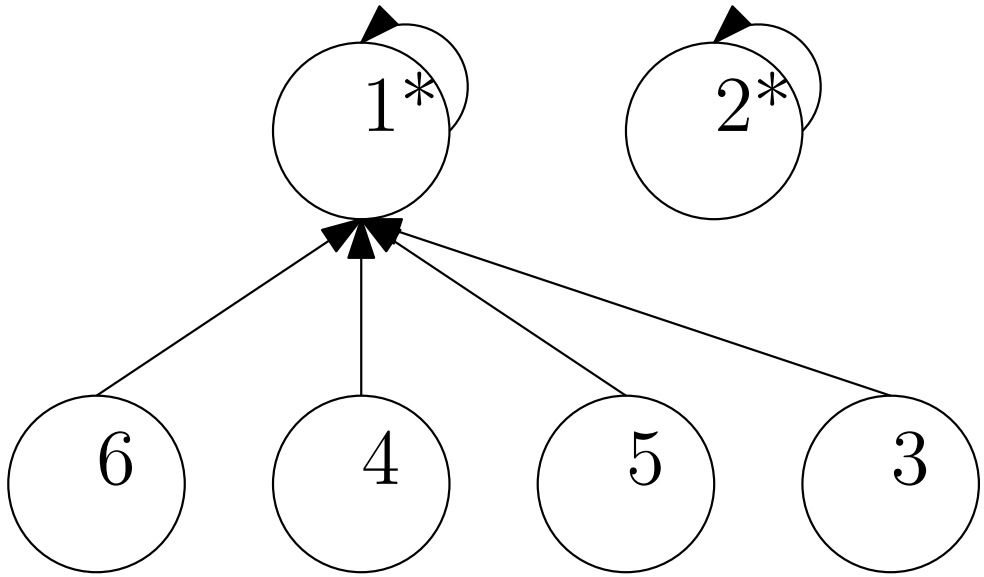
- union(1,6)
- union(1,3)



Beispiel

Implementieren der Union Find Struktur

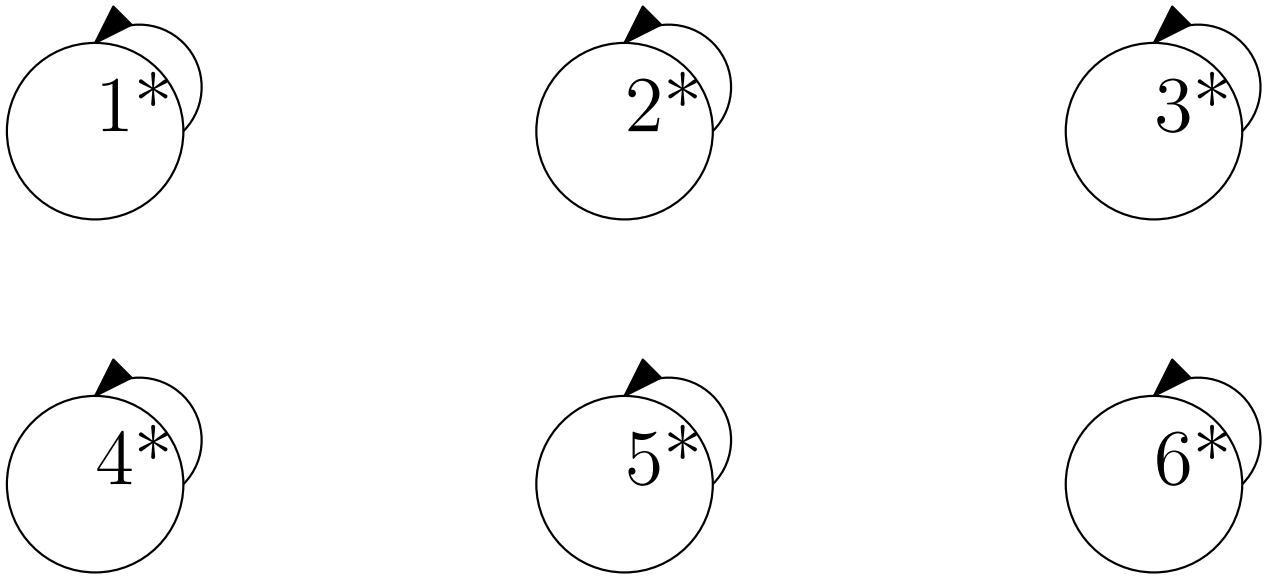
- union(1,3)
- findSet(a) für a $\in$ [1,6] braucht konstante Zeit



Beispiel - worst case

Implementieren der Union Find Struktur

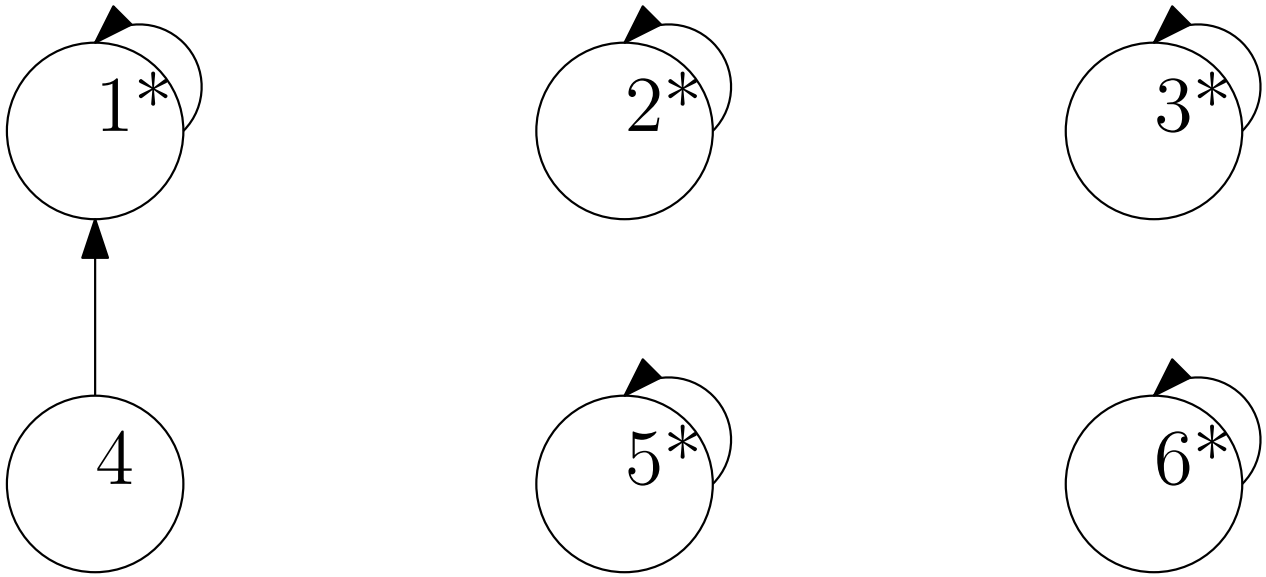
- makeSet(i) für $1 \leq i \leq 6$
- union(1,4)



Beispiel - worst case

Implementieren der Union Find Struktur

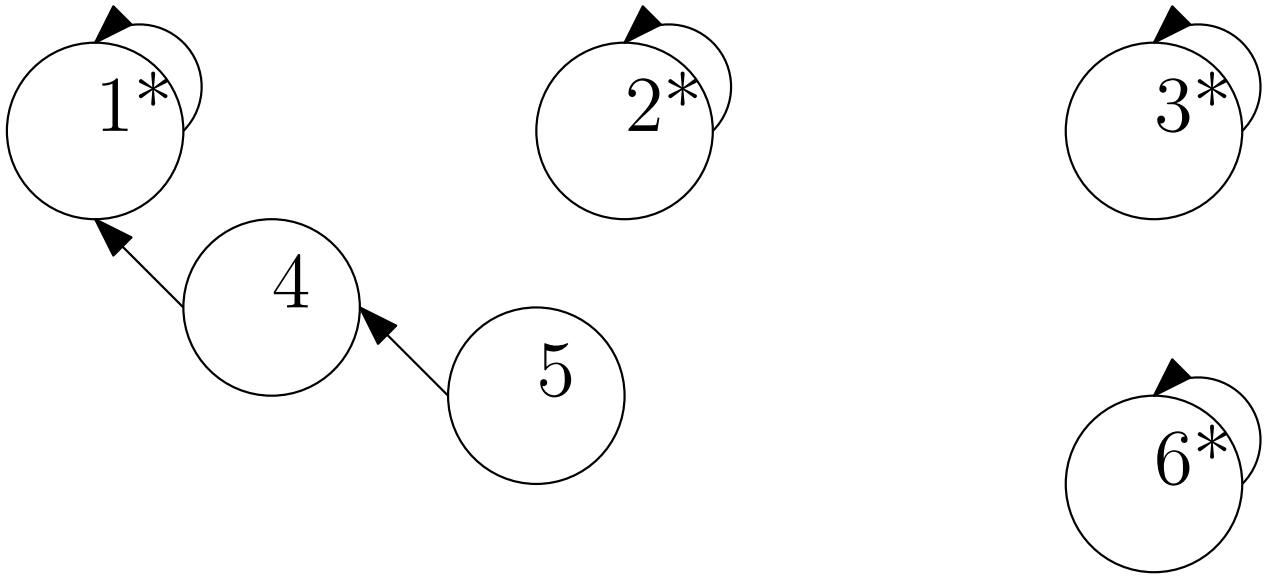
- union(1,4)
- union(4,5)



Beispiel - worst case

Implementieren der Union Find Struktur

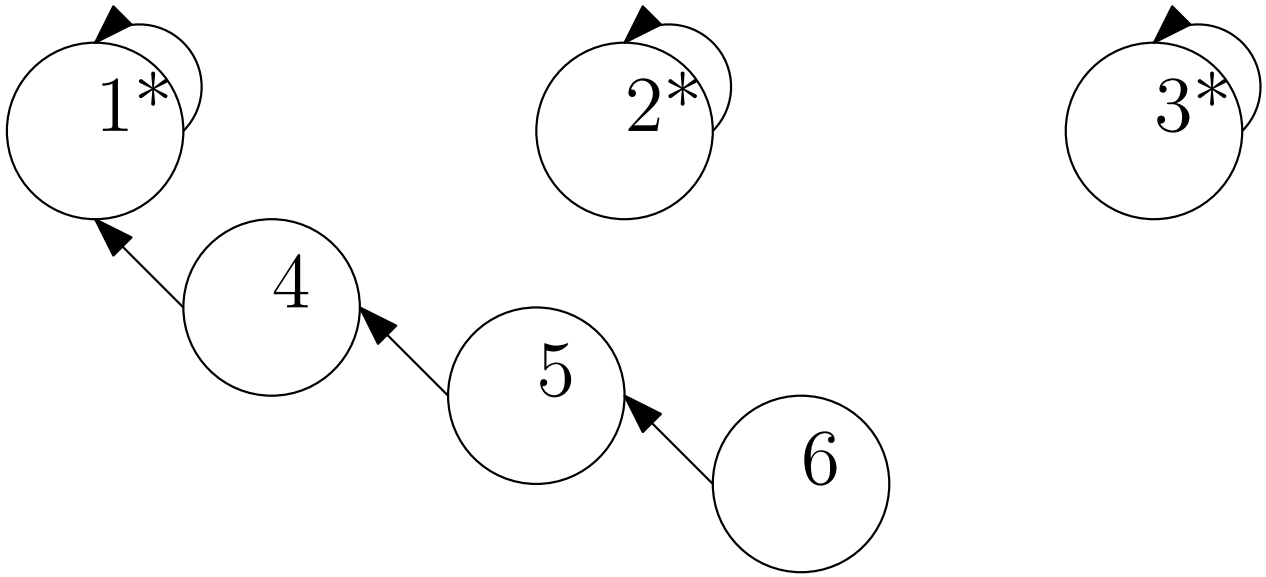
- union(4,5)
- union(5,6)



Beispiel - worst case

Implementieren der Union Find Struktur

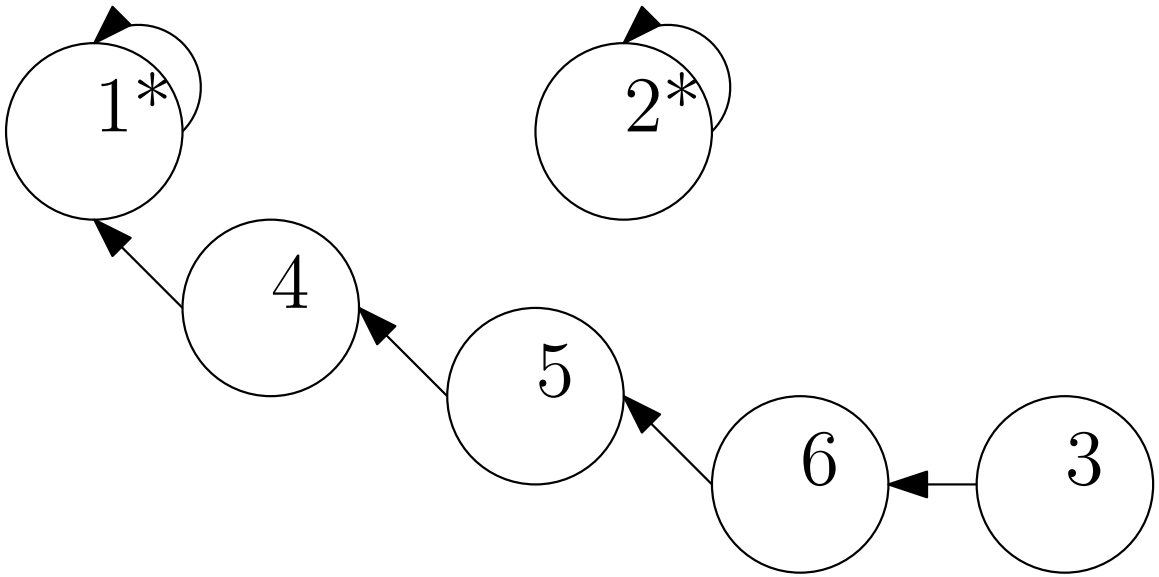
- union(5,6)
- union(6,3)



Beispiel - worst case

Implementieren der Union Find Struktur

- union(6,3)
- findSet(a) für a $\in$ [1,6] ist langsam



Pfadkompression

Ohne Optimierung

- makeSet(Object) in $O(1)$
- union(Object1, Object2) in $O(1)$
- findSet(Object) in $O(n)$

Pfadkompression

- Pfade werden nach dem findSet() Aufruf auf die Länge 1 reduziert
- Worst case:
  - makeSet(Object) in $O(1)$
  - union(Object1, Object2) in $O(1)$
  - findSet(Object) in $O(n \log n)$
- Amortisiert:
- Bei einer Folge von m findeSet()- und n-1 Union()-Operationen in $O(n + m * \alpha(n))$ ≈O(n+m*4)

Pfadkompression

Ohne Optimierung

- makeSet(Object) in $O(1)$
- union(Object1, Object2) in $O(1)$
- findSet(Object) in $O(n)$

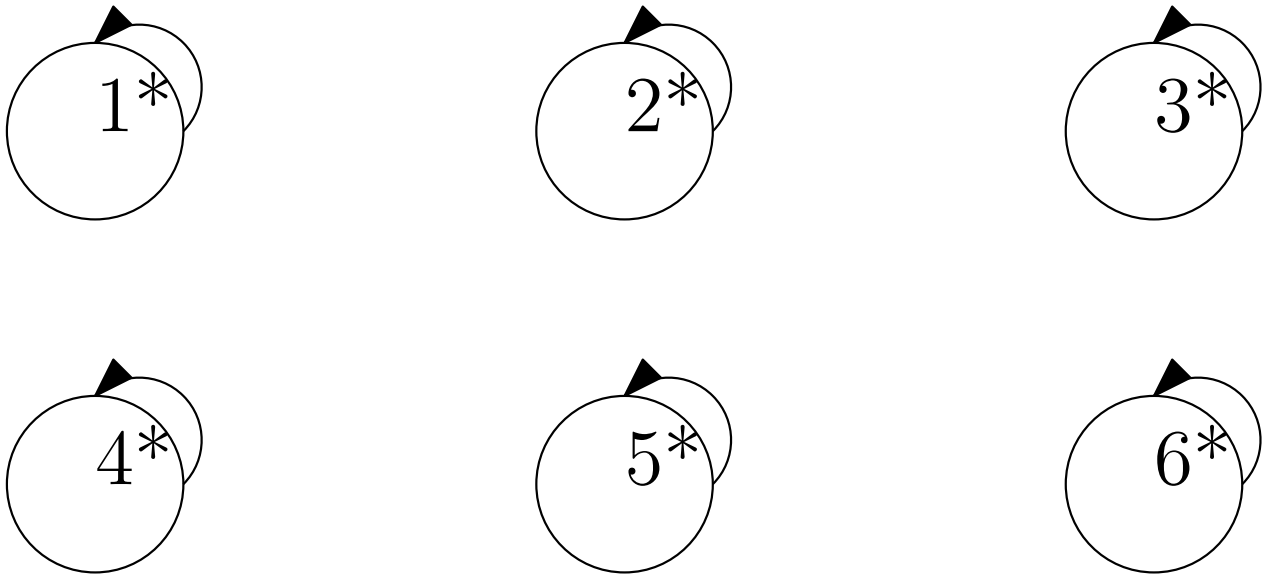
Pfadkompression

- Pfade werden nach dem findSet() Aufruf auf die Länge 1 reduziert
- Worst case:
  - makeSet(Object) in $O(1)$
  - union(Object1, Object2) in $O(1)$
  - findSet(Object) in $O(n \log n)$
- Amortisiert:
- Bei einer Folge von m findeSet()- und n-1 Union()-Operationen in $O(n + m * \alpha(n))$ ≈O(n+m*4)

Beispiel - Pfadkompression (1)

Implementieren der Union Find Struktur

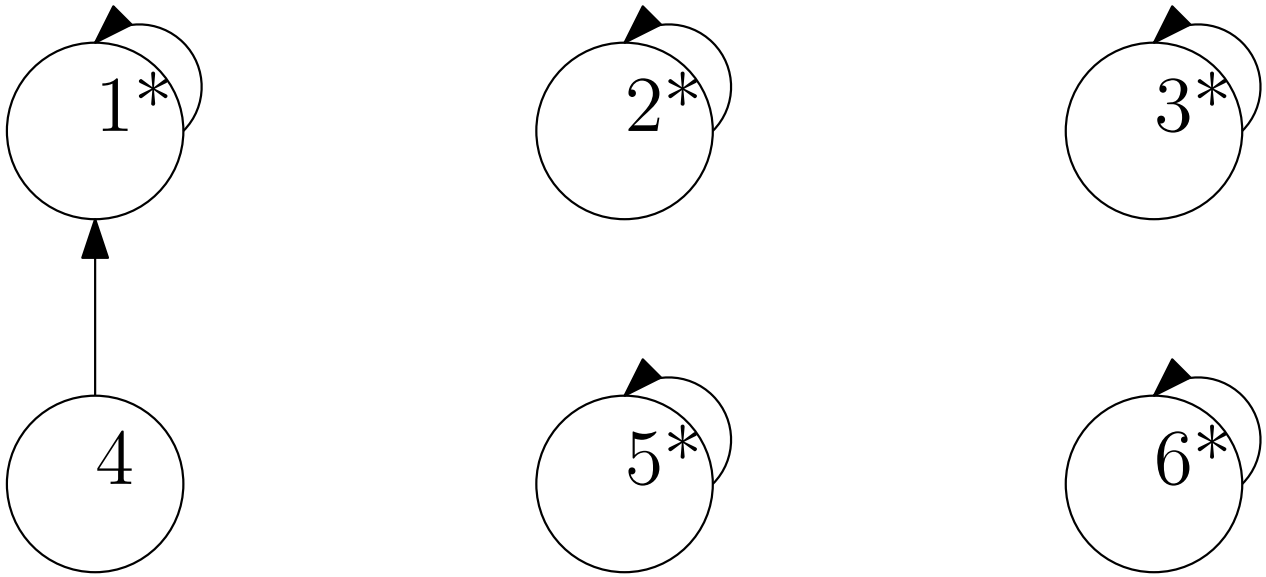
- makeSet(i) für $1 \leq i \leq 6$
- union(1,4)



Beispiel - Pfadkompression (1)

Implementieren der Union Find Struktur

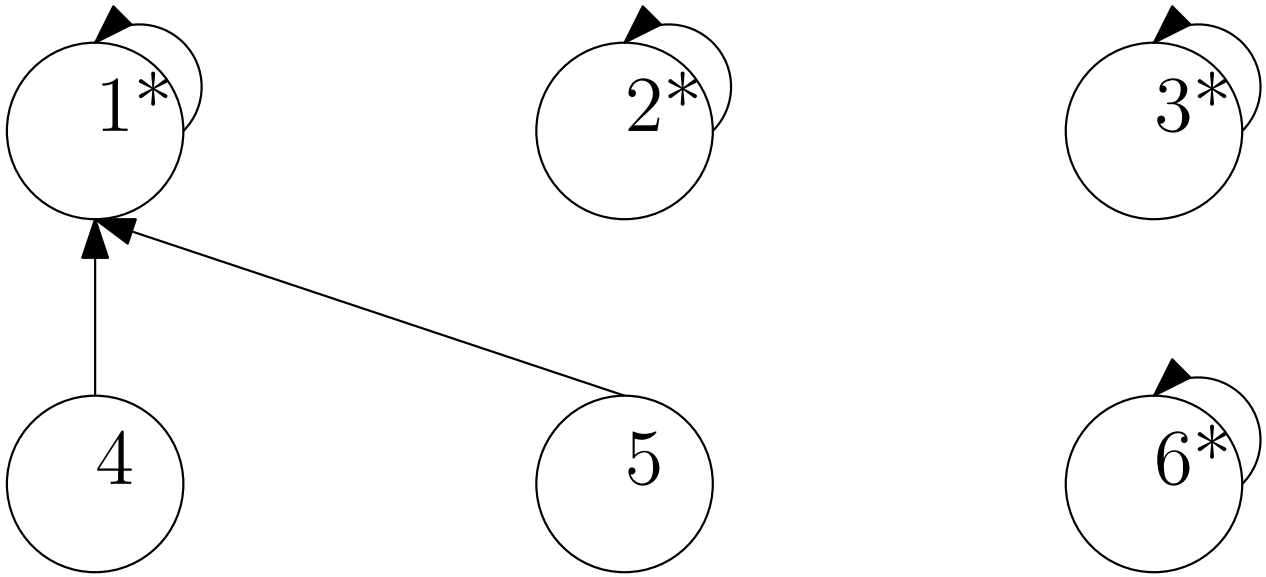
- union(1,4)
- union(4,5)



Beispiel - Pfadkompression (1)

Implementieren der Union Find Struktur

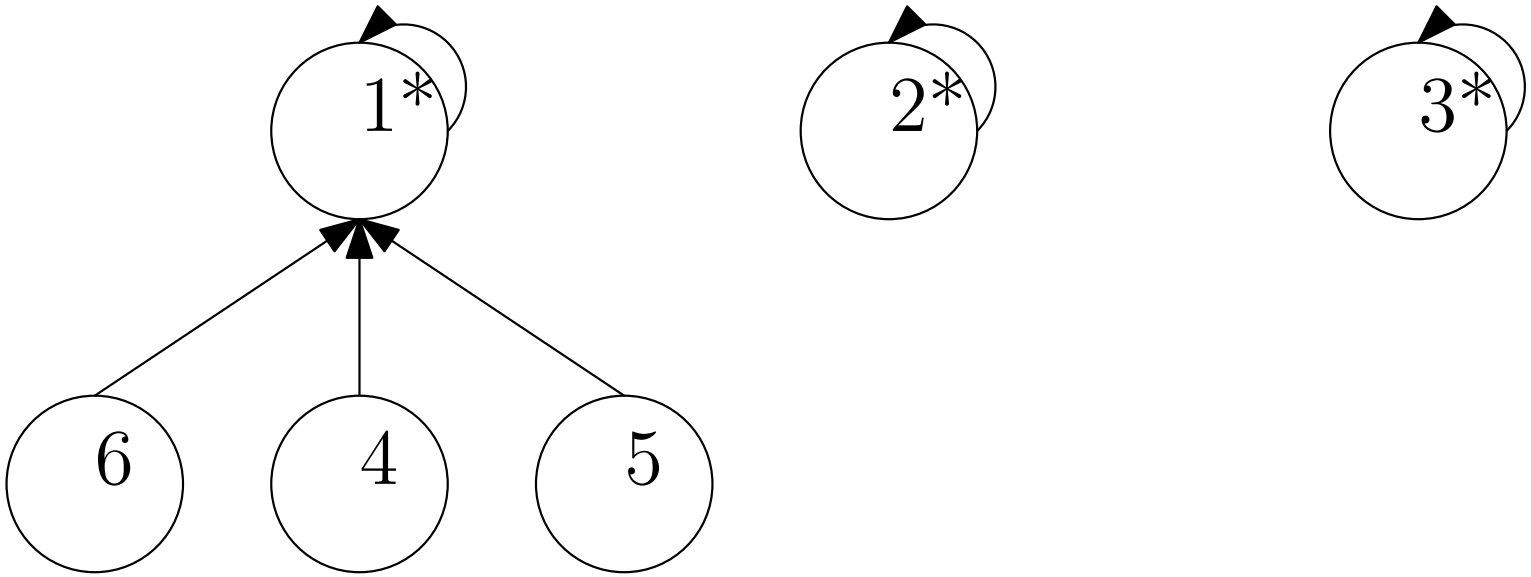
- union(4,5)
- union(5,6)



Beispiel - Pfadkompression (1)

Implementieren der Union Find Struktur

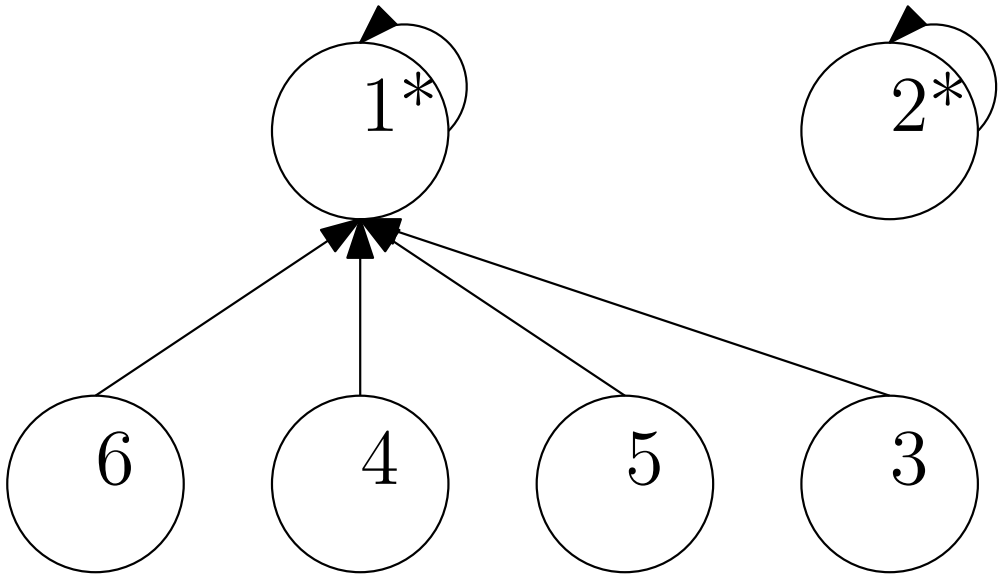
- union(5,6)
- union(6,3)



Beispiel - Pfadkompression (1)

Implementieren der Union Find Struktur

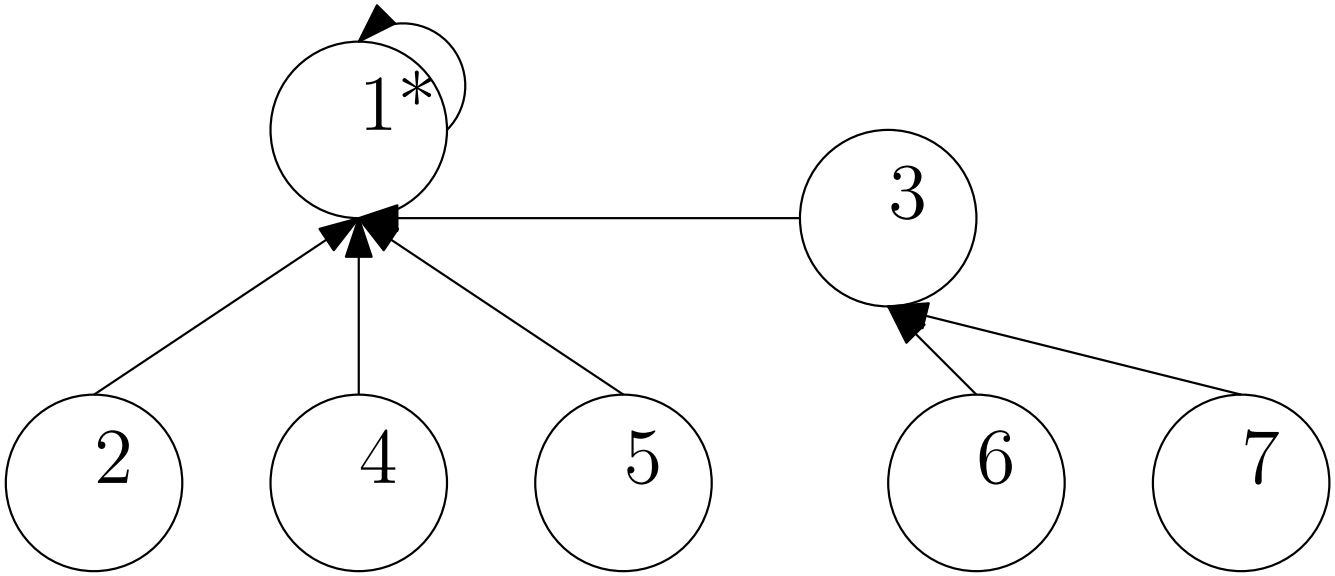
- union(6,3)
- findSet(a) für a $\in$ [1,6] ist schnell



Beispiel - Pfadkompression (2)

Implementieren der Union Find Struktur

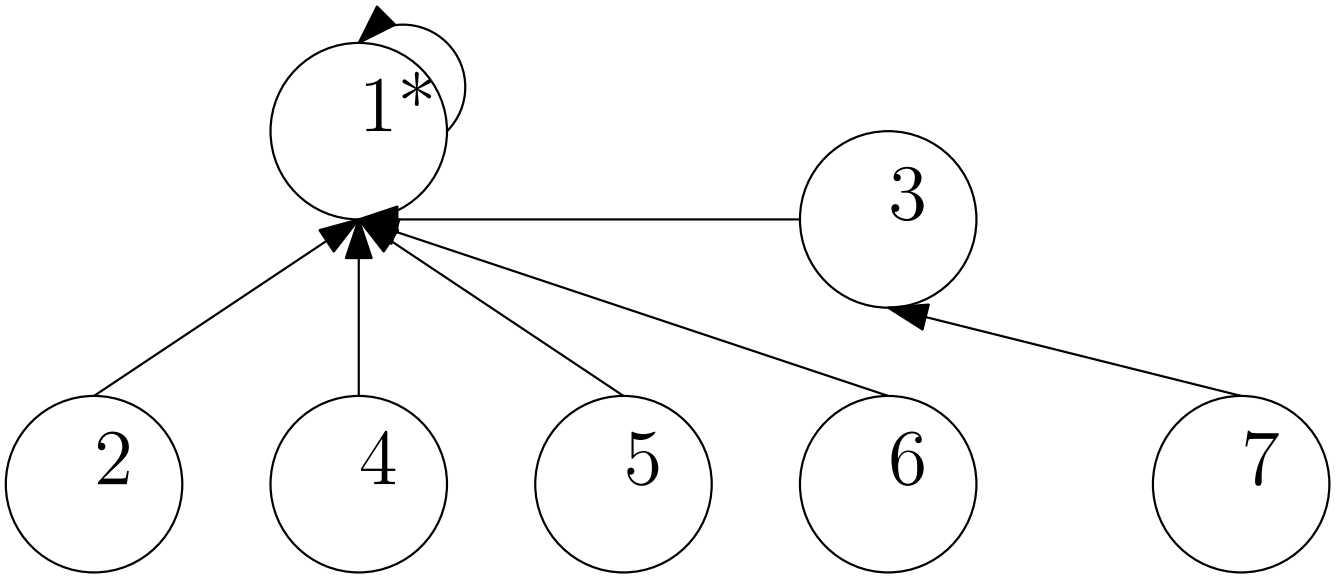
- makeSet(i) für $1 \leq i \leq 6$
- union(1,2)...union(1,3)
- findSet(6)



Beispiel - Pfadkompression (2)

Implementieren der Union Find Struktur

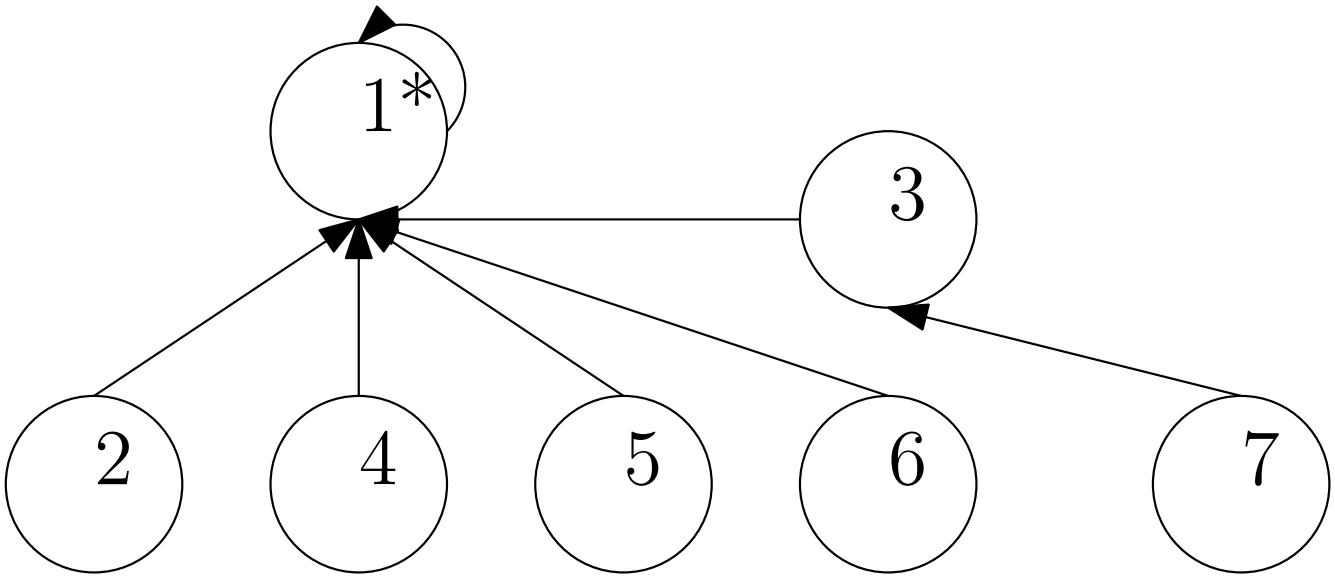
- findSet(6)
- findSet(3)



Beispiel - Pfadkompression (2)

Implementieren der Union Find Struktur

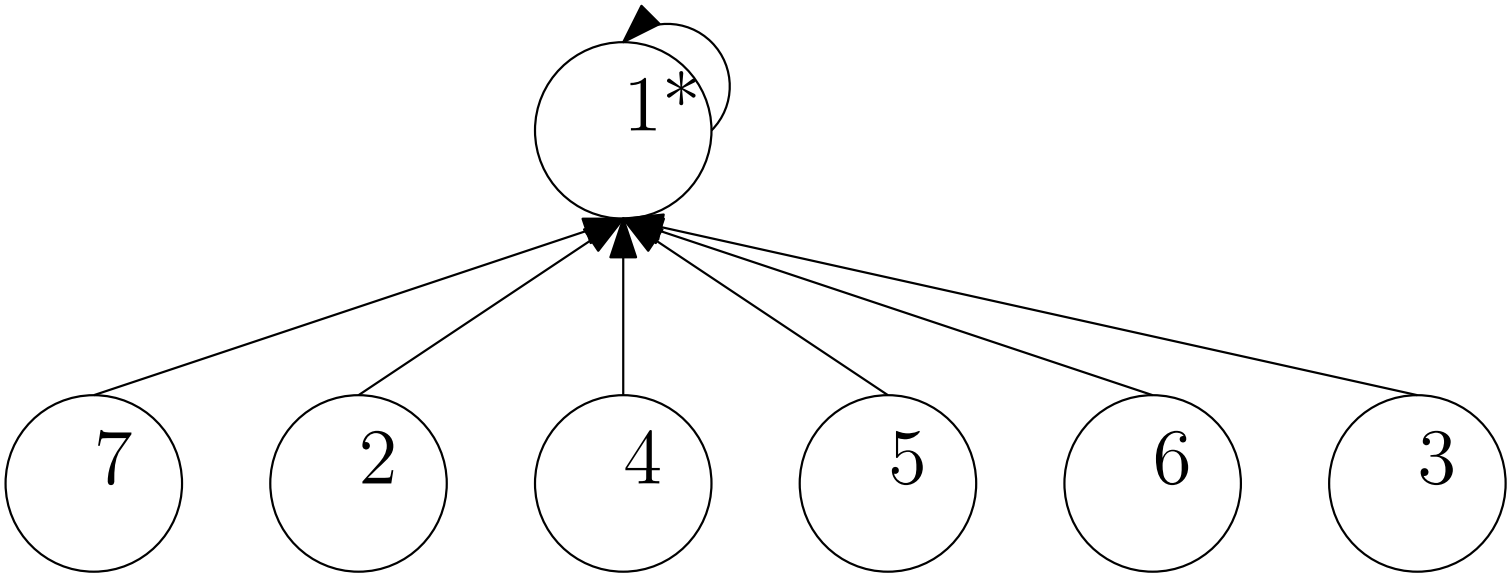
- findSet(3)
- findSet(7)



Beispiel - Pfadkompression (2)

Implementieren der Union Find Struktur

- findSet(7)



Pfadkompression

Pfadkompression

- Pfade werden nach dem findSet() Aufruf auf die Länge 1 reduziert
- Worst case:
  - makeSet(Object) in $O(1)$
  - union(Object1, Object2) in $O(1)$
  - findSet(Object) in $O(n \log n)$
- Amortisiert:
- Bei einer Folge von m findeSet()- und n-1 Union()-Operationen in $O(n + m * \alpha(n)) \approx$O(n+m*4)

Minimale Spannbäume

Kruskals Algorithmus

- Berechnet ebenfalls einen MST des gegebenen Graphen
- **Aber:** Arbeitet auf einer sortierten Kantenliste

Vorgehen

Wähle in jedem Schritt die leichteste Kante aus, die mit den bisher gewählten Kanten keinen Kreis bildet.

Minimale Spannbäume

Kruskals Algorithmus

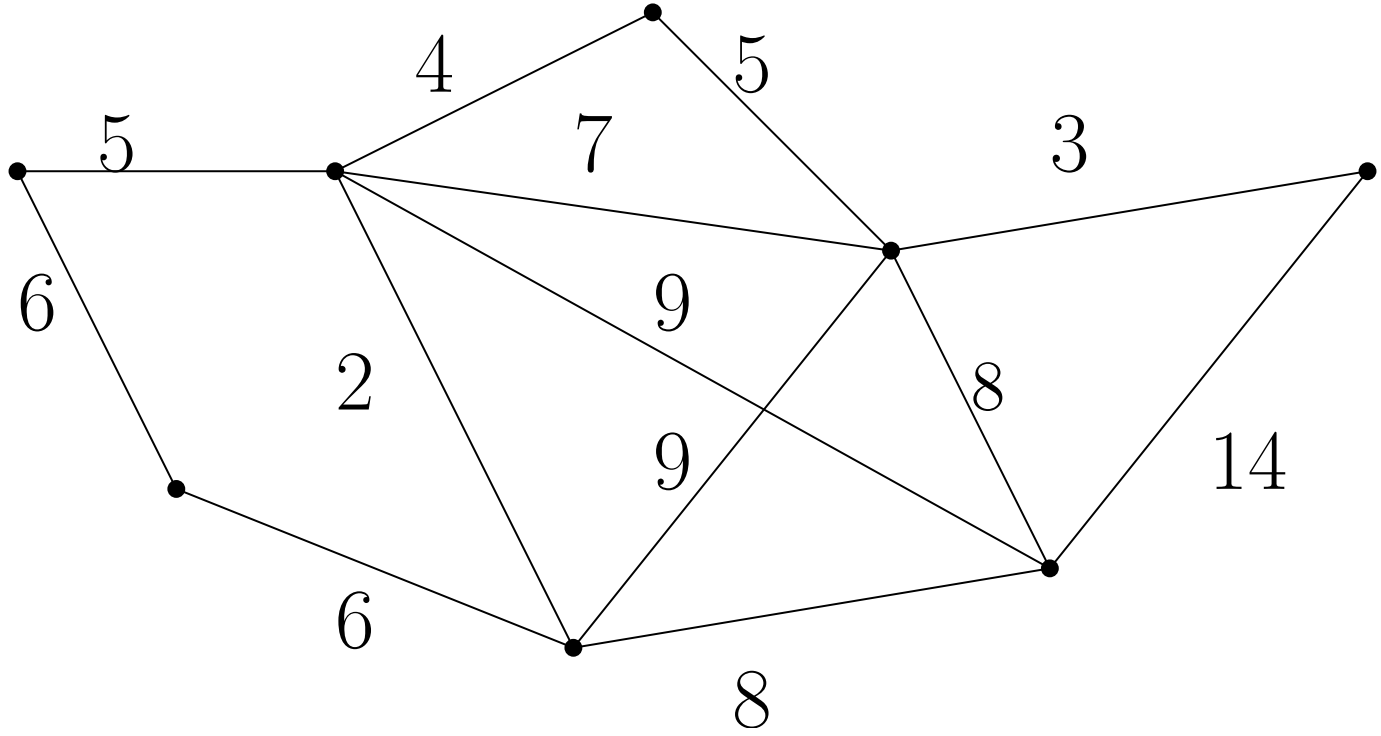
- Berechnet ebenfalls einen MST des gegebenen Graphen
- **Aber:** Arbeitet auf einer sortierten Kantenliste

Vorgehen

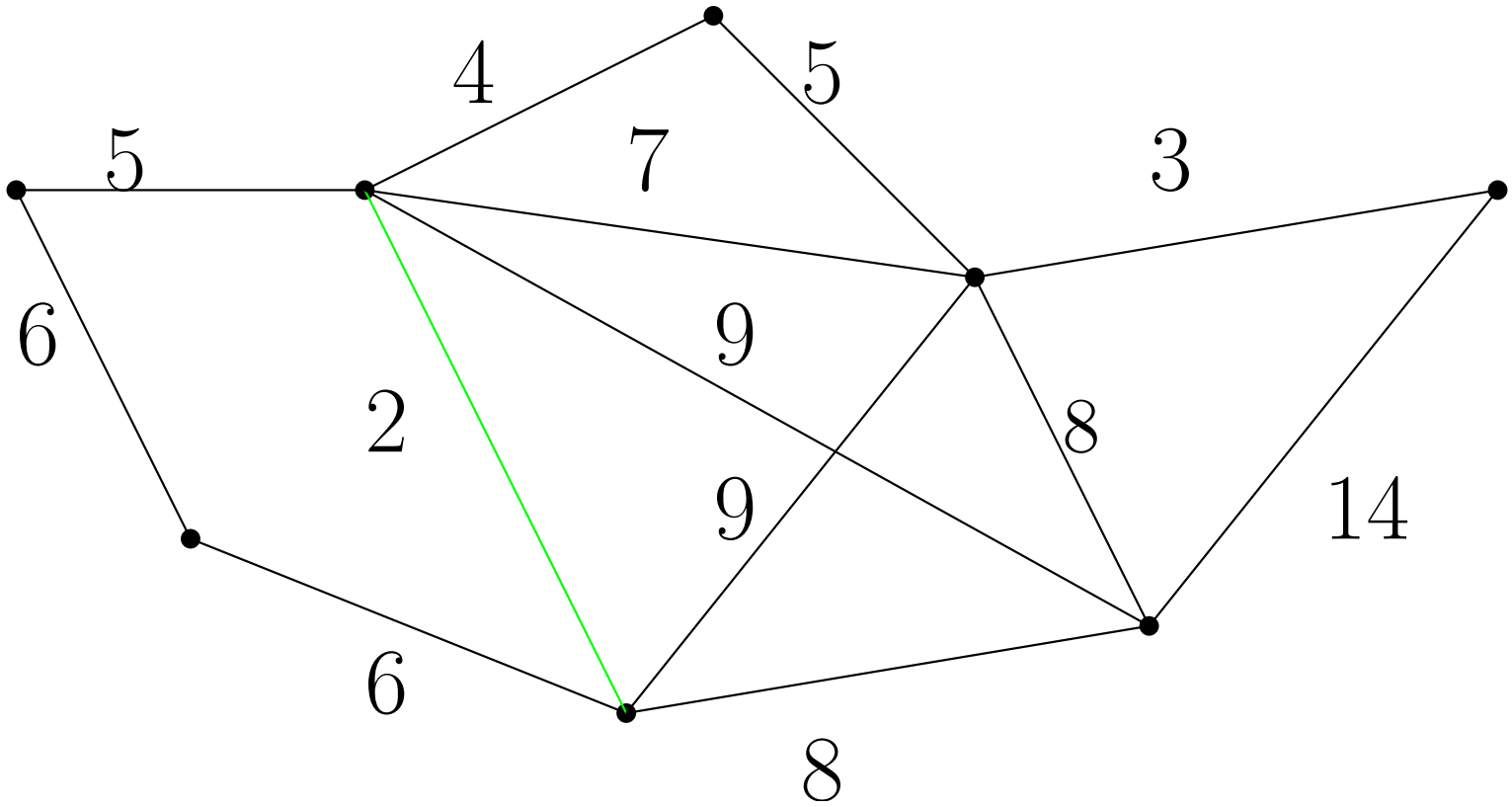
Wähle in jedem Schritt die leichteste Kante aus, die mit den bisher gewählten Kanten keinen Kreis bildet.

Beispiel

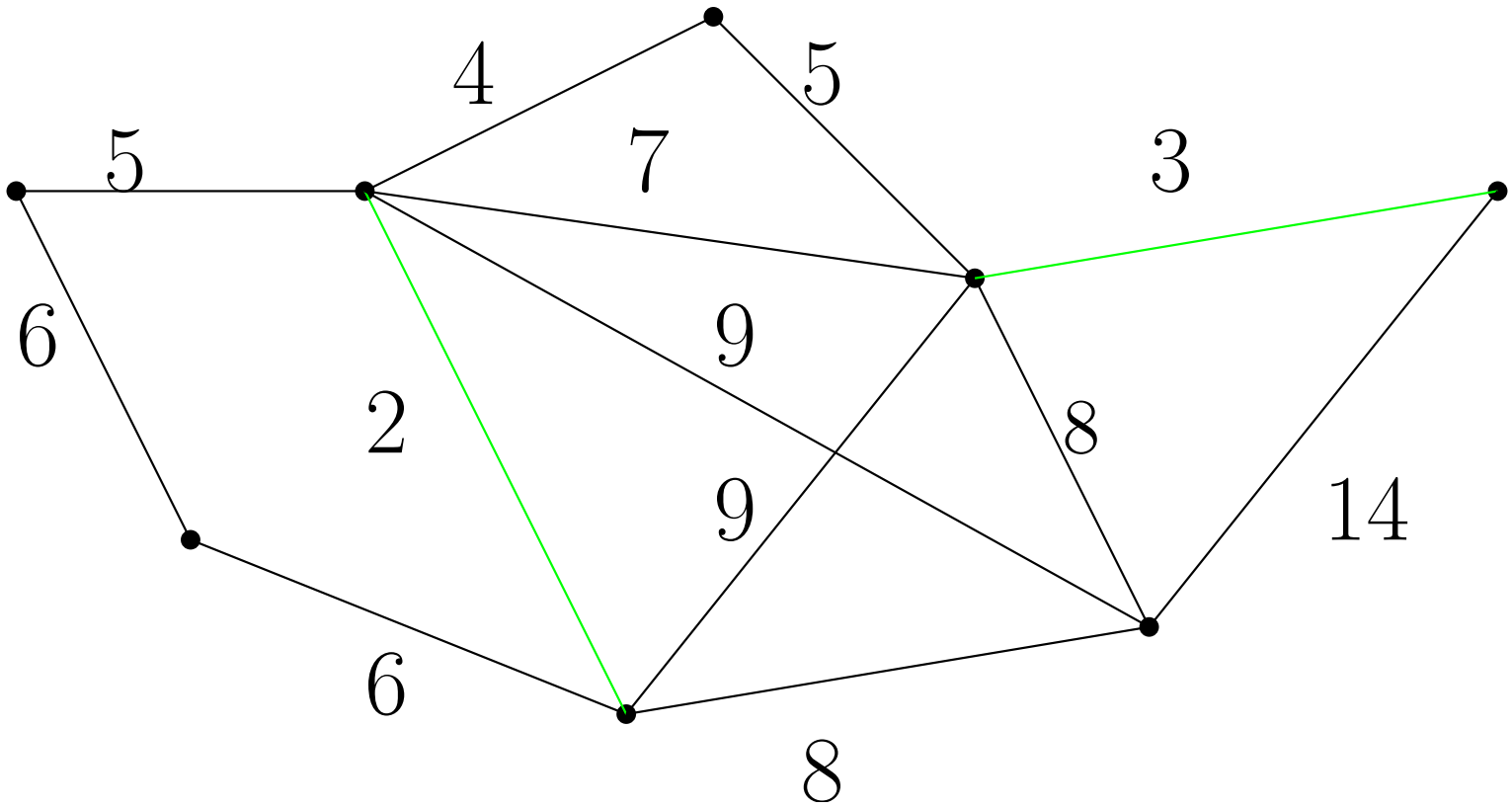
Berechne einen minimalen Spannbaum T zum Graphen G. G sei gegeben durch die folgende Grafik:



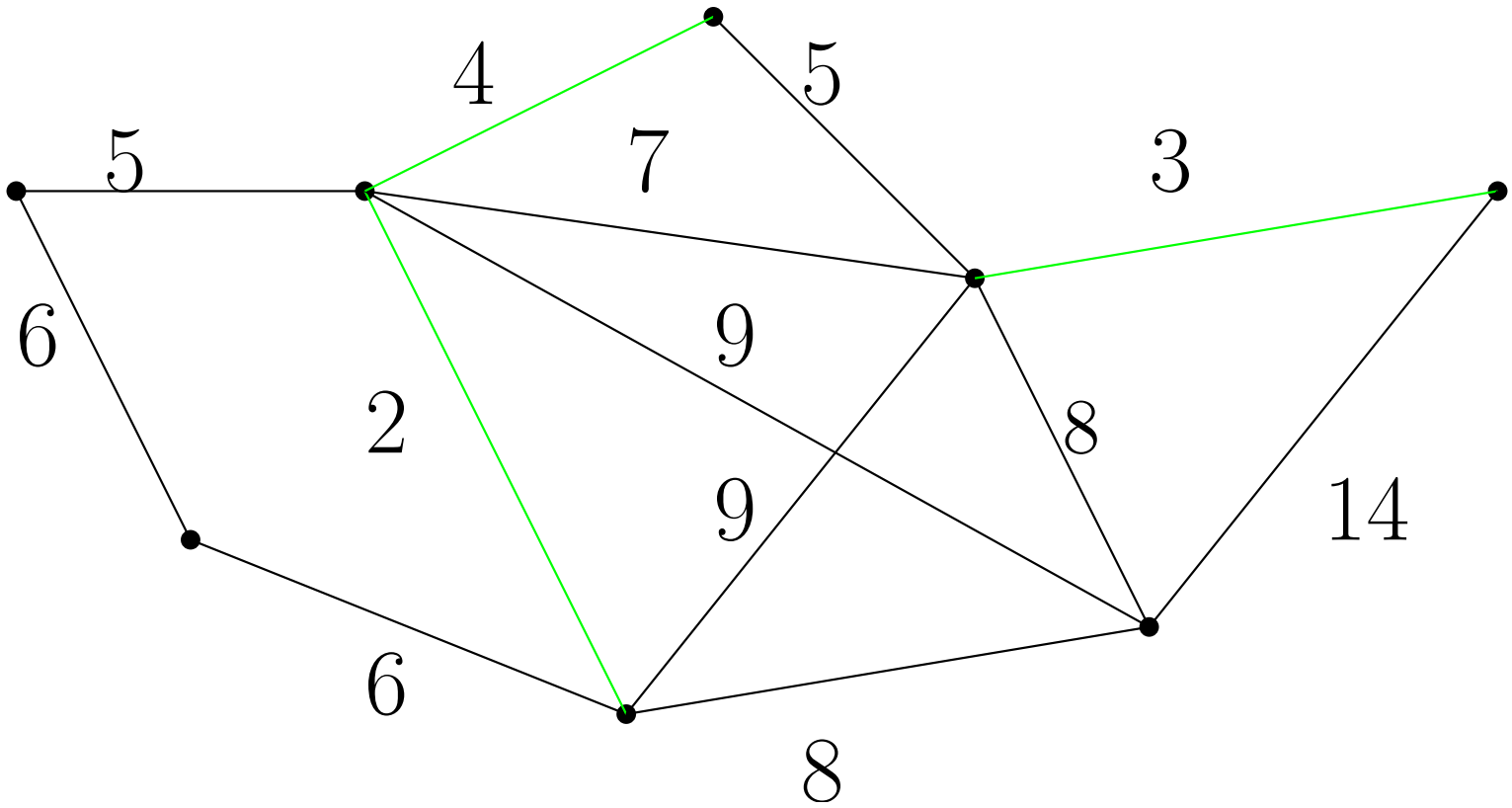
Lösung mit Kruskal



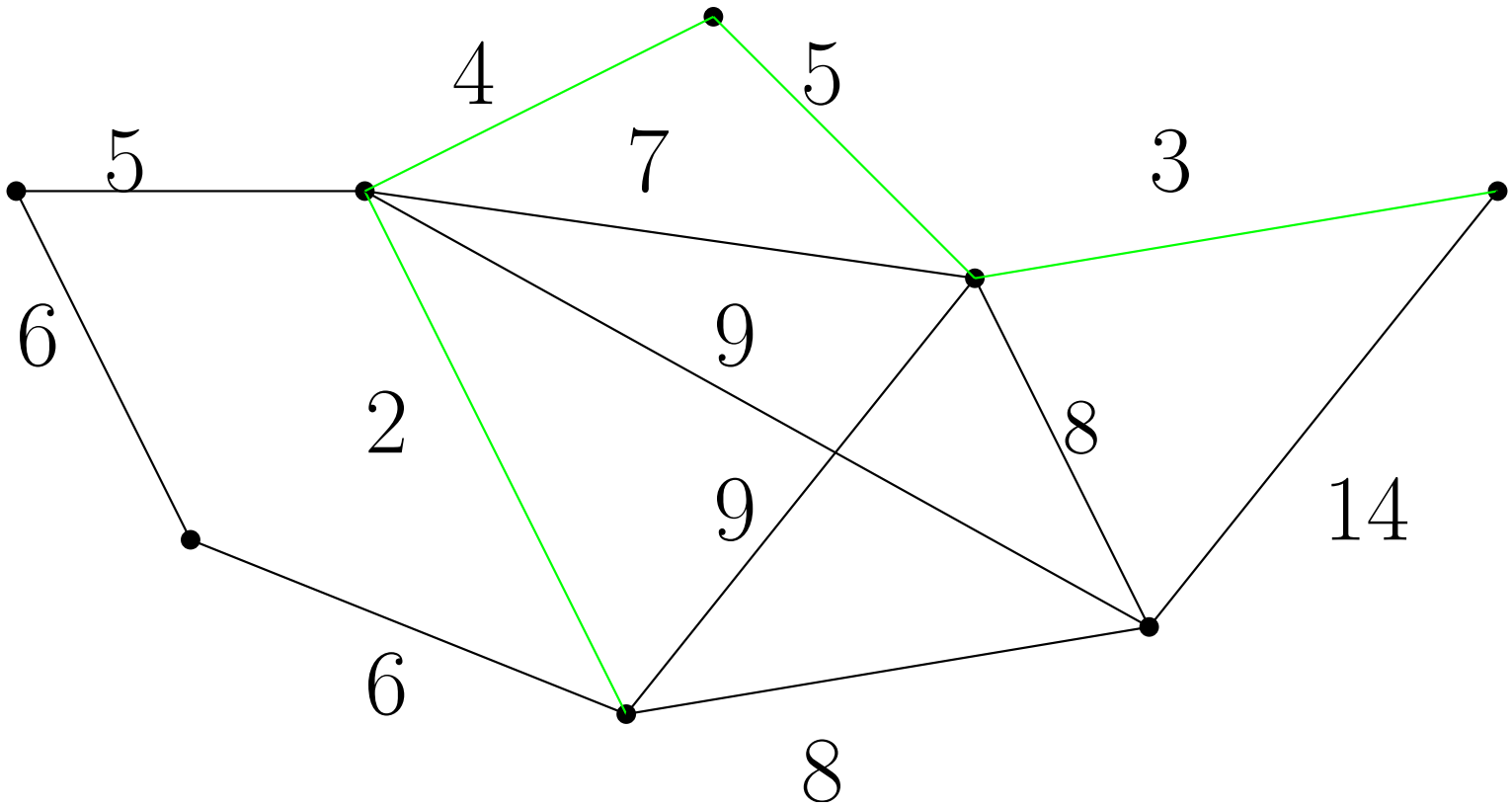
Lösung mit Kruskal



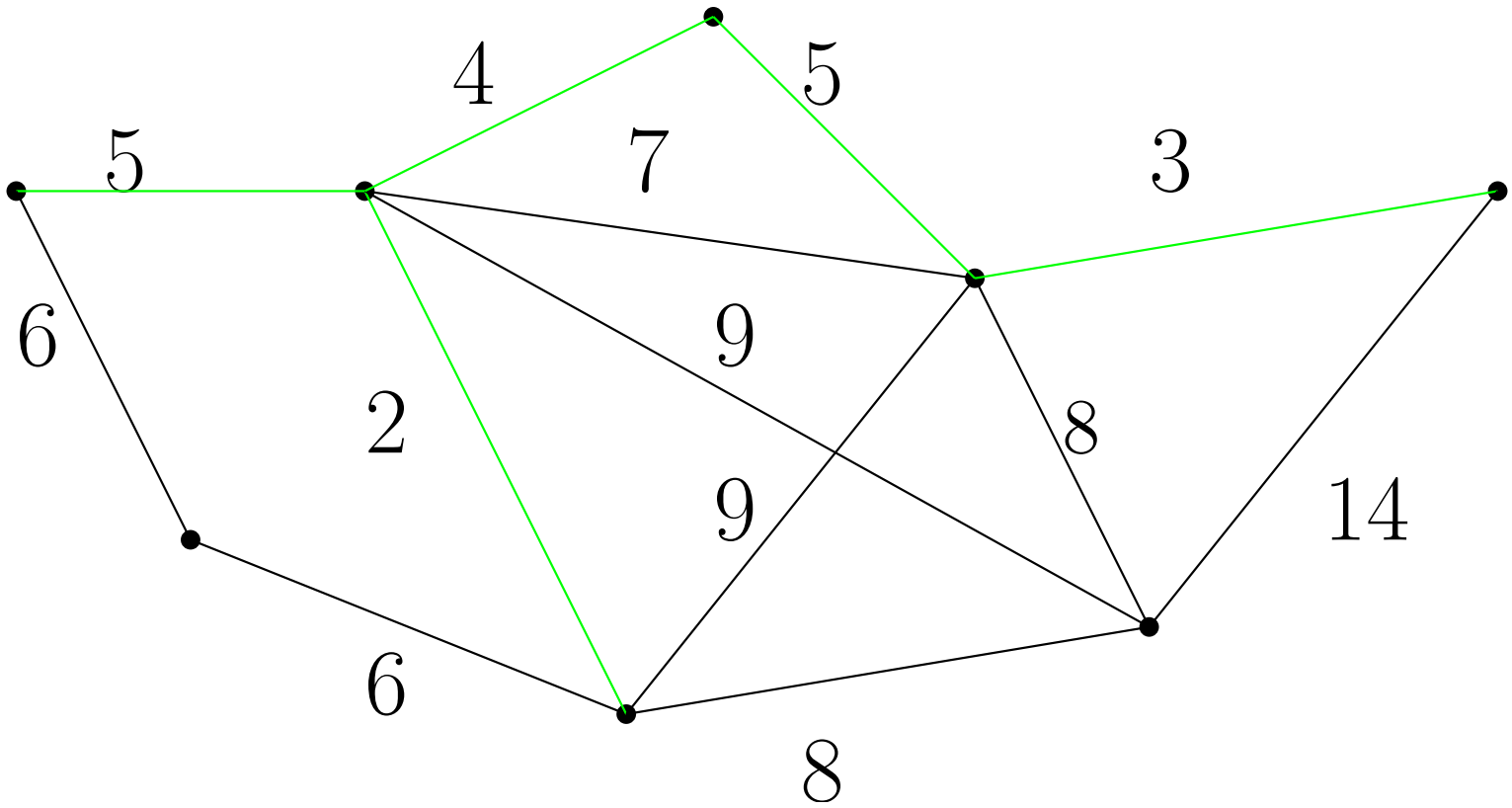
Lösung mit Kruskal



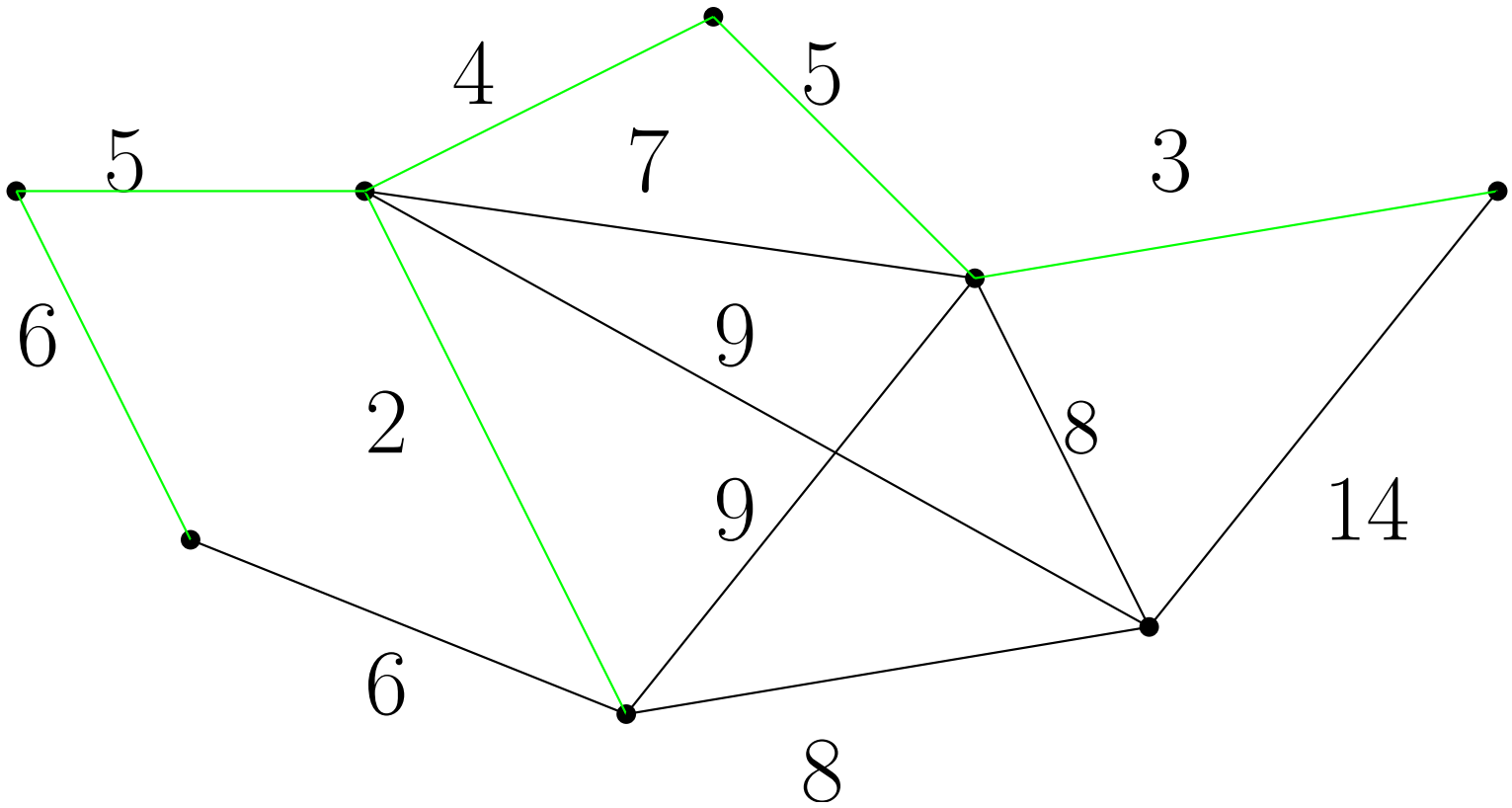
Lösung mit Kruskal



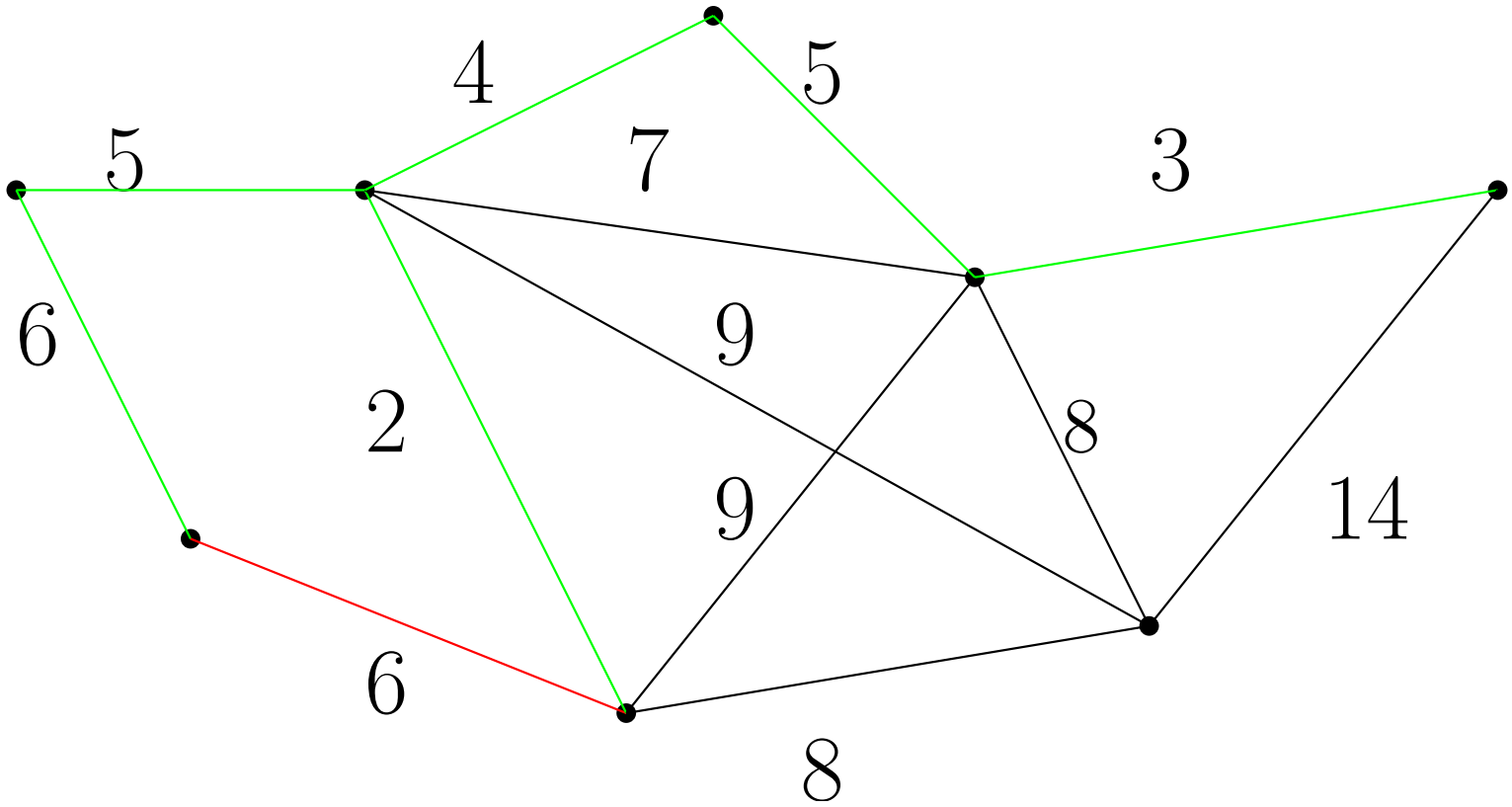
Lösung mit Kruskal



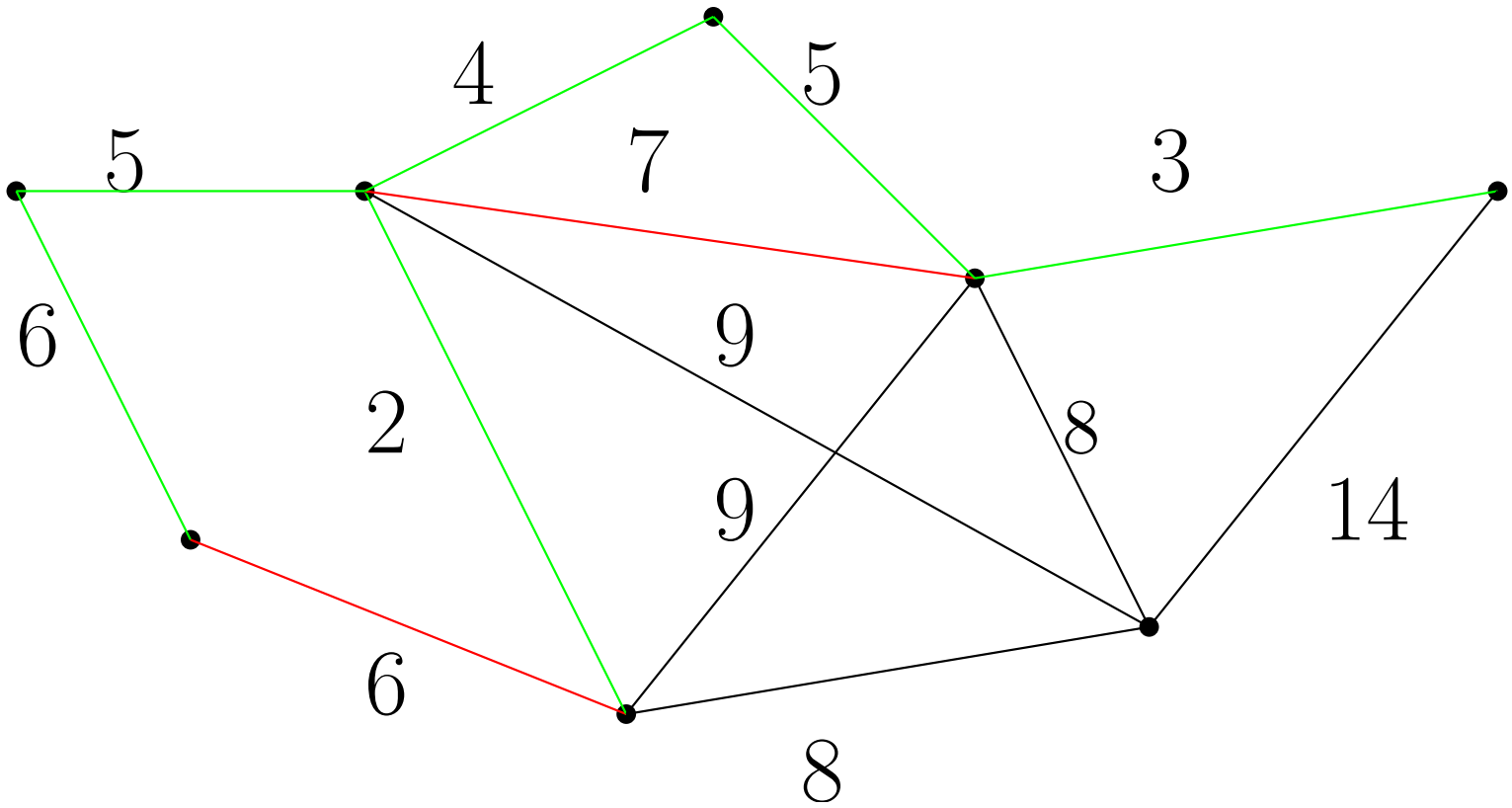
Lösung mit Kruskal



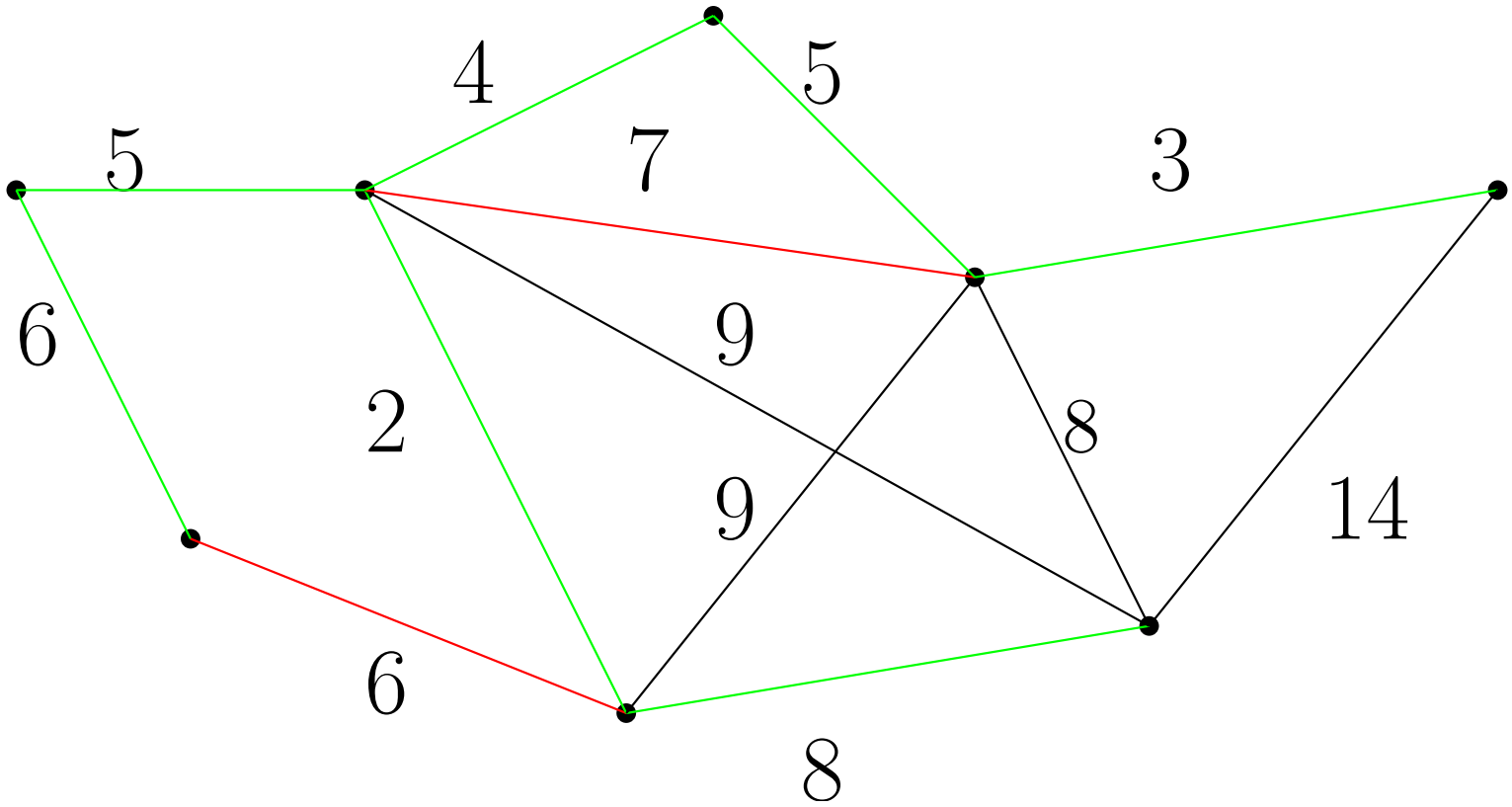
Lösung mit Kruskal



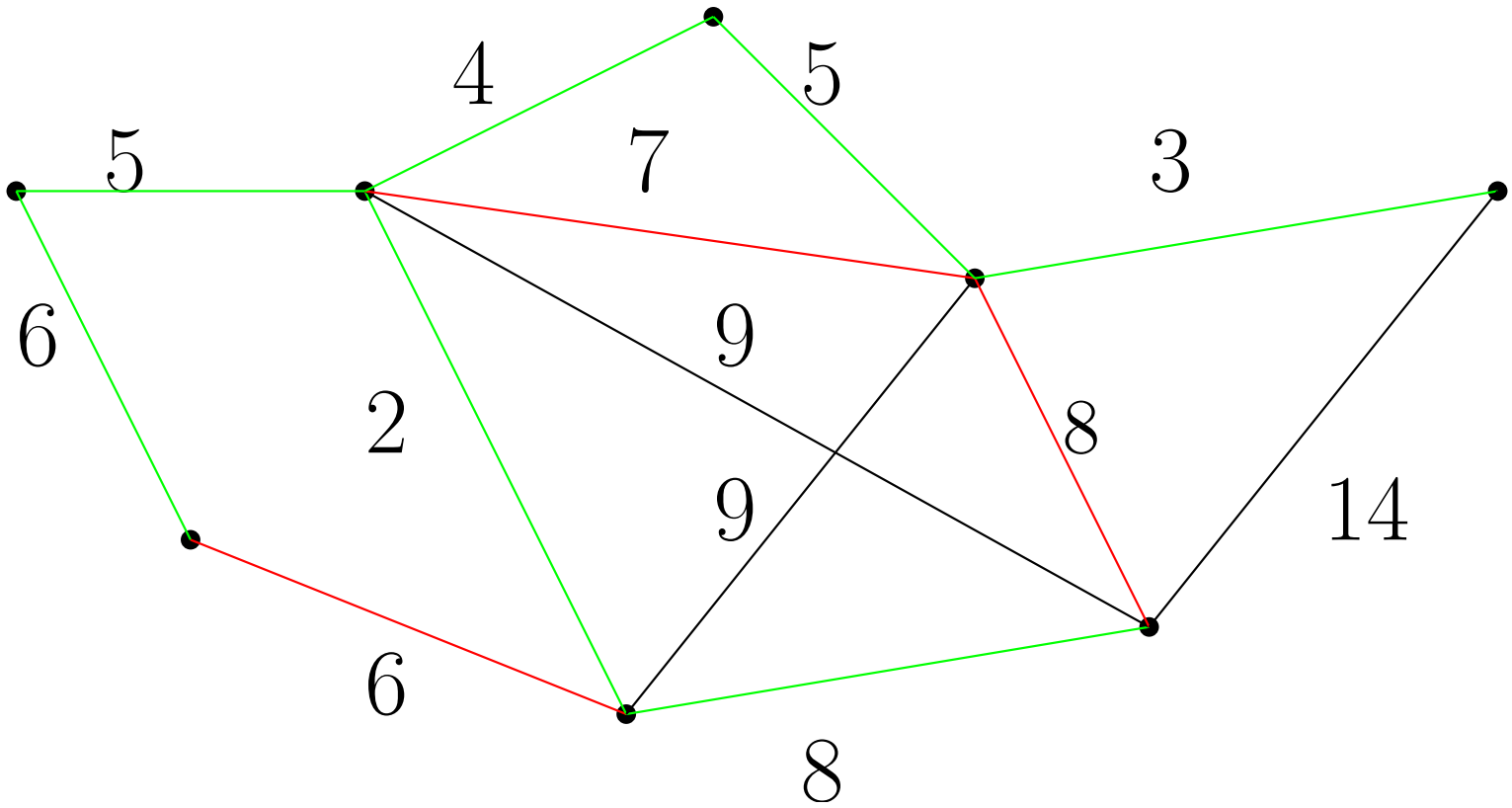
Lösung mit Kruskal



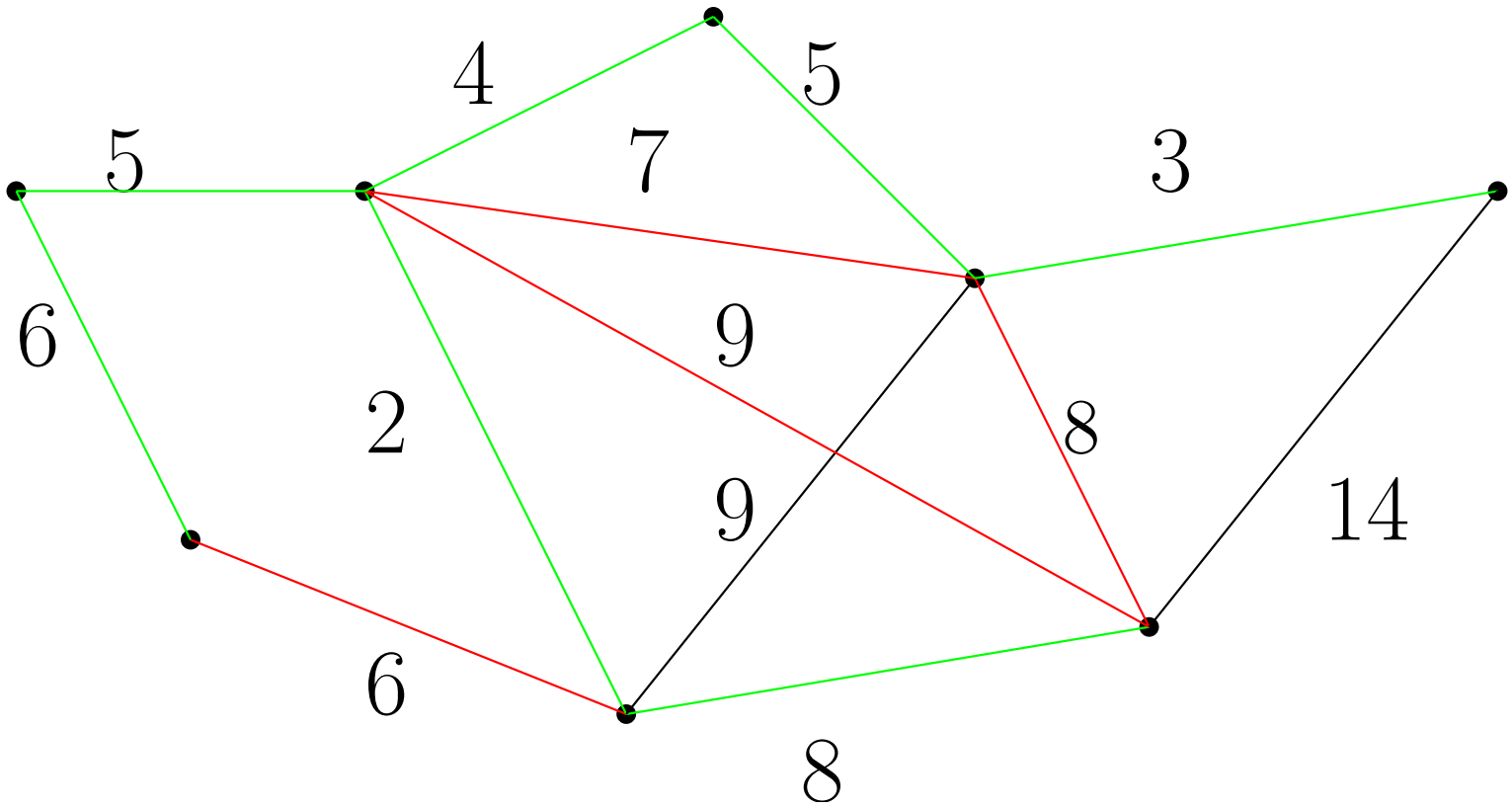
Lösung mit Kruskal



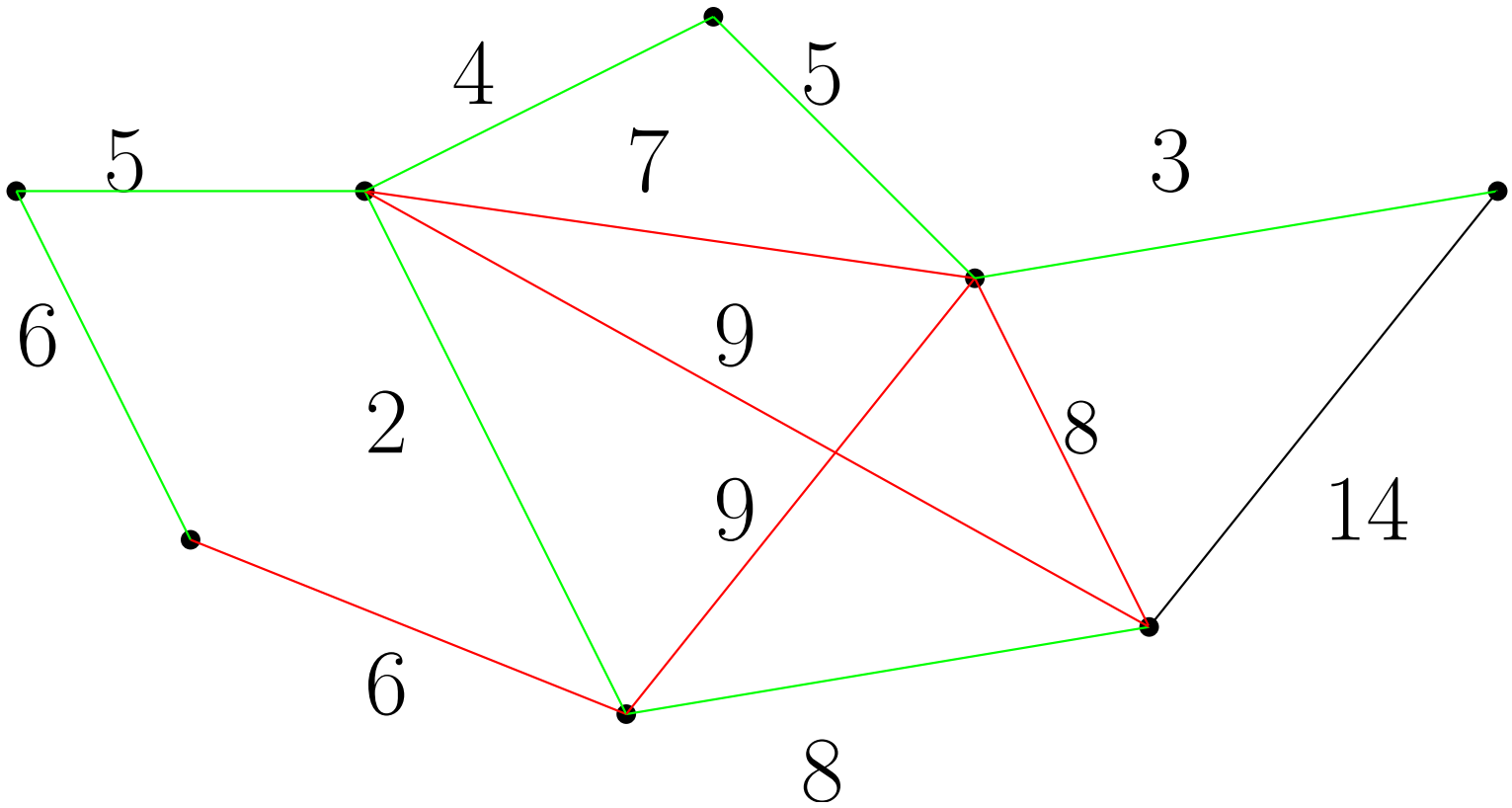
Lösung mit Kruskal



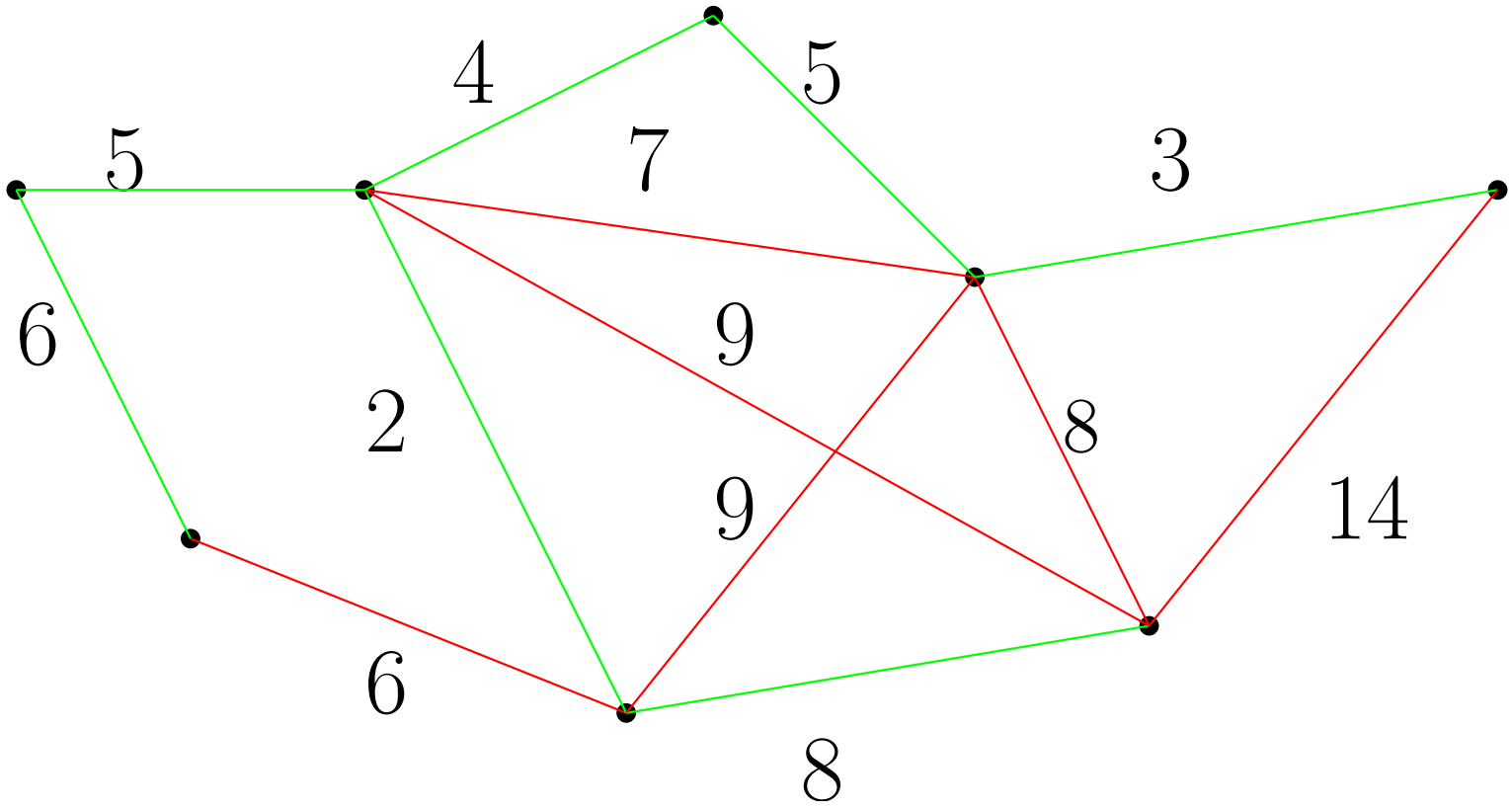
Lösung mit Kruskal



Lösung mit Kruskal



Lösung



Die Grün eingefärbten Kanten ergeben nun einen minimalen Spannbaum von G.

Laufzeit

Bestimmt durch Kanten

Die Laufzeit des Algorithmus wird zunächst durch das sortieren der Kanten beschränkt, das Überprüfen auf Kreisfreiheit geht i.A. schneller. Dementsprechend liegt die Laufzeit in $O(|E| * log(|E|))$.

Schneller mit vorsortierter Kantenliste

Bei bereits Vorsortierter Kantenliste und unter Verwendung der sog. Union-Find-Datenstruktur liegt die Laufzeit des Algorithmus in $O(|V| + |E| * \varphi^{-1}(|V|))$.
Dabei ist φ^{-1} die inverse Ackermannfunktion und für alle 'praktischen' Eingaben kleiner als 5.

Laufzeit

Bestimmt durch Kanten

Die Laufzeit des Algorithmus wird zunächst durch das sortieren der Kanten beschränkt, das Überprüfen auf Kreisfreiheit geht i.A. schneller. Dementsprechend liegt die Laufzeit in $O(|E| * log(|E|))$.

Schneller mit vorsortierter Kantenliste

Bei bereits Vorsortierter Kantenliste und unter Verwendung der sog. Union-Find-Datenstruktur liegt die Laufzeit des Algorithmus in $O(|V| + |E| * \varphi^{-1}(|V|))$.
Dabei ist φ^{-1} die inverse Ackermannfunktion und für alle 'praktischen' Eingaben kleiner als 5.