technische universität
dortmund

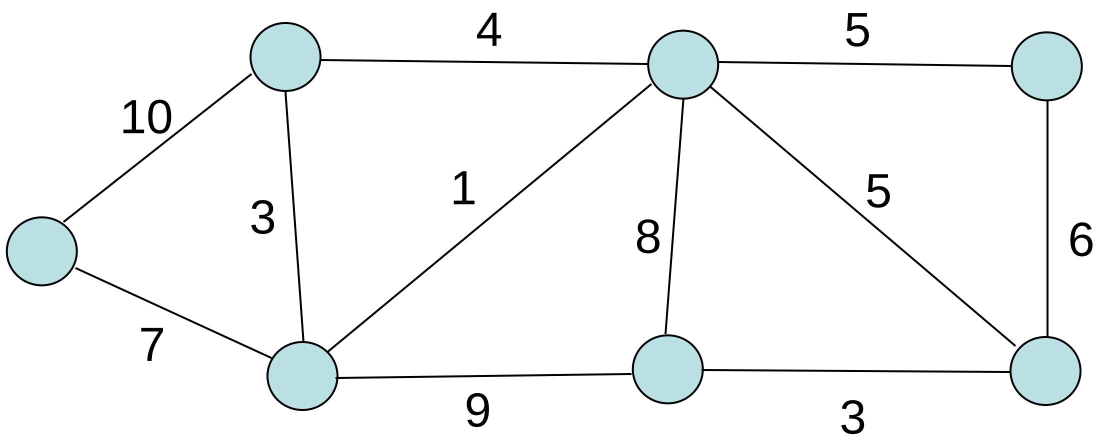
LS 2 /
Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Graphalgorithmen

Minimale Spannbäume

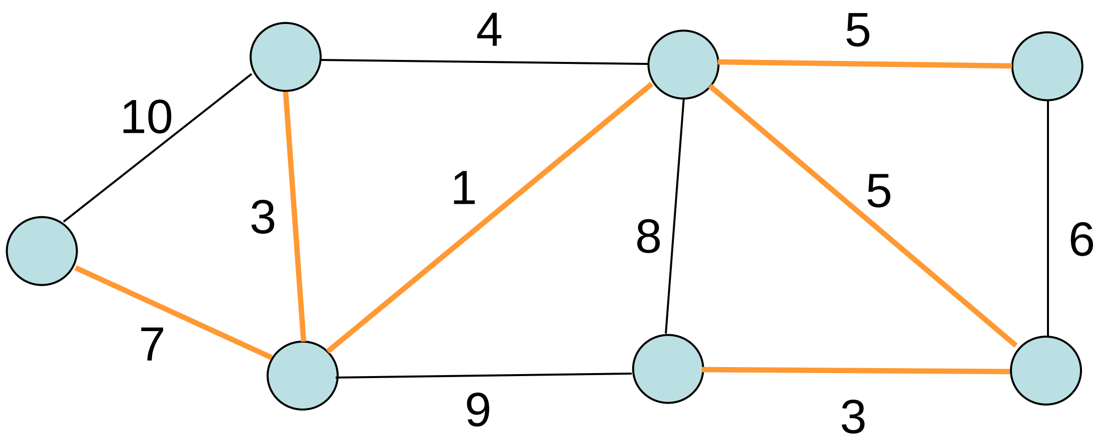
- Gegeben: Gewichteter, ungerichteter, zusammenhängender Graph $G = (V, E)$
- Gesucht: Ein aufspannender Baum mit minimalem Gewicht
  (Summe der Kantengewichte des Baums)
- Aufspannender Baum: Inklusionsmaximaler kreisfreier Teilgraph mit Knotenmenge V



Graphalgorithmen

Minimale Spannbäume

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



Graphalgorithmen

Berechnung von minimalen Spannbäumen

- Gieriger Algorithmus
- Invariante: Menge A von Kanten, die immer Untermenge eines minimalen Spannbaums ist
- Algorithmus: Finde Kante, die zu A hinzugefügt werden kann, ohne dass Invariante verletzt wird

Definition

Eine Kante, die zu A hinzugefügt werden kann, ohne die Invariante zu verletzen, heißt sicher.

Graphalgorithmen

GenerischerMSTAlgorithmus(G)

1. $A \leftarrow \emptyset$
2. **while** A ist kein minimaler Spannbaum **do**
3. finde Kante (u, v), die sicher für A ist
4. $A \leftarrow A \cup \{(u, v)\}$
5. **return** A

Graphalgorithmen

Wie findet man eine sichere Kante?

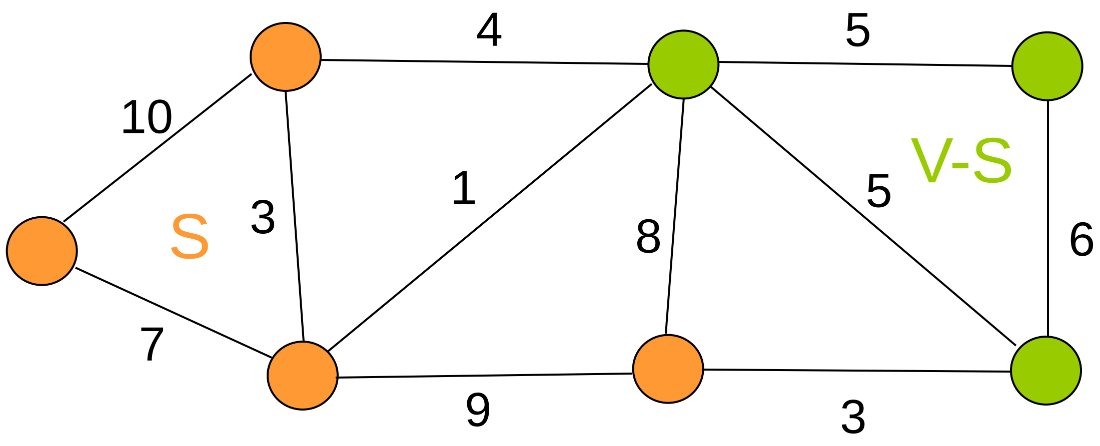
GenerischerMSTAlgorithmus(G)

1. $A \leftarrow \emptyset$
2. **while** A ist kein minimaler Spannbaum **do**
3. finde Kante (u, v), die sicher für A ist
4. $A \leftarrow A \cup \{(u, v)\}$
5. **return** A

Graphalgorithmen

Definition

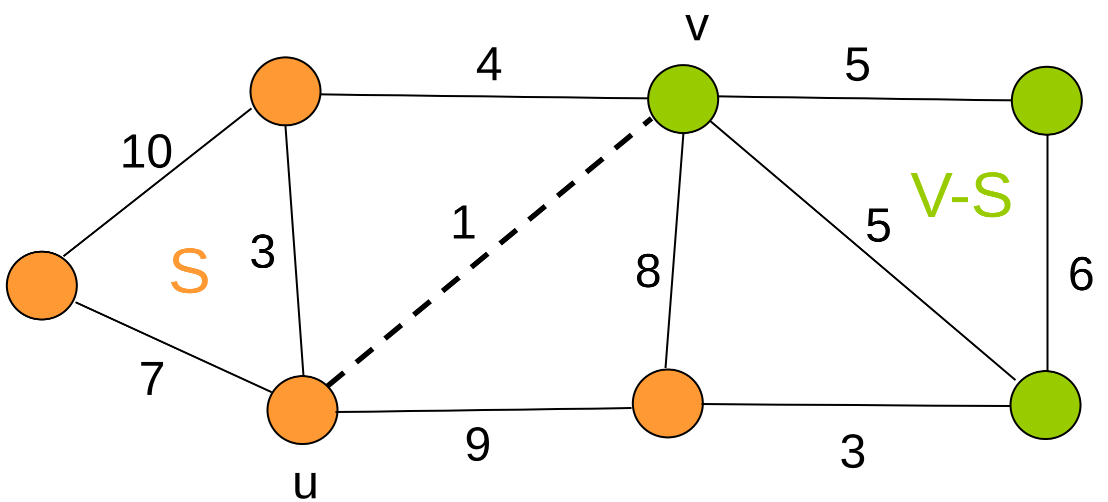
- Ein Schnitt $(S, V - S)$ in einem ungerichteten Graph $G = (V, E)$ ist eine Partition von V.
- Eine Kante $(u, v) \in E$ kreuzt den Schnitt $(S, V - S)$, wenn $u \in S$ und $v \in V - S$ liegt.



Graphalgorithmen

Definition

- Ein Schnitt $(S, V - S)$ in einem ungerichteten Graph $G = (V, E)$ ist eine Partition von V.
- Eine Kante $(u, v) \in E$ kreuzt den Schnitt $(S, V - S)$, wenn $u \in S$ und $v \in V - S$ liegt.

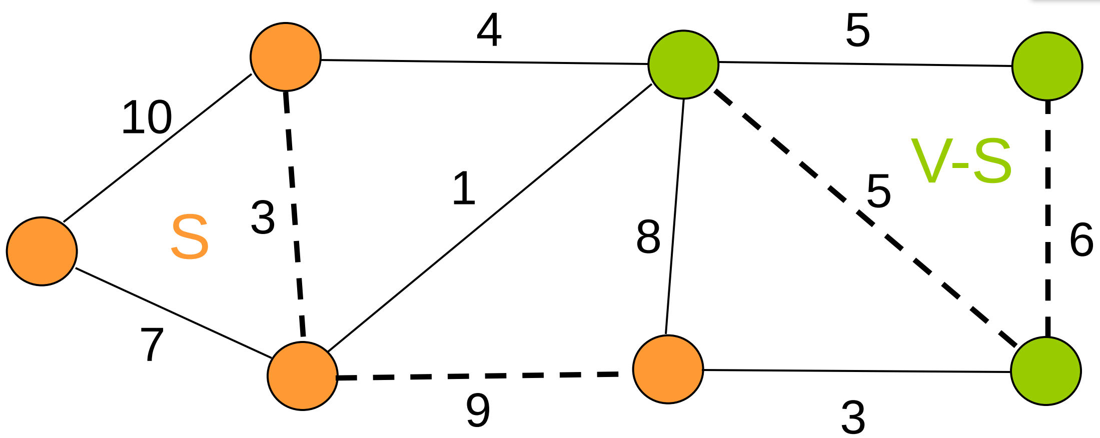


Graphalgorithmen

Definition

- Ein Schnitt respektiert eine Menge A von Kanten, wenn keine der Kanten den Schnitt kreuzt.

Beispiel:
Der Schnitt respektiert die gestrichelten Kanten.

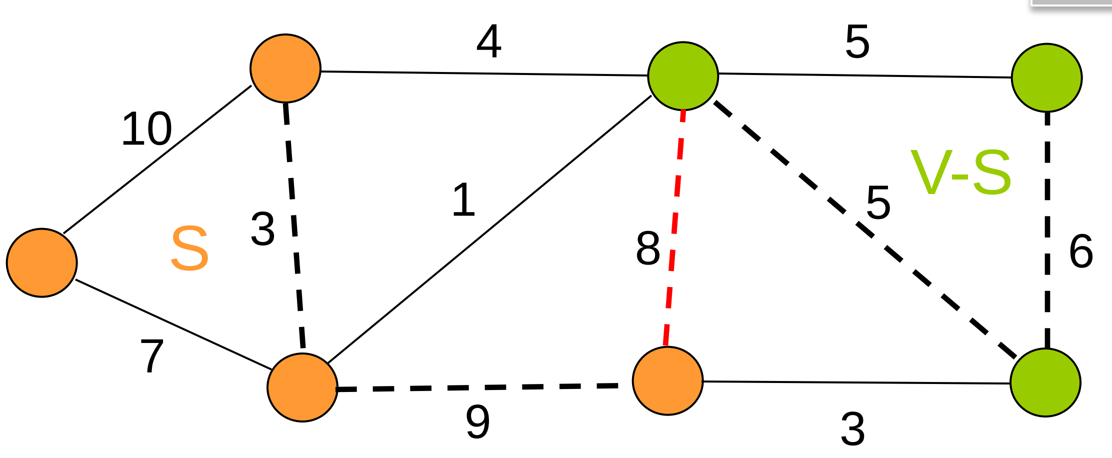


Graphalgorithmen

Definition

- Ein Schnitt respektiert eine Menge A von Kanten, wenn keine der Kanten den Schnitt kreuzt.

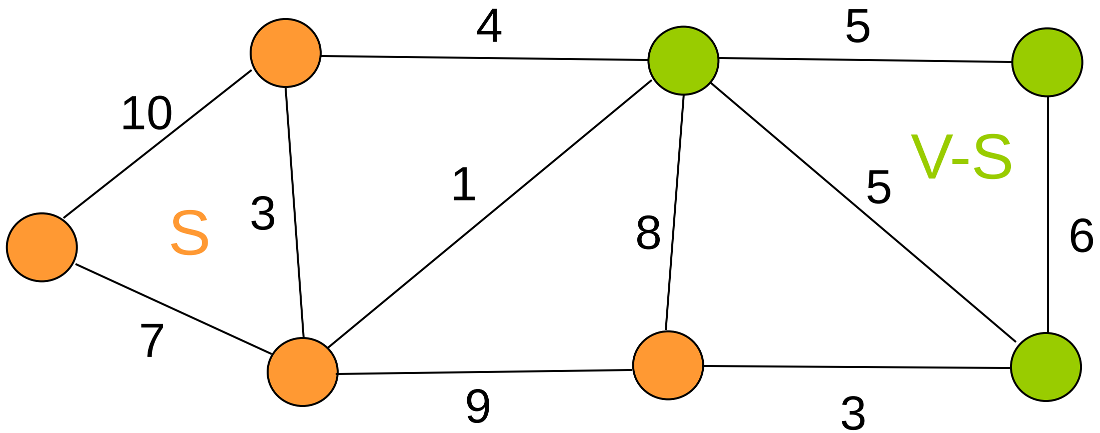
Zweites Beispiel:
Der Schnitt respektiert die gestrichelten Kanten nicht.



Graphalgorithmen

Definition

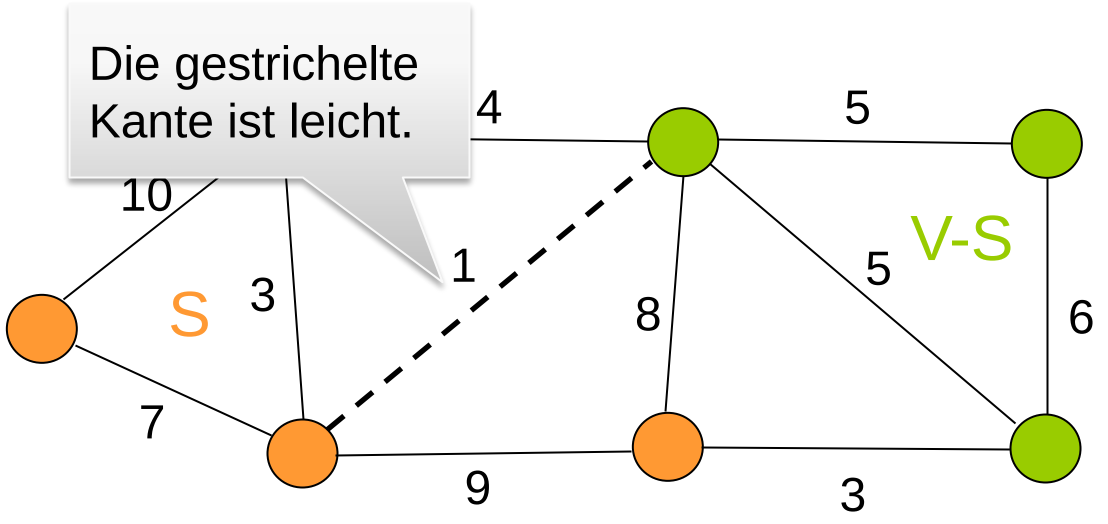
- Eine Kante, die einen Schnitt kreuzt, heißt leicht, wenn sie minimales Gewicht unter allen Kanten hat, die den Schnitt kreuzen.



Graphalgorithmen

Definition

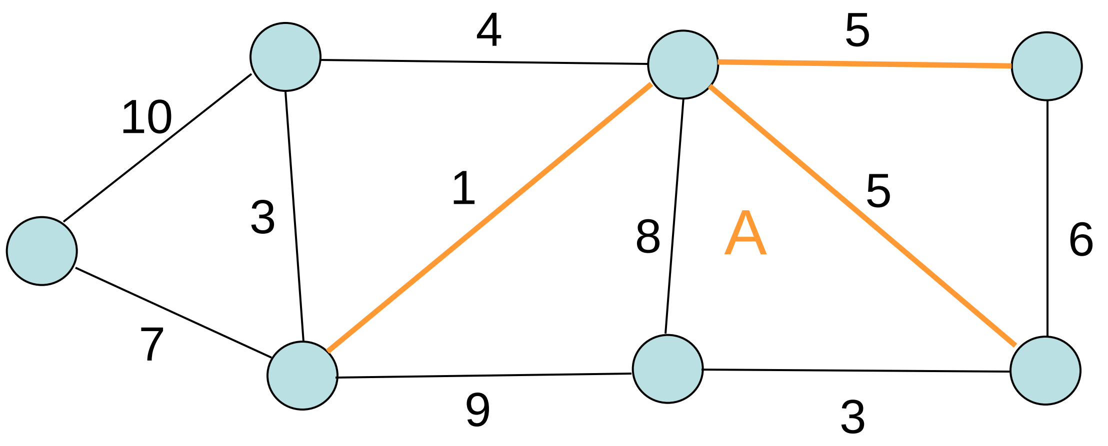
- Eine Kante, die einen Schnitt kreuzt, heißt leicht, wenn sie minimales Gewicht unter allen Kanten hat, die den Schnitt kreuzen.



Graphalgorithmen

Satz 69

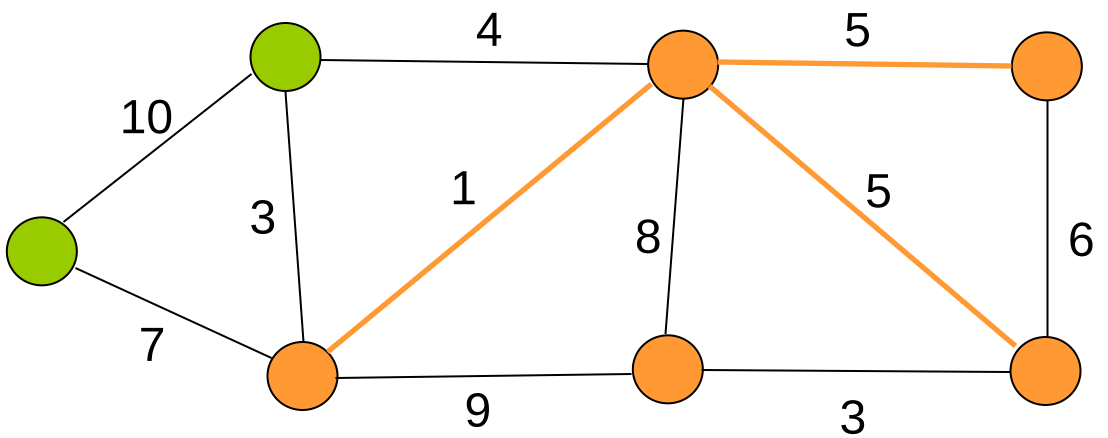
Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.



Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

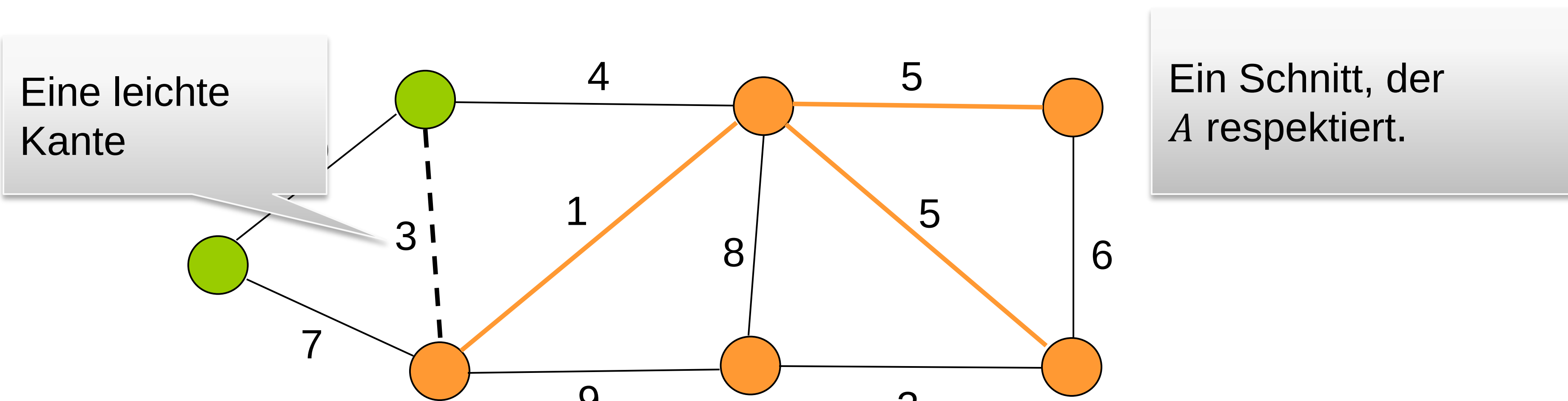


Ein Schnitt, der A respektiert.

Graphalgorithmen

Satz 69

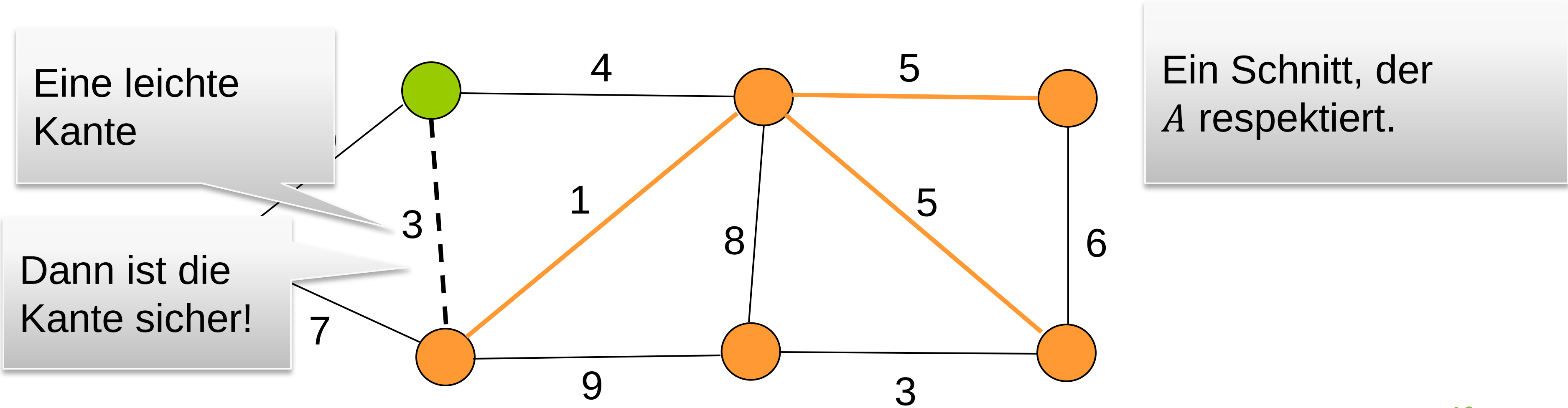
Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.



Graphalgorithmen

Satz 69

Sei $G=(V,E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V-S)$ ein Schnitt von G, der A respektiert und sei (u,v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u,v) sicher für A.



Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Für einen Baum T bezeichne $w(T) = \sum_{e \in T} w(e)$ sein Gewicht

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Für einen Baum T bezeichne $w(T) = \sum_{e \in T} w(e)$ sein Gewicht
- Sei T min. Spannbaum, der A enthält

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Für einen Baum T bezeichne $w(T) = \sum_{e \in T} w(e)$ sein Gewicht
- Sei T min. Spannbaum, der A enthält
- Annahme: Sei $(S, V - S)$ ein Schnitt wie im Satz und sei (u, v) eine leichte Kante, die den Schnitt kreuzt

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Für einen Baum T bezeichne $w(T) = \sum_{e \in T} w(e)$ sein Gewicht
- Sei T min. Spannbaum, der A enthält
- Annahme: Sei $(S, V - S)$ ein Schnitt wie im Satz und sei (u, v) eine leichte Kante, die den Schnitt kreuzt
- Wir konstruieren min. Spannbaum T‘, der A und (u, v) enthält

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Für einen Baum T bezeichne $w(T) = \sum_{e \in T} w(e)$ sein Gewicht
- Sei T min. Spannbaum, der A enthält
- Annahme: Sei $(S, V - S)$ ein Schnitt wie im Satz und sei (u, v) eine leichte Kante, die den Schnitt kreuzt
- Wir konstruieren min. Spannbaum T‘, der A und (u, v) enthält
- Wenn (u, v) in T ist, so sind wir fertig

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Für einen Baum T bezeichne $w(T) = \sum_{e \in T} w(e)$ sein Gewicht
- Sei T min. Spannbaum, der A enthält
- Annahme: Sei $(S, V - S)$ ein Schnitt wie im Satz und sei (u, v) eine leichte Kante, die den Schnitt kreuzt
- Wir konstruieren min. Spannbaum T‘, der A und (u, v) enthält
- Wenn (u, v) in T ist, so sind wir fertig

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Ansonsten: Kante (u, v) bildet Kreis mit Pfad p von u nach v in T

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Ansonsten: Kante (u, v) bildet Kreis mit Pfad p von u nach v in T
- Da u und v auf gegenüberliegenden Seiten des Schnitts $(S, V - S)$ liegen, gibt es mind. eine Kante aus p, die auch den Schnitt kreuzt

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Ansonsten: Kante (u, v) bildet Kreis mit Pfad p von u nach v in T
- Da u und v auf gegenüberliegenden Seiten des Schnitts $(S, V - S)$ liegen, gibt es mind. eine Kante aus p, die auch den Schnitt kreuzt
- Sei (x, y) eine solche Kante

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Ansonsten: Kante (u, v) bildet Kreis mit Pfad p von u nach v in T
- Da u und v auf gegenüberliegenden Seiten des Schnitts $(S, V - S)$ liegen, gibt es mind. eine Kante aus p, die auch den Schnitt kreuzt
- Sei (x, y) eine solche Kante
- (x, y) ist nicht in A, da der Schnitt A respektiert

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Ansonsten: Kante (u, v) bildet Kreis mit Pfad p von u nach v in T
- Da u und v auf gegenüberliegenden Seiten des Schnitts $(S, V - S)$ liegen, gibt es mind. eine Kante aus p, die auch den Schnitt kreuzt
- Sei (x, y) eine solche Kante
- (x, y) ist nicht in A, da der Schnitt A respektiert
- Da (x, y) auf dem eindeutig bestimmten Pfad von u nach v in T ist, wird T durch Entfernen von (x, y) in zwei Komponenten aufgeteilt.

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Ansonsten: Kante (u, v) bildet Kreis mit Pfad p von u nach v in T
- Da u und v auf gegenüberliegenden Seiten des Schnitts $(S, V - S)$ liegen, gibt es mind. eine Kante aus p, die auch den Schnitt kreuzt
- Sei (x, y) eine solche Kante
- (x, y) ist nicht in A, da der Schnitt A respektiert
- Da (x, y) auf dem eindeutig bestimmten Pfad von u nach v in T ist, wird T durch Entfernen von (x, y) in zwei Komponenten aufgeteilt.

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Hinzunahme von (u, v) verbindet diese Komponenten wieder (da Pfad p und (u, v) einen Kreis bilden)

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Hinzunahme von (u, v) verbindet diese Komponenten wieder (da Pfad p und (u, v) einen Kreis bilden)
- Definiere: $T' = T - \{(x, y)\} \cup \{(u, v)\}$

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Hinzunahme von (u, v) verbindet diese Komponenten wieder
  (da Pfad p und (u, v) einen Kreis bilden)
- Definiere: $T' = T - \{(x, y)\} \cup \{(u, v)\}$
- Wir zeigen, dass T‘ min. Spannbaum ist

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Hinzunahme von (u, v) verbindet diese Komponenten wieder (da Pfad p und (u, v) einen Kreis bilden)
- Definiere: $T' = T - \{(x, y)\} \cup \{(u, v)\}$
- Wir zeigen, dass T‘ min. Spannbaum ist
- T‘ ist ein Spannbaum, da T‘ nach Konstruktion kreisfrei ist und T genauso viele Kanten wie der Spannbaum T hat

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Hinzunahme von (u, v) verbindet diese Komponenten wieder (da Pfad p und (u, v) einen Kreis bilden)
- Definiere: $T' = T - \{(x, y)\} \cup \{(u, v)\}$
- Wir zeigen, dass T‘ min. Spannbaum ist
- T‘ ist ein Spannbaum, da T‘ nach Konstruktion kreisfrei ist und T genauso viele Kanten wie der Spannbaum T hat

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Da (u, v) leichte Kante ist, die $(S, V - S)$ kreuzt und (x, y) ebenfalls $(S, V - S)$ kreuzt, gilt $w(u, v) \leq w(x, y)$. Daher

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Da (u, v) leichte Kante ist, die $(S, V - S)$ kreuzt und (x, y) ebenfalls $(S, V - S)$ kreuzt, gilt $w(u, v) \leq w(x, y)$. Daher
- $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Da (u, v) leichte Kante ist, die $(S, V - S)$ kreuzt und (x, y) ebenfalls $(S, V - S)$ kreuzt, gilt $w(u, v) \leq w(x, y)$. Daher
- $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$
- T ist aber min. Spannbaum und somit gilt $w(T) \leq w(T')$

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Da (u, v) leichte Kante ist, die $(S, V - S)$ kreuzt und (x, y) ebenfalls $(S, V - S)$ kreuzt, gilt $w(u, v) \leq w(x, y)$. Daher
- $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$
- T ist aber min. Spannbaum und somit gilt $w(T) \leq w(T')$
- Daher muss T' ebenfalls min. Spannbaum sein

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Da (u, v) leichte Kante ist, die $(S, V - S)$ kreuzt und (x, y) ebenfalls $(S, V - S)$ kreuzt, gilt $w(u, v) \leq w(x, y)$. Daher
- $w(T‘) = w(T) - w(x, y) + w(u, v) \leq w(T)$
- T ist aber min. Spannbaum und somit gilt $w(T) \leq w(T‘)$
- Daher muss $T‘$ ebenfalls min. Spannbaum sein

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Es bleibt zu zeigen, dass (u, v) sicher für A ist.

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Es bleibt zu zeigen, dass (u, v) sicher für A ist.
- Dies folgt direkt aus $A \subseteq T'$, da $A \subseteq T$, $(x, y) \notin A$ (weil (x, y) den Schnitt kreuzt und der Schnitt A respektiert) und $(u, v) \in T$‘ und weil T‘ min. Spannbaum ist

Graphalgorithmen

Satz 69

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter und gewichteter Graph. Sei $A \subseteq E$ Teilmenge eines minimalen Spannbaums von G ist. Sei $(S, V - S)$ ein Schnitt von G, der A respektiert und sei (u, v) eine leichte Kanten, die diesen Schnitt kreuzt. Dann ist (u, v) sicher für A.

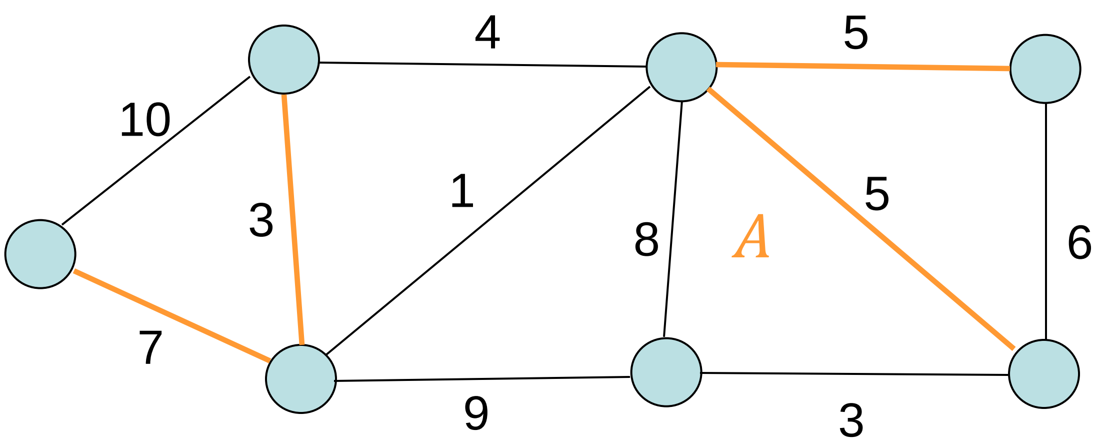
Beweis

- Es bleibt zu zeigen, dass (u, v) sicher für A ist.
- Dies folgt direkt aus $A \subseteq T'$, da $A \subseteq T$, $(x, y) \notin A$ (weil (x, y) den Schnitt kreuzt und der Schnitt A respektiert) und $(u, v) \in T'$ und weil T' min. Spannbaum ist

Graphalgorithmen

Korollar 70

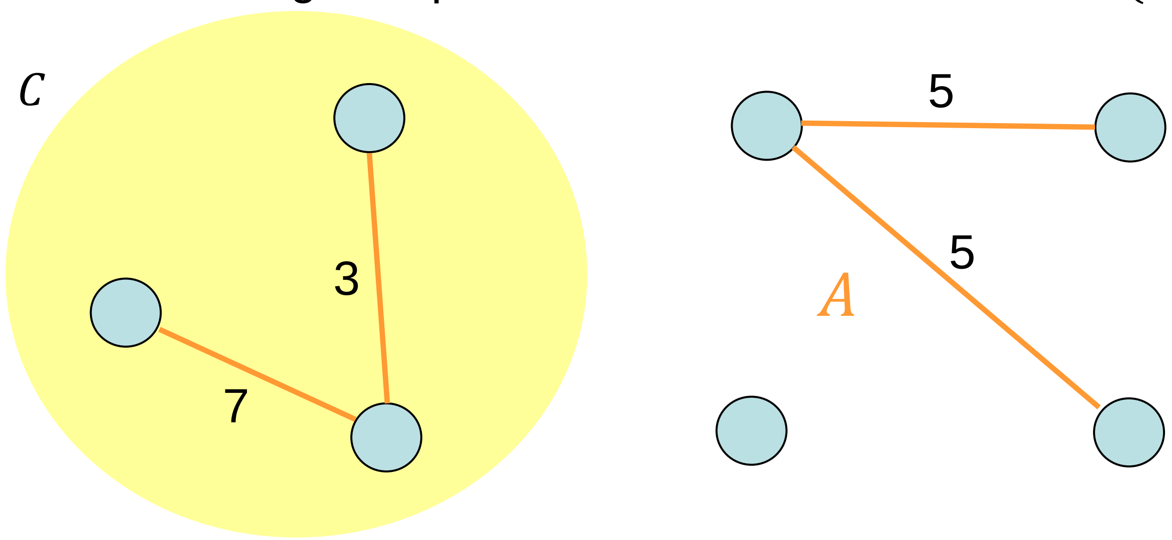
Sei $G = (V, E)$ ein zusammenhängender, ungerichteter, gewichteter Graph. Sei A eine Teilmenge von E, die in einem minimalen Spannbaum von G enthalten ist und sei C eine Zusammenhangskomponente (Baum) im Wald $H = (V, A)$. Wenn (u, v) eine leichte Kante ist, die C mit einer anderen Zusammenhangskomponente in H verbindet, dann ist (u, v) sicher für A.



Graphalgorithmen

Korollar 70

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter, gewichteter Graph. Sei A eine Teilmenge von E, die in einem minimalen Spannbaum von G enthalten ist und sei C eine Zusammenhangskomponente (Baum) im Wald $H = (V, A)$. Wenn (u, v) eine leichte Kante ist, die C mit einer anderen Zusammenhangskomponente in H verbindet, dann ist (u, v) sicher für A.

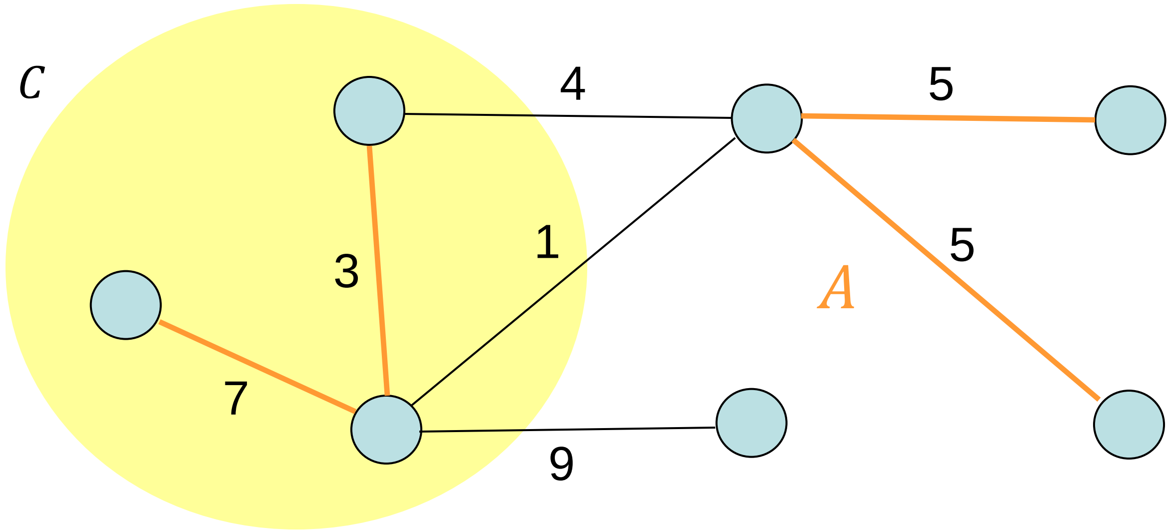


Wald H und Komponente C.

Graphalgorithmen

Korollar 70

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter, gewichteter Graph. Sei A eine Teilmenge von E, die in einem minimalen Spannbaum von G enthalten ist und sei C eine Zusammenhangskomponente (Baum) im Wald $H = (V, A)$. Wenn (u, v) eine leichte Kante ist, die C mit einer anderen Zusammenhangskomponente in H verbindet, dann ist (u, v) sicher für A.

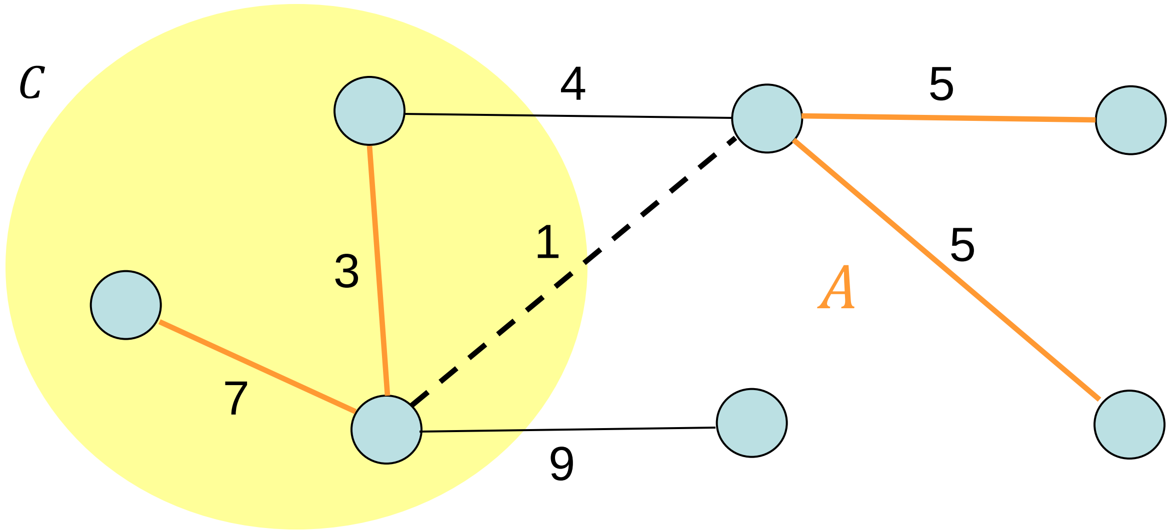


Kanten, die C verbinden.

Graphalgorithmen

Korollar 70

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter, gewichteter Graph. Sei A eine Teilmenge von E, die in einem minimalen Spannbaum von G enthalten ist und sei C eine Zusammenhangskomponente (Baum) im Wald $H = (V, A)$. Wenn (u, v) eine leichte Kante ist, die C mit einer anderen Zusammenhangskomponente in H verbindet, dann ist (u, v) sicher für A.



Leichte Kante ist sicher.

Graphalgorithmen

Korollar 70

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter, gewichteter Graph. Sei A eine Teilmenge von E, die in einem minimalen Spannbaum von G enthalten ist und sei C eine Zusammenhangskomponente (Baum) im Wald $H = (V, A)$. Wenn (u, v) eine leichte Kante ist, die C mit einer anderen Zusammenhangskomponente in H verbindet, dann ist (u, v) sicher für A.

Beweis

Der Schnitt $(C, V - C)$ respektiert A und (u, v) ist leichte Kante für diesen Schnitt. Damit folgt das Korollar aus dem vorherigen Satz.

Graphalgorithmen

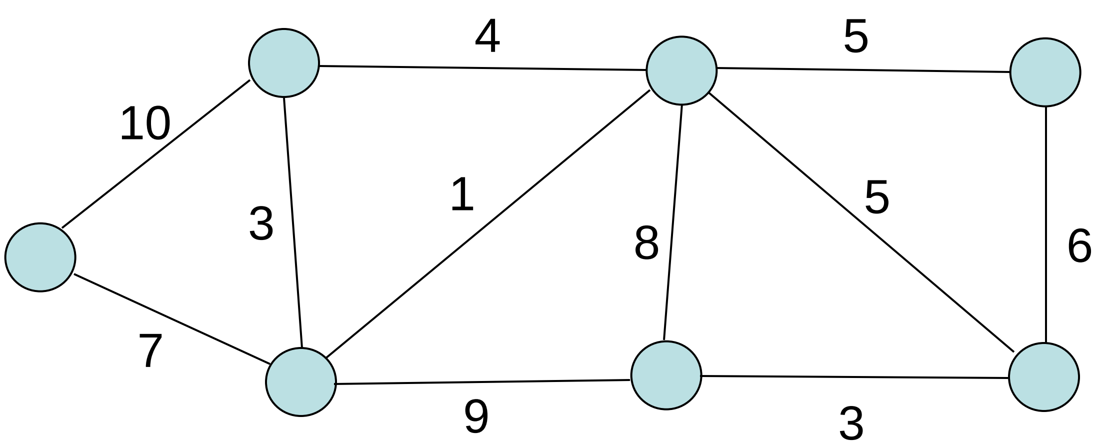
Idee des Algorithmus von Kruskal

- Verwende generischen Algorithmus
- Nimm immer die Kante mit geringstem Gewicht, die zwei Bäume im aktuellen aufspannenden Wald verbindet, und füge diese zu A hinzu
- Seien C und D diese beiden Bäume
- Die Kante ist eine leichte Kante, die C mit einem anderen Baum verbindet
- Damit ist sie sicher für A

Graphalgorithmen

Kruskal(G)

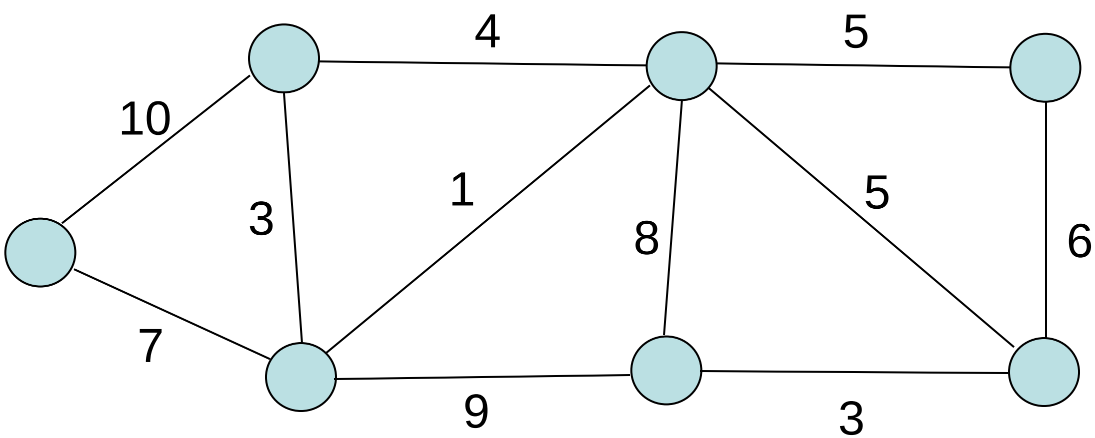
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

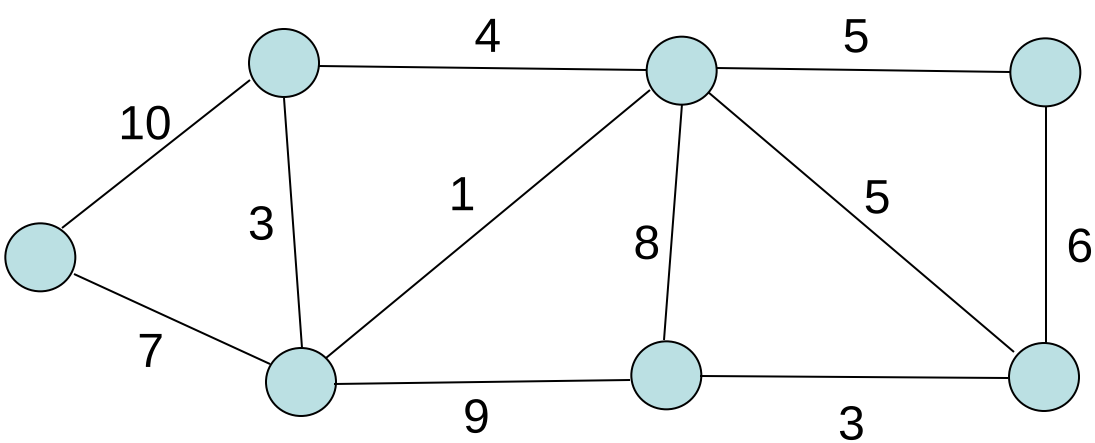
1. $\boxed{A \leftarrow \emptyset}$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

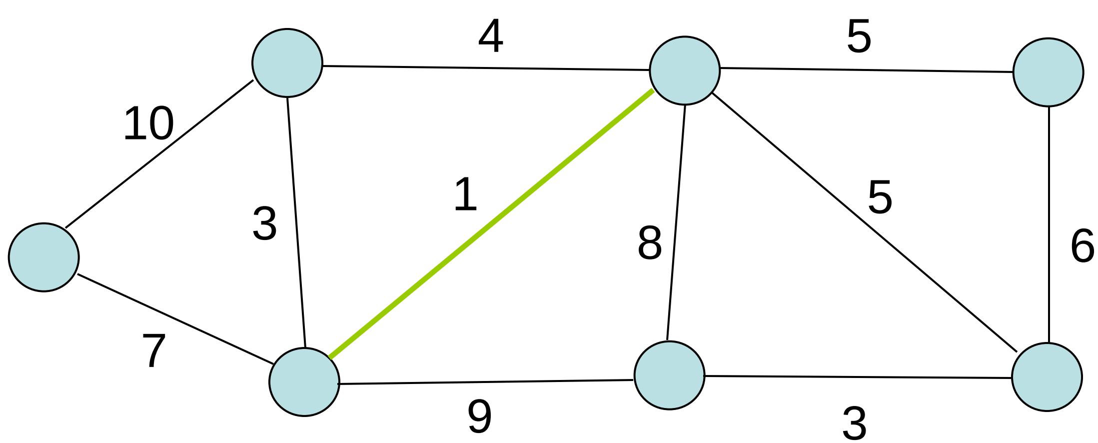
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangskomponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

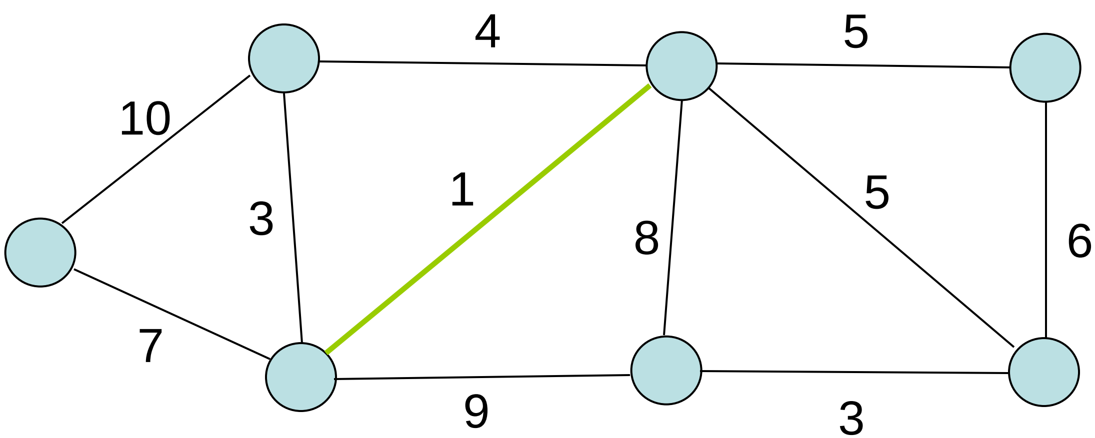
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs- komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

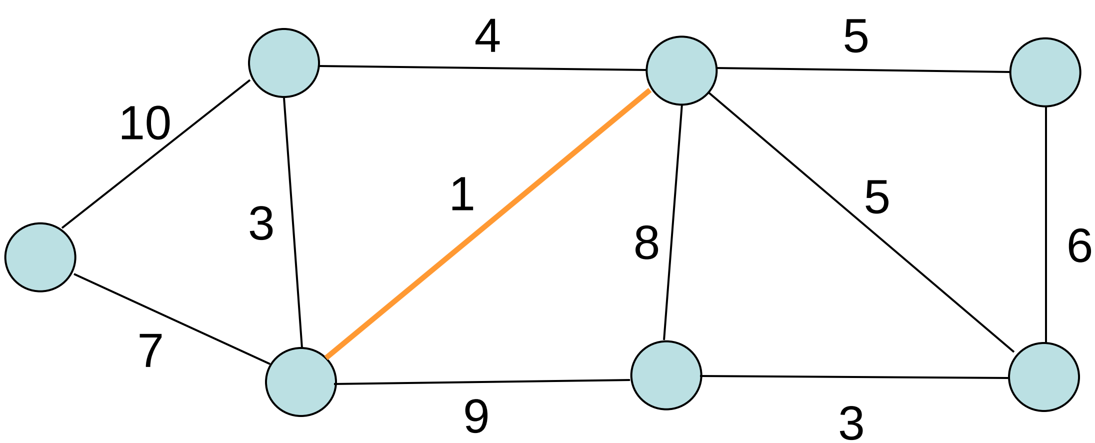
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

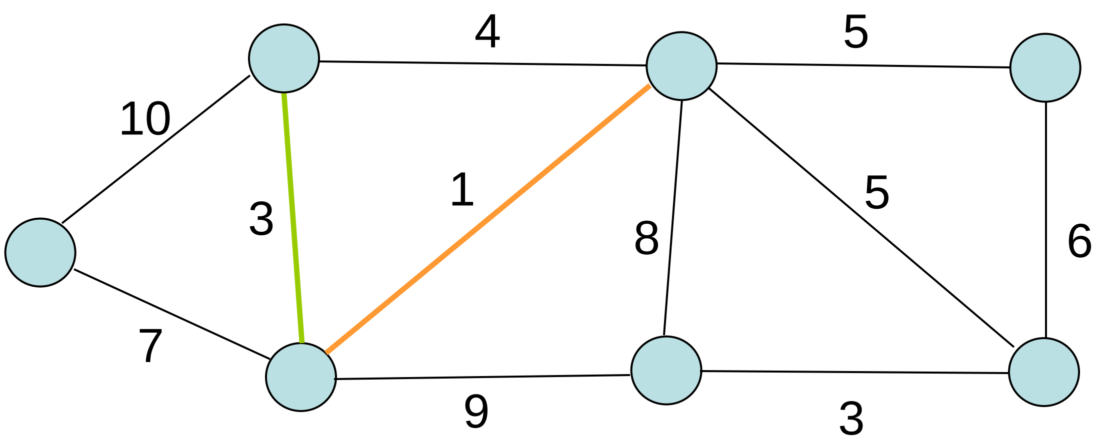
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $\boxed{A \leftarrow A \cup \{(u, v)\}}$
6. **return** A



Graphalgorithmen

Kruskal(G)

1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

$A \leftarrow \emptyset$

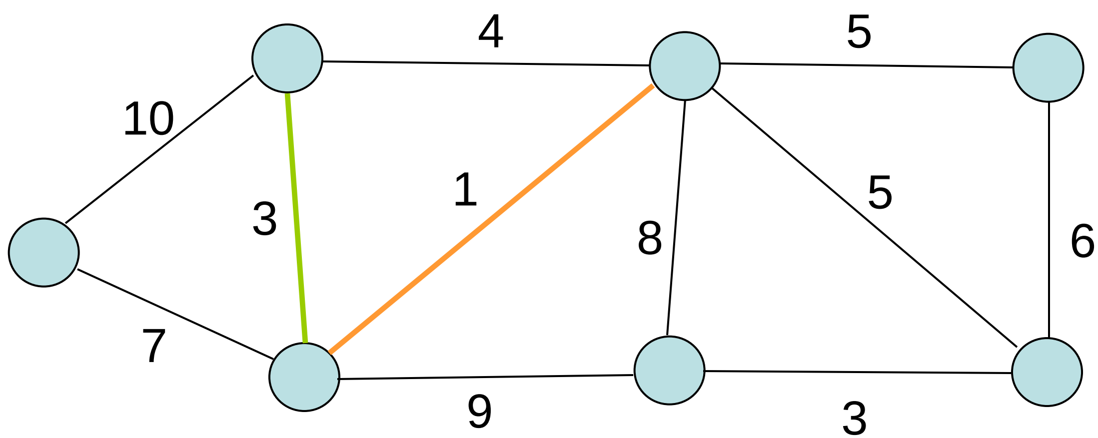
Sortiere Kanten nach Gewicht

for each $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**

if u und v sind nicht in der selben Zusammenhangs-
komponente in Graph $H = (V, A)$ **then**

$A \leftarrow A \cup \{(u, v)\}$

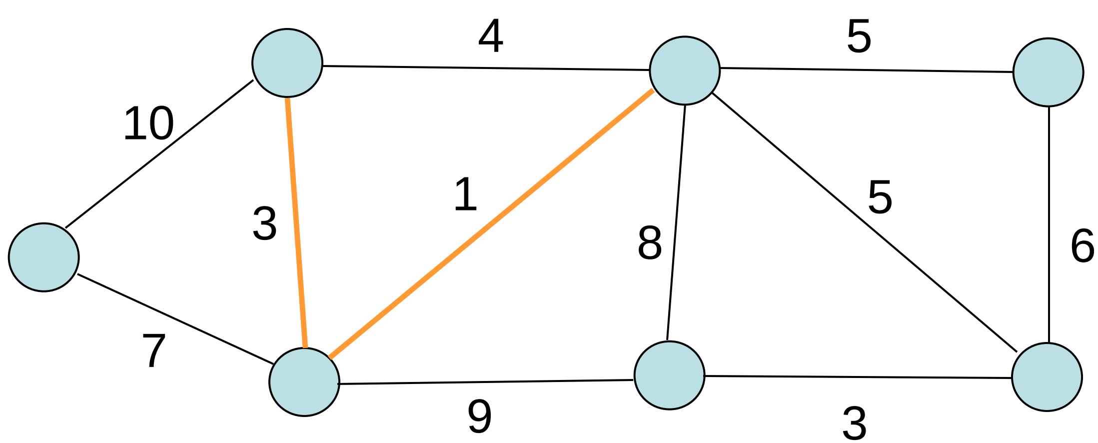
return A



Graphalgorithmen

Kruskal(G)

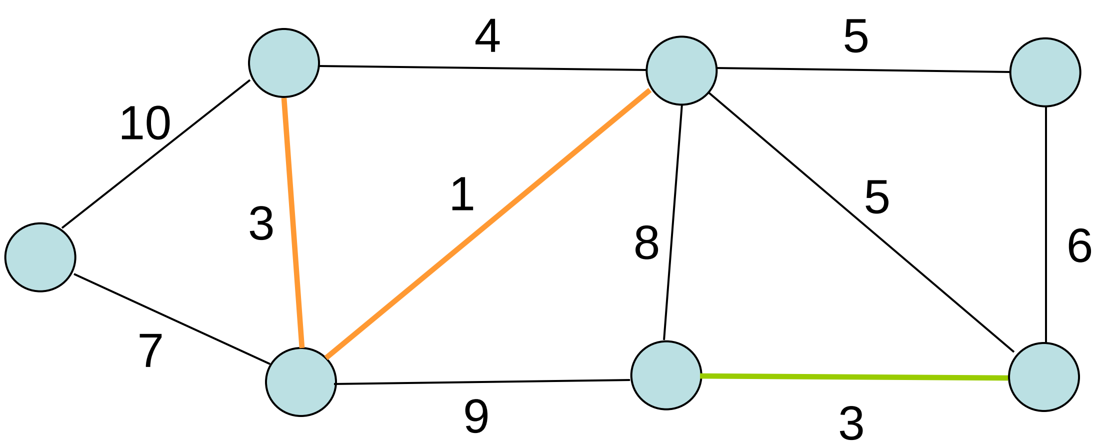
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $\boxed{A \leftarrow A \cup \{(u, v)\}}$
6. **return** A



Graphalgorithmen

Kruskal(G)

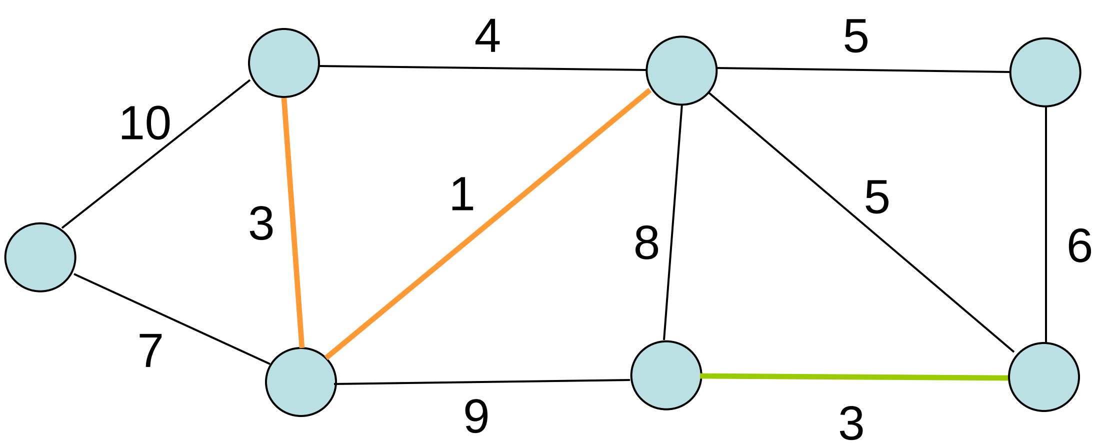
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

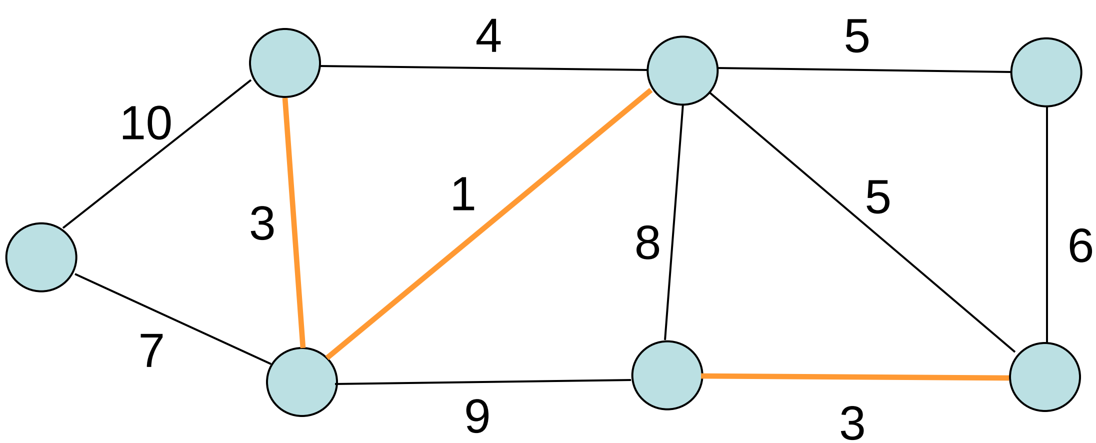
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

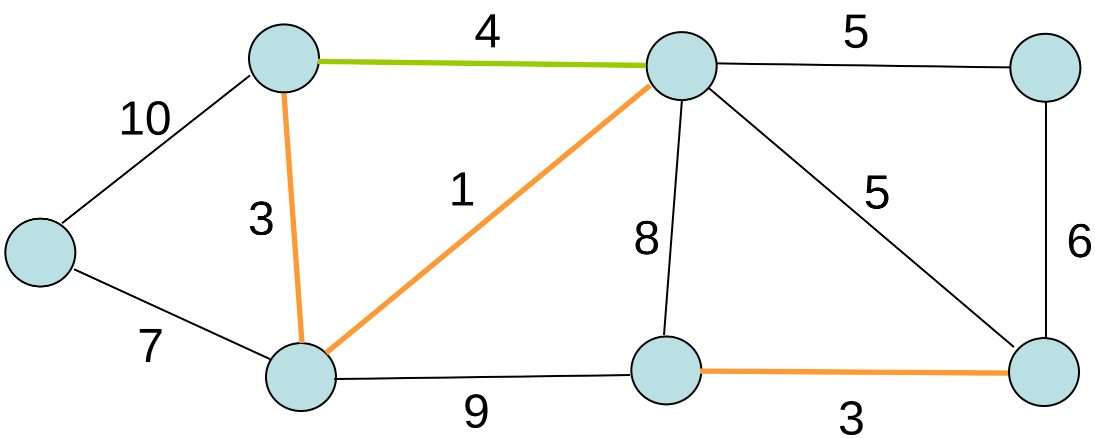
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $\boxed{A \leftarrow A \cup \{(u, v)\}}$
6. **return** A



Graphalgorithmen

Kruskal(G)

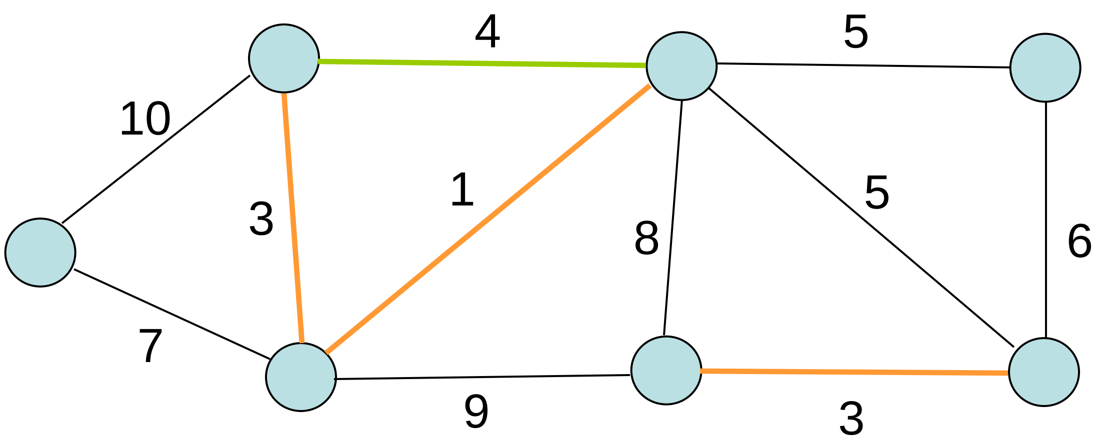
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-
   komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

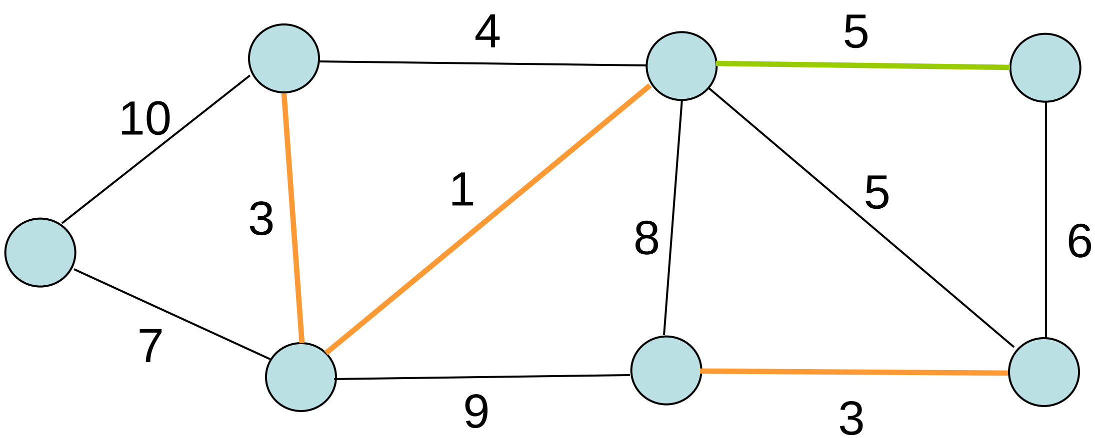
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

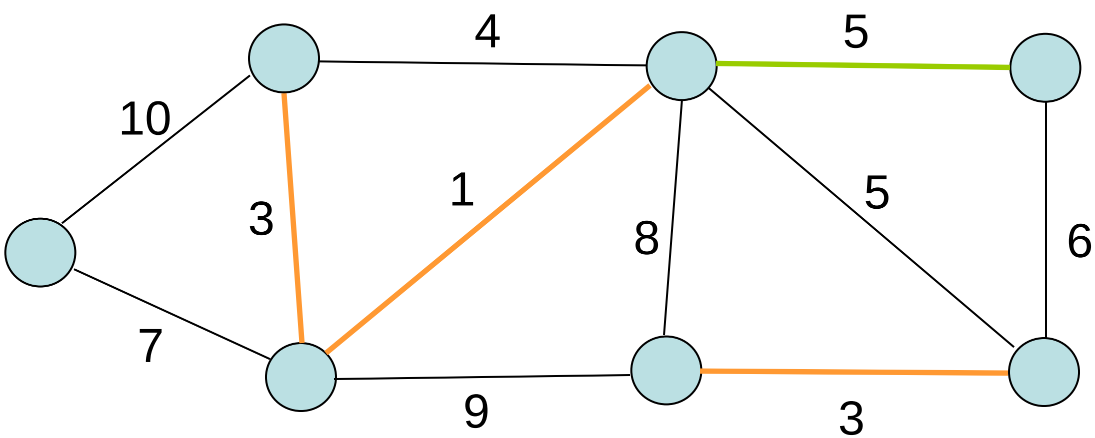
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

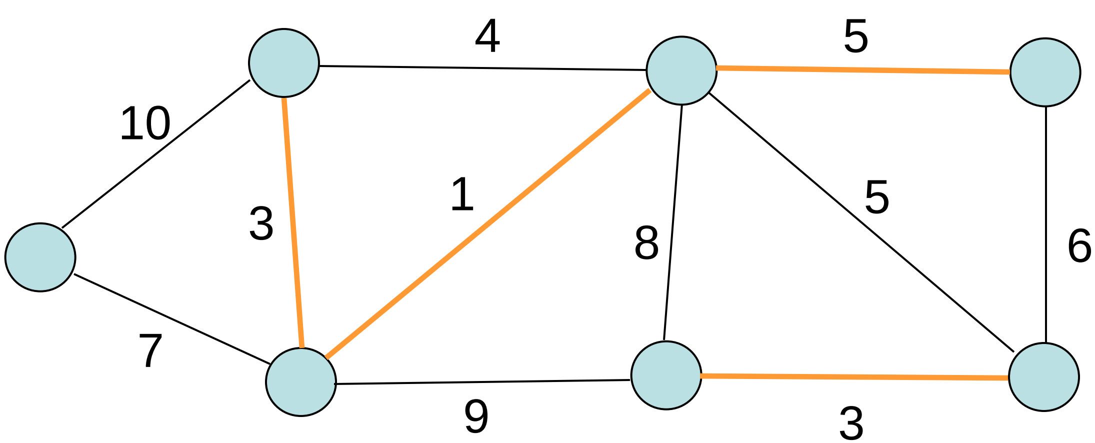
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

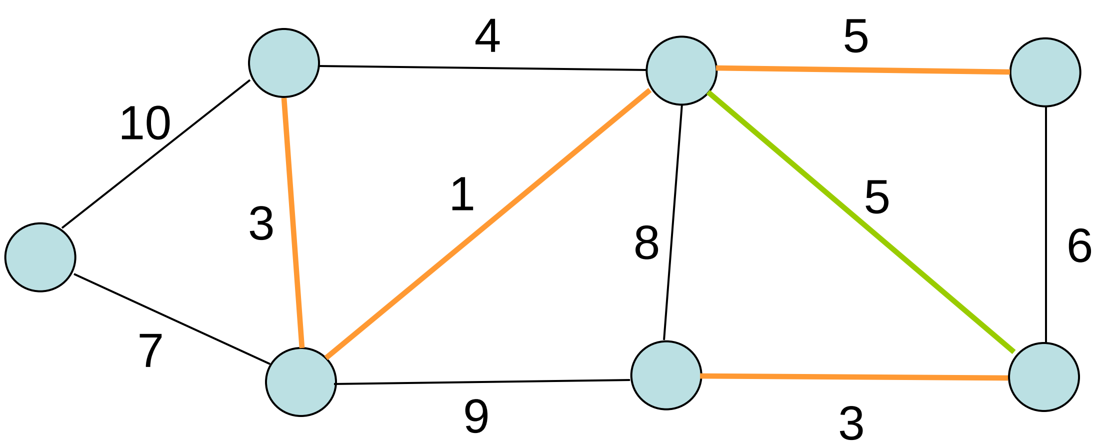
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

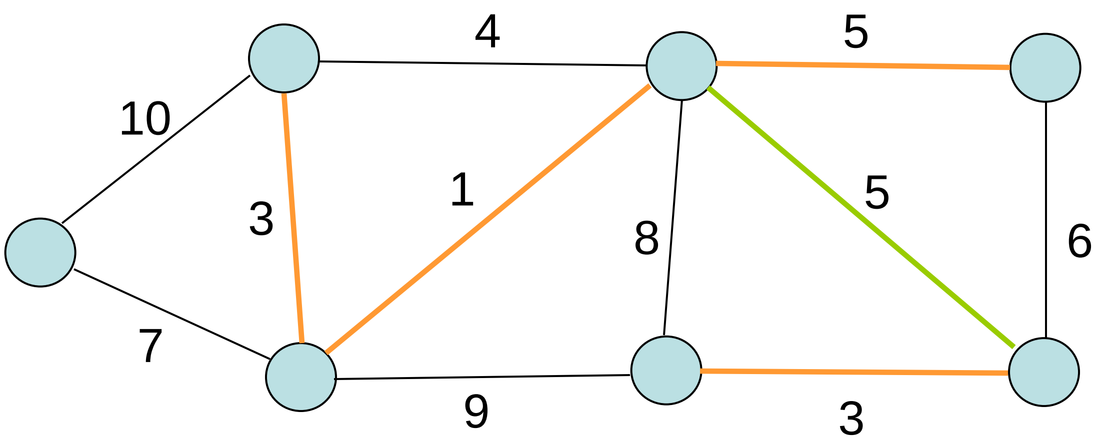
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

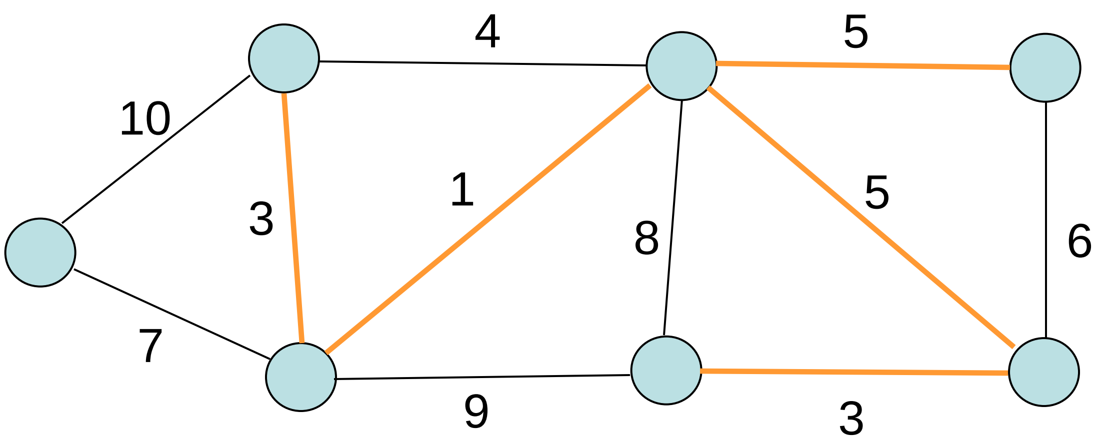
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

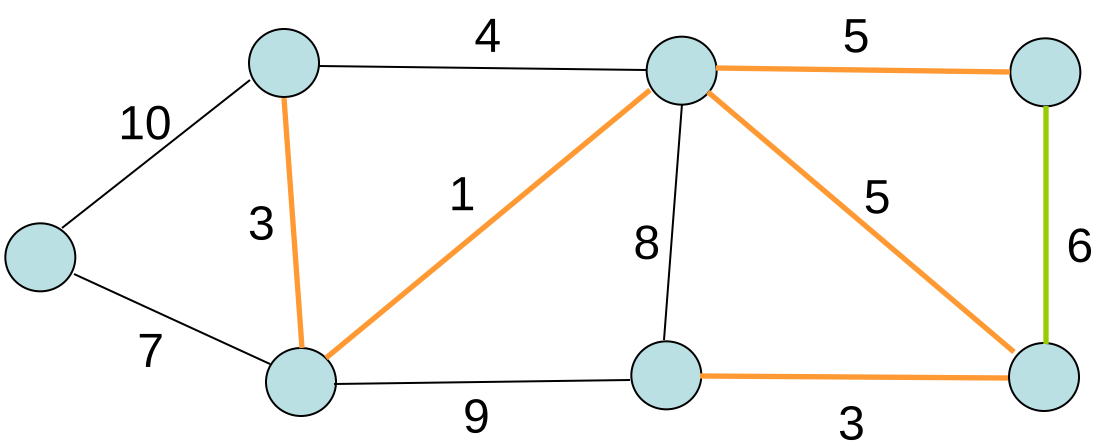
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $\boxed{A \leftarrow A \cup \{(u, v)\}}$
6. **return** A



Graphalgorithmen

Kruskal(G)

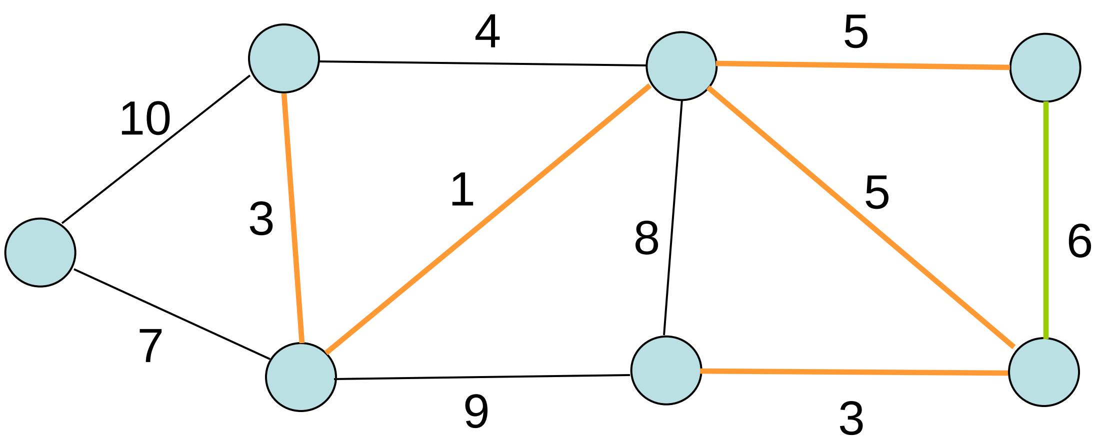
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

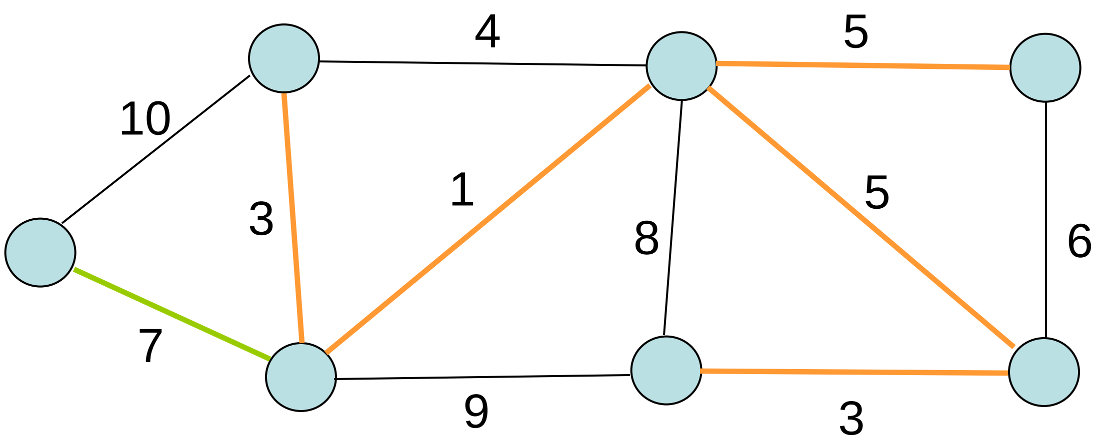
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

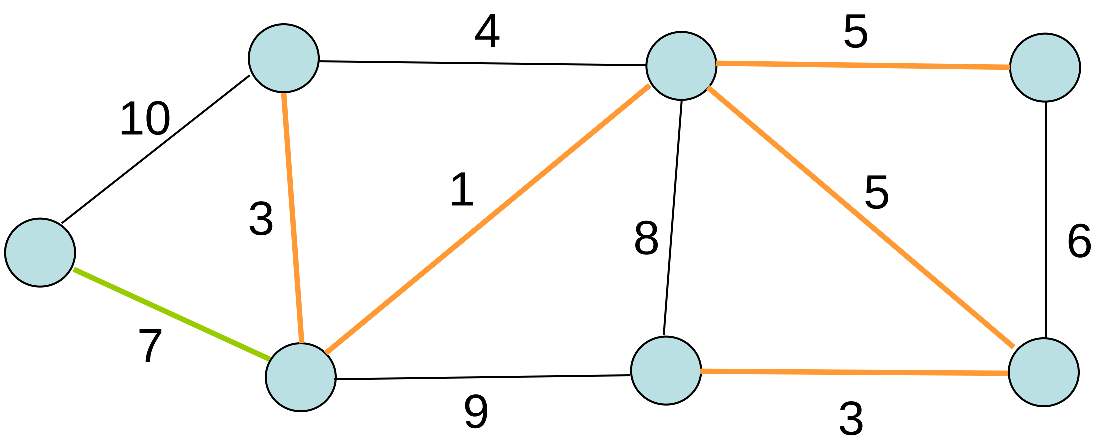
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

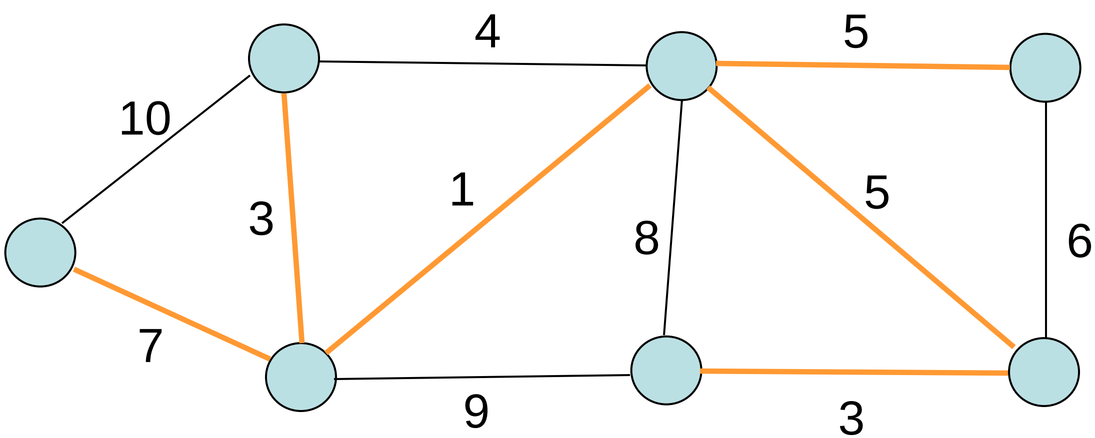
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

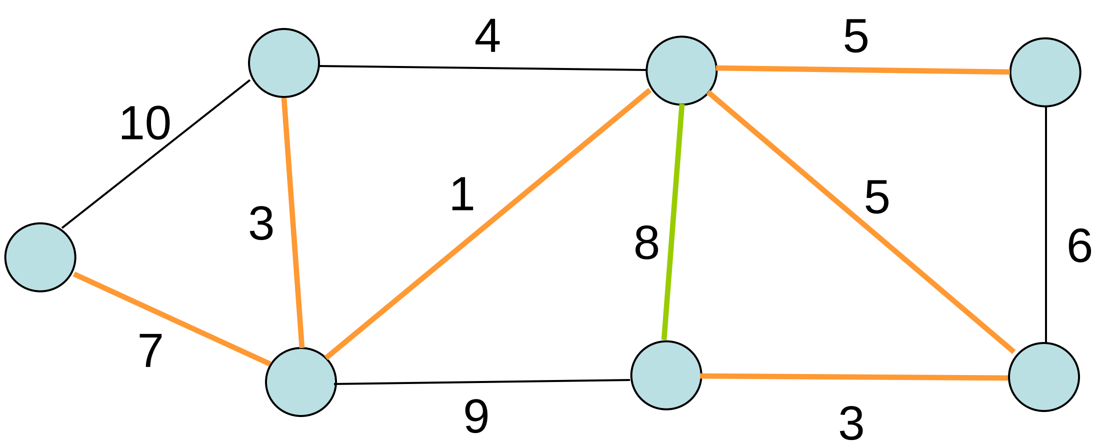
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $\boxed{A \leftarrow A \cup \{(u, v)\}}$
6. **return** A



Graphalgorithmen

Kruskal(G)

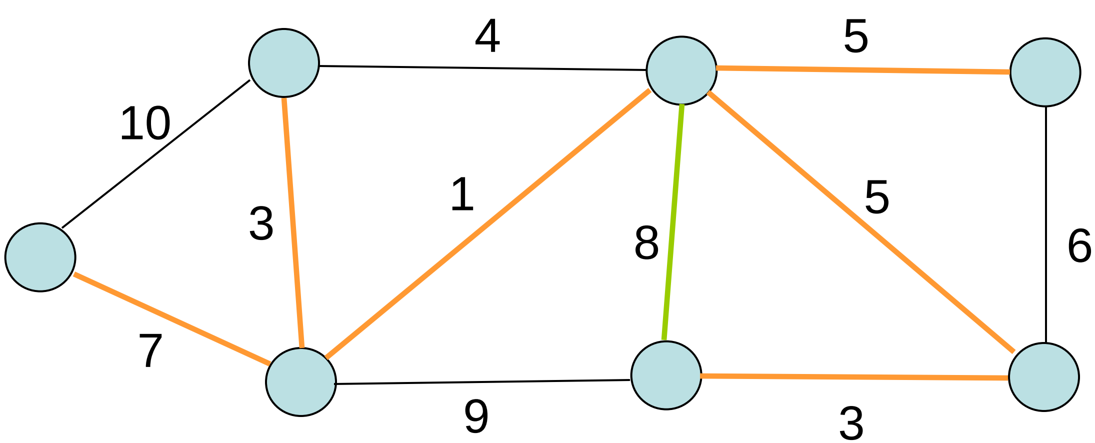
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4.   **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

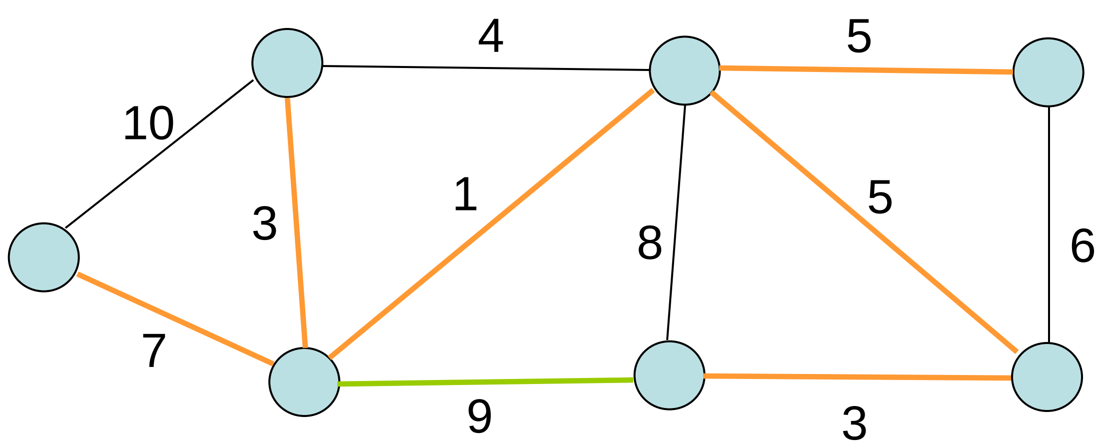
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

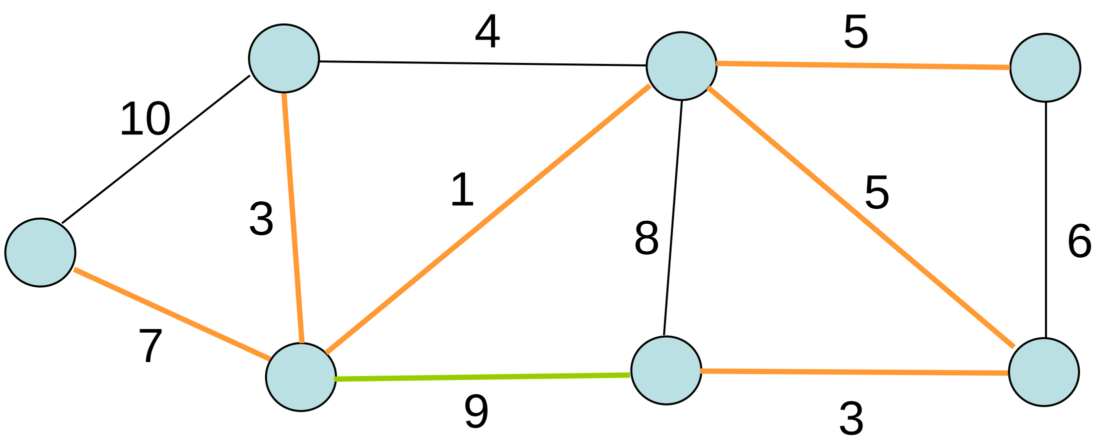
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

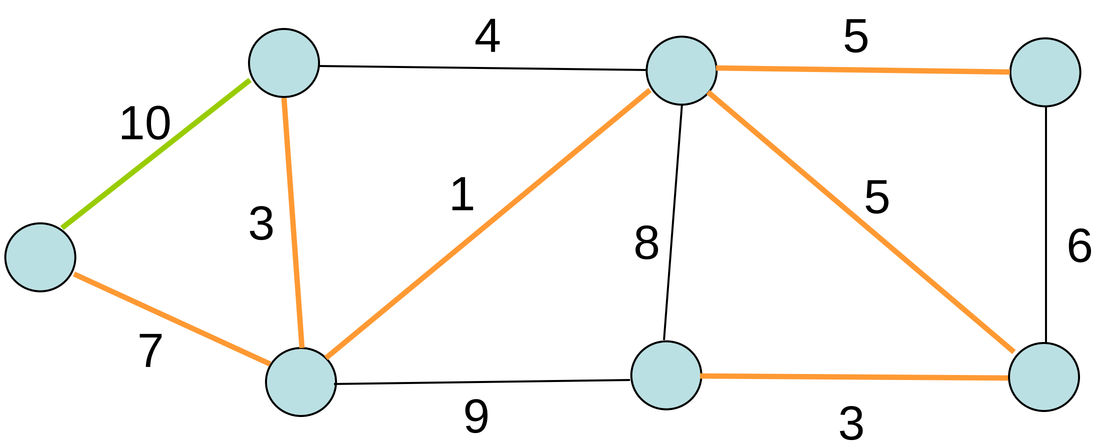
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

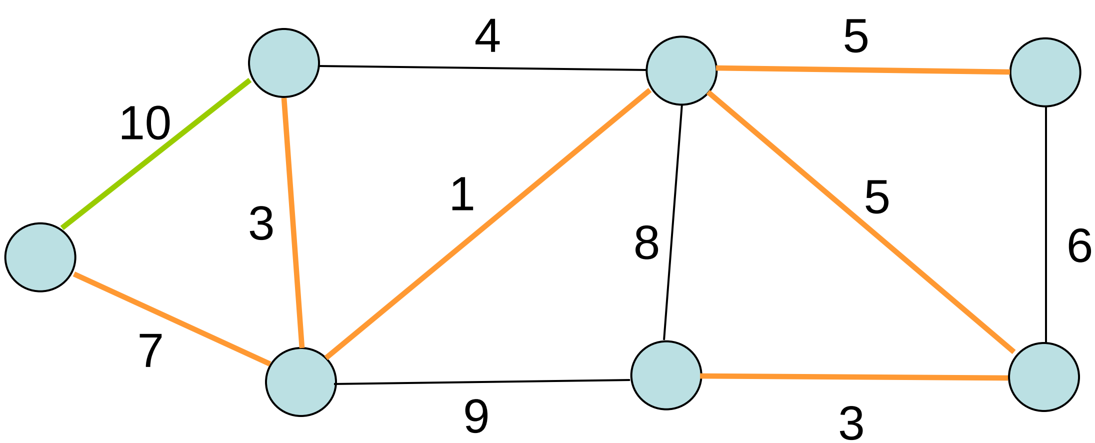
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

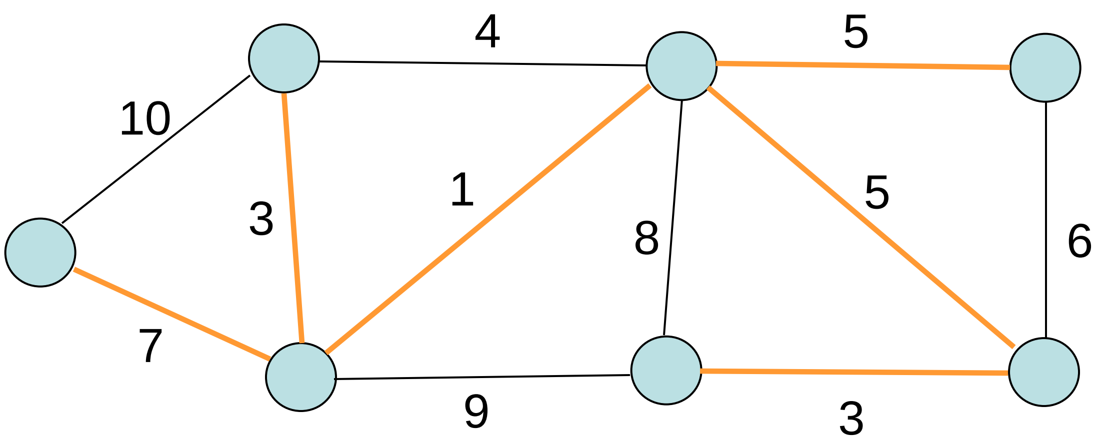
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

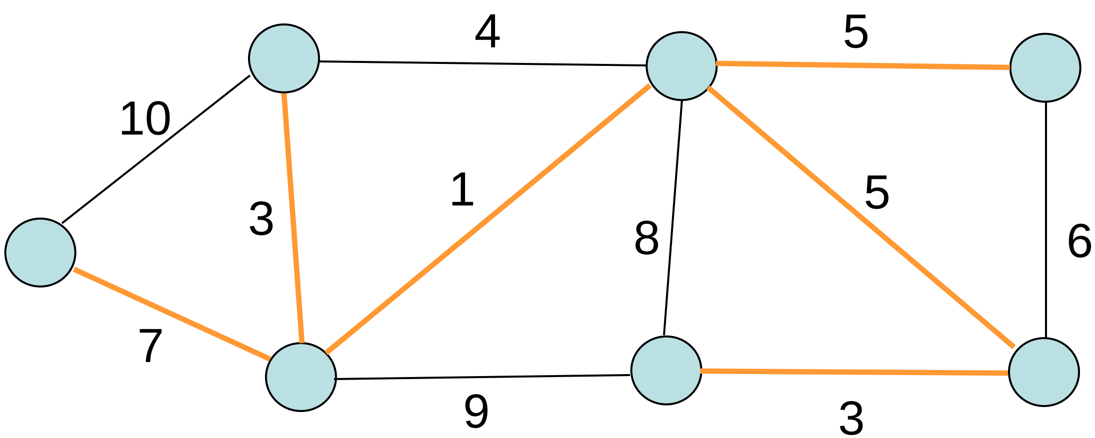
1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. $\boxed{\textbf{return } A}$



Graphalgorithmen

Kruskal(G)

1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A

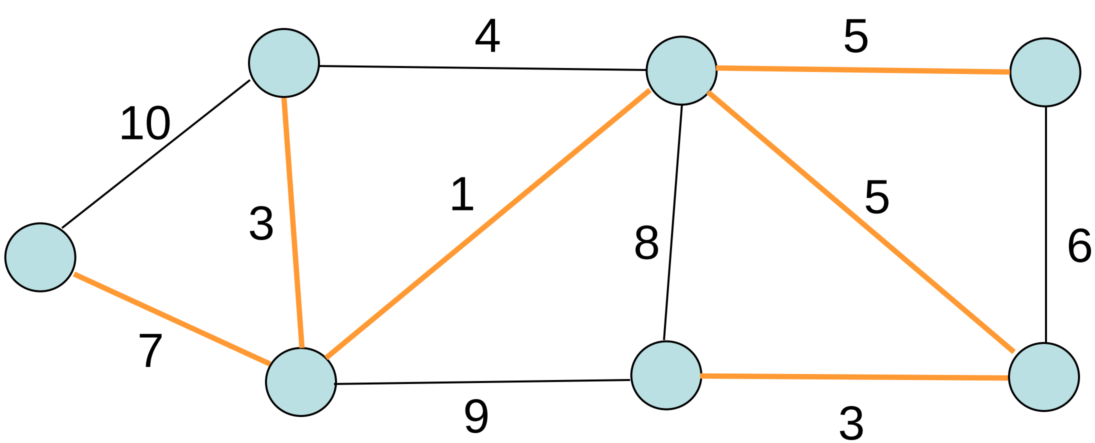


Graphalgorithmen

Laufzeit:
$\mathbf{O}(|E| \log|E| + |E| \cdot$ „Zeit für Zeile 4“$)$

Kruskal(G)

1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A

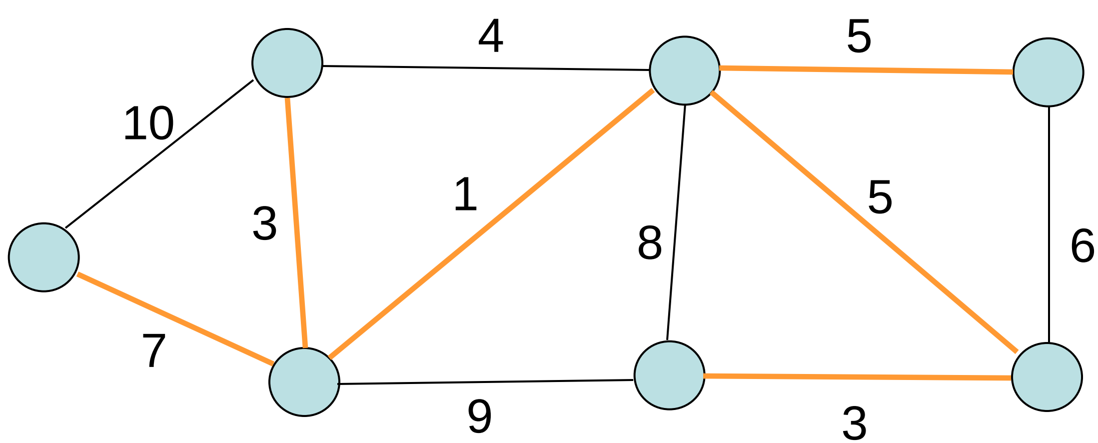


Graphalgorithmen

Wie kann man Zeile 4 implementieren?

Laufzeit:
$\mathbf{O}(|E|\log|E| + |E| \cdot$ „Zeit für Zeile 4“)

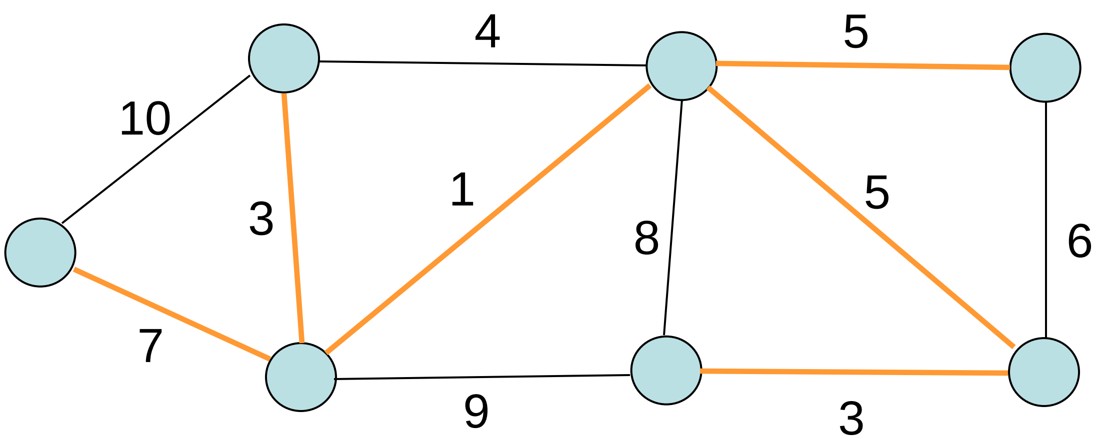
2. Sc… Kanten nach Gewicht
3. **for** e… **h** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

Kruskal(G)

1. $A \leftarrow \emptyset$
2. Sortiere Kanten nach Gewicht
3. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
4. **if** u und v sind nicht in der selben Zusammenhangs-komponente in Graph $H = (V, A)$ **then**
5. $A \leftarrow A \cup \{(u, v)\}$
6. **return** A



Graphalgorithmen

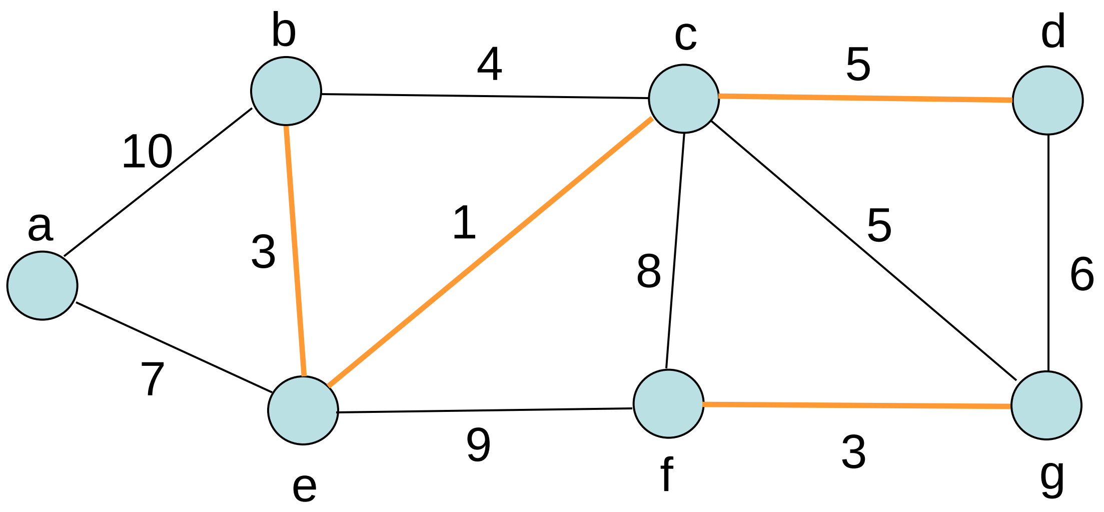
Union-Find Datenstrukturen

- Familie von *disjunkten* Mengen $S = \{S_1, \dots, S_k\}$
- Für jede Menge gibt es einen Repräsentanten
- Make-Set(x): Erzeuge neue Menge, die nur x enthält
- Union(x,y): Vereinigung der Mengen, die x bzw. y enthalten
- Find(x): Gibt Referenz auf den Repräsentanten der Menge, die x enthält

Graphalgorithmen

Idee

- Disjunkte Mengen bei Union-Find sind Knoten des Graphen bei Kruskal



Im Beispiel:
{a}, {b,c,d,e}, {f,g}

Graphalgorithmen

Kruskal(G)

1. $A \leftarrow \emptyset$
2. **for each** vertex $v \in V$ **do** Make-Set(v)
3. Sortiere Kanten nach Gewicht
4. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
5.   **if** Find(u) $\neq$ Find(v) **then**
6. $A \leftarrow A \cup \{(u, v)\}$
7.   Union(u, v)
8. **return** A

Graphalgorithmen

Zu Beginn ist jeder Knoten eine Zusammenhangskomponente in $H=(V,A)$

Kruskal(G)

1. $A \leftarrow \emptyset$
2. **for each** vertex $v \in V$ **do** Make-Set(v)
3. Sortiere Kanten nach Gewicht
4. **for each** $(u,v) \in E$ geordnet nach aufsteigendem Gewicht **do**
5.   **if** Find(u) $\neq$ Find(v) **then**
6. $A \leftarrow A \cup \{(u,v)\}$
7.   Union(u,v)
8. **return** A

Graphalgorithmen

Kruskal(G)

1. $A \leftarrow \emptyset$
2. **for each** vertex $v \in V$ **do**
3. Sortiere Kanten nach Gew
4. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
5. $\boxed{\textbf{if } \mathrm{Find}(u) \neq \mathrm{Find}(v) \textbf{ then}}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. Union(u, v)
8. **return** A

Sind u und v in derselben Zusammenhangskomponente?

Graphalgorithmen

Kruskal(G)

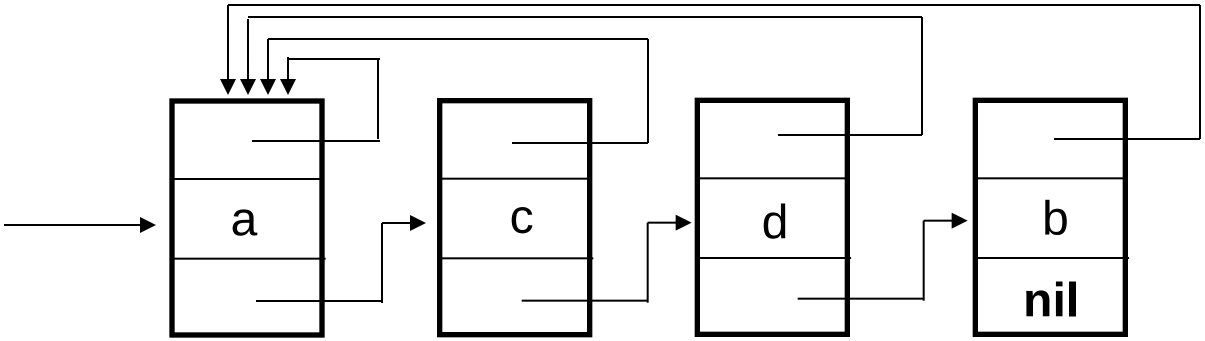
1. $A \leftarrow \emptyset$
2. **for each** vertex $v \in V$ **do** Make-Set(v)
3. Sortiere Kanten nach Gewicht
4. **for each** (u, v) … Gewicht **do**
5. **if** Find(u) $\neq$ …
6. $A \leftarrow A \cup \{(u, v)$ …
7. $\boxed{\text{Union}(u, v)}$
8. **return** A

Wenn ja, dann müssen die Zusammenhangskomponenten vereinigt werden

Graphalgorithmen

Eine einfache Union-Find Datenstruktur

- Jede Menge ist Liste
- Erstes Element ist Repräsentant
- Jedes Listenelement enthält Zeiger auf den Repräsentanten



Graphalgorithmen

Implementierung

- $\mathrm{Make\text{-}Set}$ in $\mathbf{O}(1)$ Zeit einfach
- Find in $\mathbf{O}(1)$ Zeit einfach
- Union: Hänge die eine Liste hinter die andere und aktualisiere alle Zeiger

Laufzeit

- Betrachte Sequenz von m Operationen aus $\mathrm{Make\text{-}Set}$, Find, und Union
- Laufzeit für Union $\mathbf{O}(m)$ (da wir höchstens m Elemente haben)

Graphalgorithmen

Beobachtung

- Wir hängen evtl. immer eine sehr lange Liste an eine sehr kurze
- Wenn wir immer die kurze hinter die lange hängen, müssen wir nur die Referenzen in der kurzen Liste aktualisieren
- Aber bringt das etwas (mehr als Konstanten)?

Graphalgorithmen

Beobachtung

- Wir hängen evtl. immer eine sehr lange Liste an eine sehr kurze
- Wenn wir immer die kurze hinter die lange hängen, müssen wir nur die Referenzen in der kurzen Liste aktualisieren
- Aber bringt das etwas (mehr als Konstanten)?

Ja!

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Wir analysieren zunächst, wie oft der Repräsentantenzeiger eines Elements einer Menge der Größe k maximal aktualisiert wurde

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Wir analysieren zunächst, wie oft der Repräsentantenzeiger eines Elements einer Menge der Größe k maximal aktualisiert wurde
- Betrachte Element x

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Wir analysieren zunächst, wie oft der Repräsentantenzeiger eines Elements einer Menge der Größe k maximal aktualisiert wurde
- Betrachte Element x
- Jedes mal, wenn der Repräsentantenzeiger von x aktualisiert wurde, war x in der kleineren der vereinigten Mengen

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Wir analysieren zunächst, wie oft der Repräsentantenzeiger eines Elements einer Menge der Größe k maximal aktualisiert wurde
- Betrachte Element x
- Jedes mal, wenn der Repräsentantenzeiger von x aktualisiert wurde, war x in der kleineren der vereinigten Mengen
- Damit hat sich die Größe der Menge mindestens verdoppelt

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Wir analysieren zunächst, wie oft der Repräsentantenzeiger eines Elements einer Menge der Größe k maximal aktualisiert wurde
- Betrachte Element x
- Jedes mal, wenn der Repräsentantenzeiger von x aktualisiert wurde, war x in der kleineren der vereinigten Mengen
- Damit hat sich die Größe der Menge mindestens verdoppelt

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Damit gilt für jedes $k \leq n$, dass nach $\lceil \log k \rceil$ Aktualisierungen des Repräsentantenzeigers von x, die Menge, die x enthält, mindestens k Elemente besitzt

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Damit gilt für jedes $k \le n$, dass nach $\lceil \log k \rceil$ Aktualisierungen des Repräsentantenzeigers von x, die Menge, die x enthält, mindestens k Elemente besitzt
- Da die größte Menge maximal n Elemente besitzt, wurde jeder Repräsentantenzeiger maximal $\mathbf{O}(\log n)$ mal aktualisiert (über alle Union-Operationen)

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m+ n\log n)$ Zeit.

Beweis

- Damit gilt für jedes $k \le n$, dass nach $\lceil \log k \rceil$ Aktualisierungen des Repräsentantenzeigers von x, die Menge, die x enthält, mindestens k Elemente besitzt
- Da die größte Menge maximal n Elemente besitzt, wurde jeder Repräsentantenzeiger maximal $\mathbf{O}(\log n)$ mal aktualisiert (über alle Union-Operationen)
- Damit ist die Gesamtlaufzeit für die Aktualisierungen der n Objekte $\mathbf{O}(n \log n)$

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Damit gilt für jedes $k \leq n$, dass nach $\lceil \log k \rceil$ Aktualisierungen des Repräsentantenzeigers von x, die Menge, die x enthält, mindestens k Elemente besitzt
- Da die größte Menge maximal n Elemente besitzt, wurde jeder Repräsentantenzeiger maximal $\mathbf{O}(\log n)$ mal aktualisiert (über alle Union-Operationen)
- Damit ist die Gesamtlaufzeit für die Aktualisierungen der n Objekte $\mathbf{O}(n \log n)$

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Jedes Make-Set und Find benötigt $\mathbf{O}(1)$ Zeit und es gibt $\mathbf{O}(m)$ davon

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Jedes Make-Set und Find benötigt $\mathbf{O}(1)$ Zeit und es gibt $\mathbf{O}(m)$ davon
- Damit ist die gesamte Laufzeit für die Sequenz $\mathbf{O}(m + n \log n)$

Graphalgorithmen

Satz 71

Wenn wir verkettete Listen als Union-Find Datenstruktur benutzen und bei einer Union Operation immer die kürzere hinter die längere Liste hängen und entsprechend aktualisieren, dann benötigt eine Sequenz von m Operationen aus Make-Set, Union und Find, von denen n Operationen Make-Set sind, $\mathbf{O}(m + n \log n)$ Zeit.

Beweis

- Jedes Make-Set und Find benötigt $\mathbf{O}(1)$ Zeit und es gibt $\mathbf{O}(m)$ davon
- Damit ist die gesamte Laufzeit für die Sequenz $\mathbf{O}(m + n \log n)$

Graphalgorithmen

Kruskal(G)

1. $A \leftarrow \emptyset$
2. **for each** vertex $v \in V$ **do** Make-Set(v)
3. Sortiere Kanten nach Gewicht
4. **for each** $(u, v) \in E$ geordnet nach aufsteigendem Gewicht **do**
5. **if** Find(u) $\neq$ Find(v) **then**
6. $A \leftarrow A \cup \{(u, v)\}$
7. Union(u, v)
8. **return** A

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Der Algorithmus hält die Invariante aufrecht, dass die Mengen in der Union-Find Datenstruktur den Zusammenhangskomponenten des durch die bisher ausgewählten Kanten A definierten Graphen entsprechen.

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Der Algorithmus hält die Invariante aufrecht, dass die Mengen in der Union-Find Datenstruktur den Zusammenhangskomponenten des durch die bisher ausgewählten Kanten A definierten Graphen entsprechen.
- Da die Kanten in aufsteigender Reihenfolge ihrer Gewichte betrachtet werden, ist jede Kante, die zwei solche Zusammenhangskomponenten verbindet, eine leichte Kante

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Der Algorithmus hält die Invariante aufrecht, dass die Mengen in der Union-Find Datenstruktur den Zusammenhangskomponenten des durch die bisher ausgewählten Kanten A definierten Graphen entsprechen.
- Da die Kanten in aufsteigender Reihenfolge ihrer Gewichte betrachtet werden, ist jede Kante, die zwei solche Zusammenhangskomponenten verbindet, eine leichte Kante
- Somit ist sie nach Korollar 70 auch sicher

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Der Algorithmus hält die Invariante aufrecht, dass die Mengen in der Union-Find Datenstruktur den Zusammenhangskomponenten des durch die bisher ausgewählten Kanten A definierten Graphen entsprechen.
- Da die Kanten in aufsteigender Reihenfolge ihrer Gewichte betrachtet werden, ist jede Kante, die zwei solche Zusammenhangskomponenten verbindet, eine leichte Kante
- Somit ist sie nach Korollar 70 auch sicher
- Wir wissen, dass der Algorithmus nur sichere Kanten in A einfügt

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Der Algorithmus hält die Invariante aufrecht, dass die Mengen in der Union-Find Datenstruktur den Zusammenhangskomponenten des durch die bisher ausgewählten Kanten A definierten Graphen entsprechen.
- Da die Kanten in aufsteigender Reihenfolge ihrer Gewichte betrachtet werden, ist jede Kante, die zwei solche Zusammenhangskomponenten verbindet, eine leichte Kante
- Somit ist sie nach Korollar 70 auch sicher
- Wir wissen, dass der Algorithmus nur sichere Kanten in A einfügt
- Es bleibt zu zeigen, dass A am Ende des Algorithmus ein aufspannender Baum ist

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Der Algorithmus hält die Invariante aufrecht, dass die Mengen in der Union-Find Datenstruktur den Zusammenhangskomponenten des durch die bisher ausgewählten Kanten A definierten Graphen entsprechen.
- Da die Kanten in aufsteigender Reihenfolge ihrer Gewichte betrachtet werden, ist jede Kante, die zwei solche Zusammenhangskomponenten verbindet, eine leichte Kante
- Somit ist sie nach Korollar 70 auch sicher
- Wir wissen, dass der Algorithmus nur sichere Kanten in A einfügt
- Es bleibt zu zeigen, dass A am Ende des Algorithmus ein aufspannender Baum ist

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G=(V,E)$.

Beweis

- Annahme: A ist am Ende des Algorithmus kein aufspannender Baum

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G=(V,E)$.

Beweis

- Annahme: A ist am Ende des Algorithmus kein aufspannender Baum
- Da A nur aus sicheren Kanten besteht, ist A dann Teilmenge eines min. Spannbaums T

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Annahme: A ist am Ende des Algorithmus kein aufspannender Baum
- Da A nur aus sicheren Kanten besteht, ist A dann Teilmenge eines min. Spannbaums T
- Betrachte eine Kante (u, v), aus $T - A$

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Annahme: A ist am Ende des Algorithmus kein aufspannender Baum
- Da A nur aus sicheren Kanten besteht, ist A dann Teilmenge eines min. Spannbaums T
- Betrachte eine Kante (u, v), aus $T - A$
- Das Entfernen von (u, v) aus T definiert einen Schnitt $(S, V - S)$

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Annahme: A ist am Ende des Algorithmus kein aufspannender Baum
- Da A nur aus sicheren Kanten besteht, ist A dann Teilmenge eines min. Spannbaums T
- Betrachte eine Kante (u, v), aus $T - A$
- Das Entfernen von (u, v) aus T definiert einen Schnitt $(S, V - S)$
- Da A keine Kante enthält, die $(S, V - S)$ kreuzt, ist jede Zusammenhangs-komponente von A entweder Teilmenge von S oder von $V - S$

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Annahme: A ist am Ende des Algorithmus kein aufspannender Baum
- Da A nur aus sicheren Kanten besteht, ist A dann Teilmenge eines min. Spannbaums T
- Betrachte eine Kante (u, v), aus $T - A$
- Das Entfernen von (u, v) aus T definiert einen Schnitt $(S, V - S)$
- Da A keine Kante enthält, die $(S, V - S)$ kreuzt, ist jede Zusammenhangs-komponente von A entweder Teilmenge von S oder von $V - S$
- Wenn nun der Algorithmus (u, v) betrachtet, so liegt u in einer Komponente mit Knoten aus S und v in einer Komponente mit Knoten aus $V - S$

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Annahme: A ist am Ende des Algorithmus kein aufspannender Baum
- Da A nur aus sicheren Kanten besteht, ist A dann Teilmenge eines min. Spannbaums T
- Betrachte eine Kante (u, v), aus $T - A$
- Das Entfernen von (u, v) aus T definiert einen Schnitt $(S, V - S)$
- Da A keine Kante enthält, die $(S, V - S)$ kreuzt, ist jede Zusammenhangs-komponente von A entweder Teilmenge von S oder von $V - S$
- Wenn nun der Algorithmus (u, v) betrachtet, so liegt u in einer Komponente mit Knoten aus S und v in einer Komponente mit Knoten aus $V - S$

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Damit liegen u und v in unterschiedlichen Komponenten und somit ist $\mathrm{Find}(u) \neq \mathrm{Find}(v)$

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G=(V,E)$.

Beweis

- Damit liegen u und v in unterschiedlichen Komponenten und somit ist $\mathrm{Find}(u) \neq \mathrm{Find}(v)$
- Dann hätte der Algorithmus aber (u,v) in A aufgenommen. Widerspruch!

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Damit liegen u und v in unterschiedlichen Komponenten und somit ist $\mathrm{Find}(u) \neq \mathrm{Find}(v)$
- Dann hätte der Algorithmus aber (u, v) in A aufgenommen. Widerspruch!
- Somit ist A am Ende des Algorithmus ein aufspannender Baum und auch ein min. Spannbaum, da alle Kanten sicher waren

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Damit liegen u und v in unterschiedlichen Komponenten und somit ist $\mathrm{Find}(u) \neq \mathrm{Find}(v)$
- Dann hätte der Algorithmus aber (u, v) in A aufgenommen. Widerspruch!
- Somit ist A am Ende des Algorithmus ein aufspannender Baum und auch ein min. Spannbaum, da alle Kanten sicher waren

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Laufzeit:
- Der Algorithmus benötigt $\mathbf{O}(|E| \log|E|)$ Zeit zum Sortieren

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Laufzeit:
- Der Algorithmus benötigt $\mathbf{O}(|E| \log|E|)$ Zeit zum Sortieren
- Er führt $\mathbf{O}(|E| + |V|)$ Operationen mit der Union-Find-Datenstruktur durch

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E| \log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Laufzeit:
- Der Algorithmus benötigt $\mathbf{O}(|E| \log|E|)$ Zeit zum Sortieren
- Er führt $\mathbf{O}(|E| + |V|)$ Operationen mit der Union-Find-Datenstruktur durch
- Davon sind $|V|$ Operationen Make-Set

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G=(V,E)$.

Beweis

- Laufzeit:
- Der Algorithmus benötigt $\mathbf{O}(|E|\log|E|)$ Zeit zum Sortieren
- Er führt $\mathbf{O}(|E|+|V|)$ Operationen mit der Union-Find-Datenstruktur durch
- Davon sind $|V|$ Operationen Make-Set
- Somit ist die Laufzeit für die Operationen der Union-Find-Datenstruktur $\mathbf{O}(|E|+|V|\log|V|)$

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G=(V,E)$.

Beweis

- Laufzeit:
- Der Algorithmus benötigt $\mathbf{O}(|E|\log|E|)$ Zeit zum Sortieren
- Er führt $\mathbf{O}(|E|+|V|)$ Operationen mit der Union-Find-Datenstruktur durch
- Davon sind $|V|$ Operationen Make-Set
- Somit ist die Laufzeit für die Operationen der Union-Find-Datenstruktur $\mathbf{O}(|E|+|V|\log|V|)$
- Diese dominieren die Laufzeit der zweiten **for**-Schleife

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G = (V, E)$.

Beweis

- Laufzeit:
- Der Algorithmus benötigt $\mathbf{O}(|E|\log|E|)$ Zeit zum Sortieren
- Er führt $\mathbf{O}(|E| + |V|)$ Operationen mit der Union-Find-Datenstruktur durch
- Davon sind $|V|$ Operationen Make-Set
- Somit ist die Laufzeit für die Operationen der Union-Find-Datenstruktur $\mathbf{O}(|E| + |V|\log|V|)$
- Diese dominieren die Laufzeit der zweiten **for**-Schleife
- Insgesamt ist daher die Laufzeit $\boldsymbol{O}(|E|\log|E|)$ (da $|V| = O(|E|)$ für zusammenhängende Graphen)

Graphalgorithmen

Satz 72

Der Algorithmus von Kruskal berechnet in $\mathbf{O}(|E|\log|E|)$ einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen $G=(V,E)$.

Beweis

- Laufzeit:
- Der Algorithmus benötigt $\mathbf{O}(|E|\log|E|)$ Zeit zum Sortieren
- Er führt $\mathbf{O}(|E|+|V|)$ Operationen mit der Union-Find-Datenstruktur durch
- Davon sind $|V|$ Operationen Make-Set
- Somit ist die Laufzeit für die Operationen der Union-Find-Datenstruktur $\mathbf{O}(|E|+|V|\log|V|)$
- Diese dominieren die Laufzeit der zweiten **for**-Schleife
- Insgesamt ist daher die Laufzeit $\boldsymbol{O}(|E|\log|E|)$ (da $|V|=O(|E|)$ für zusammenhängende Graphen)

Graphalgorithmen

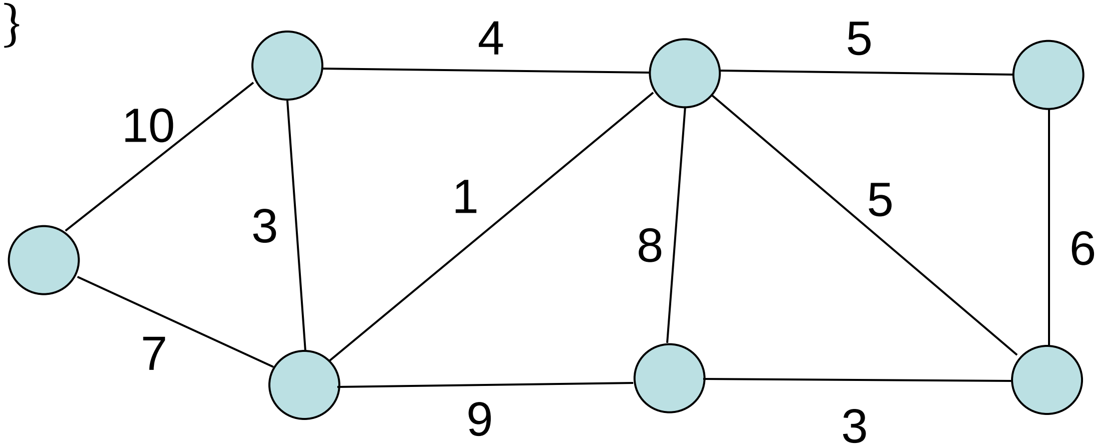
Idee des Algorithmus von Prim

- Verwende generischen Algorithmus
- Nimm immer eine Kante mit minimalem Gewicht, die einen Knoten in Baum A mit einem Knoten verbindet, der nicht in Baum A ist und füge diese zu A hinzu
- Die Kante ist eine leichte Kante, die Baum A mit einem weiteren Knoten verbindet
- Damit ist sie sicher für A

Graphalgorithmen

$\mathrm{Prim}(G, r)$

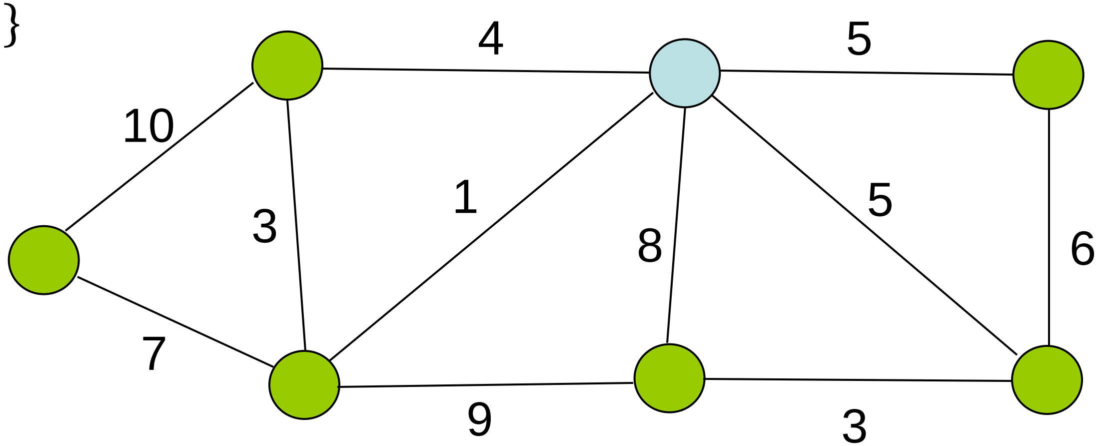
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

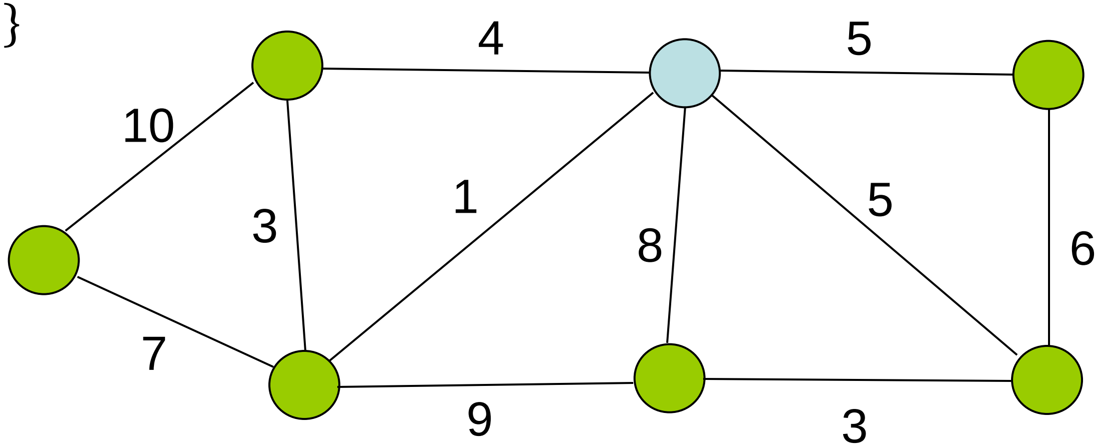
1. $\boxed{Q \leftarrow V - \{r\}}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

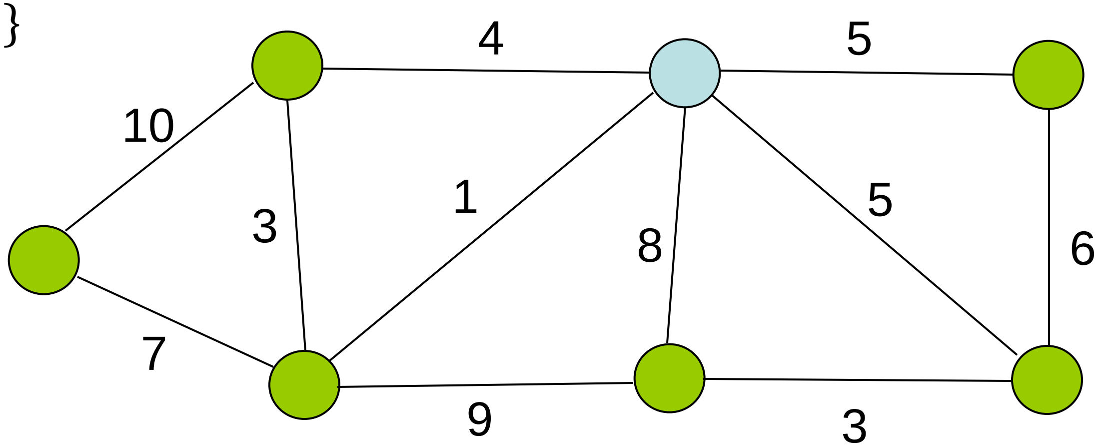
1. $Q \leftarrow V - \{r\}$
2. $\boxed{A \leftarrow \emptyset}$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

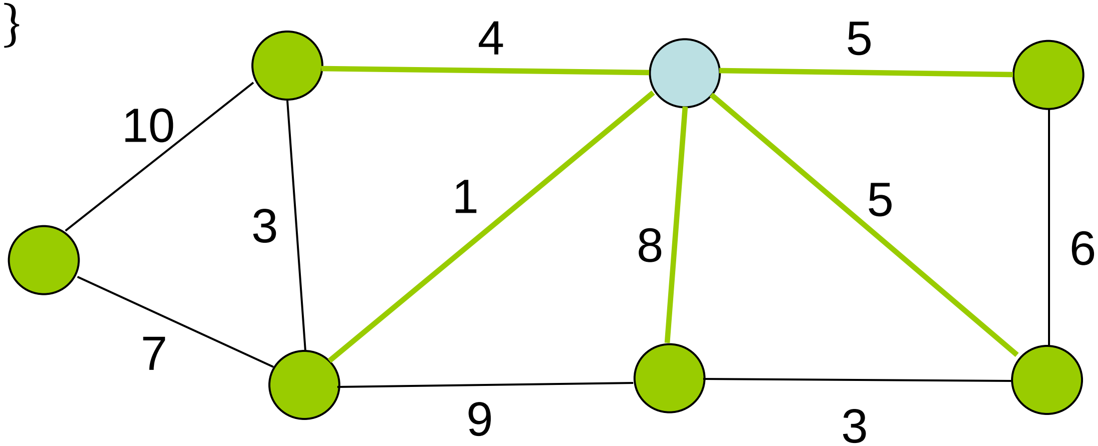
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. $\boxed{\textbf{while } Q \neq \emptyset}$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

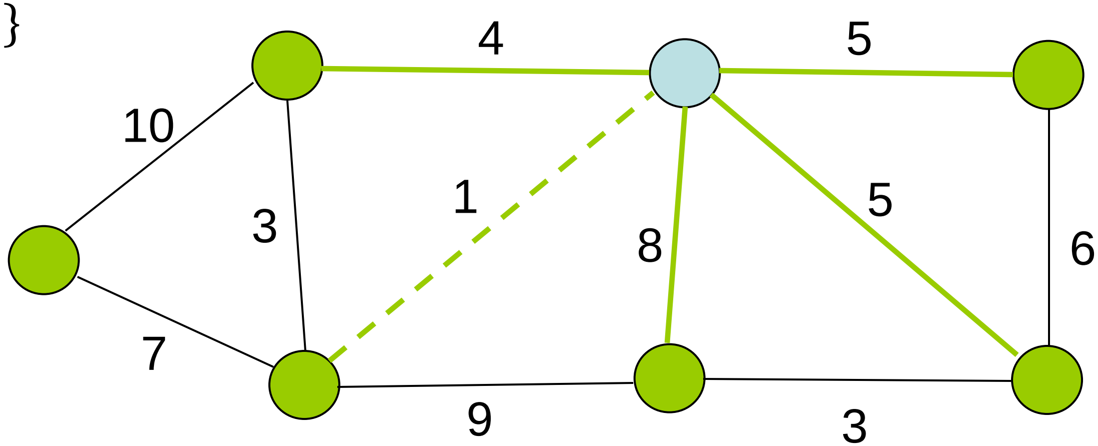
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V-Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

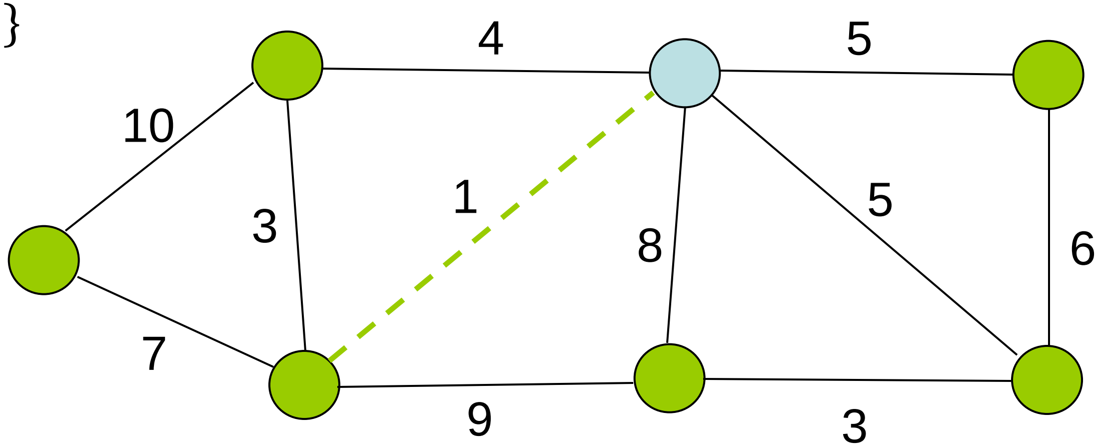
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

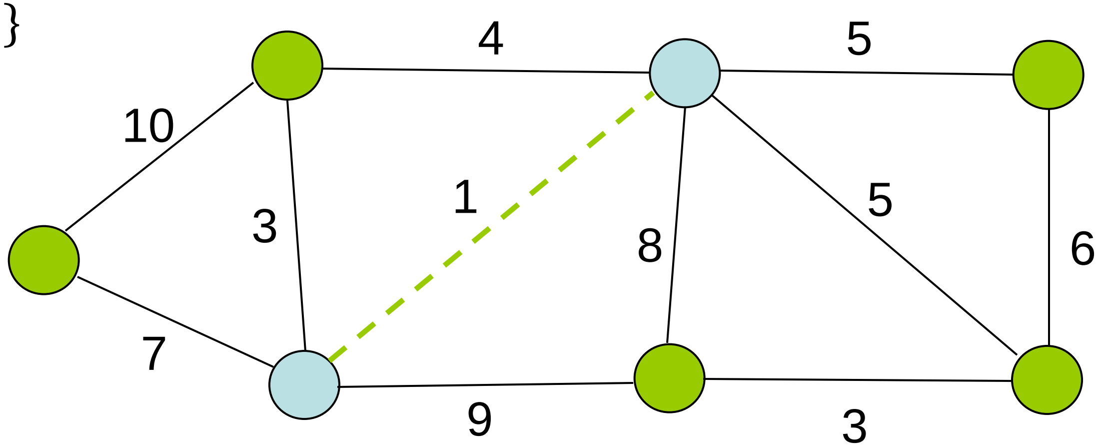
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

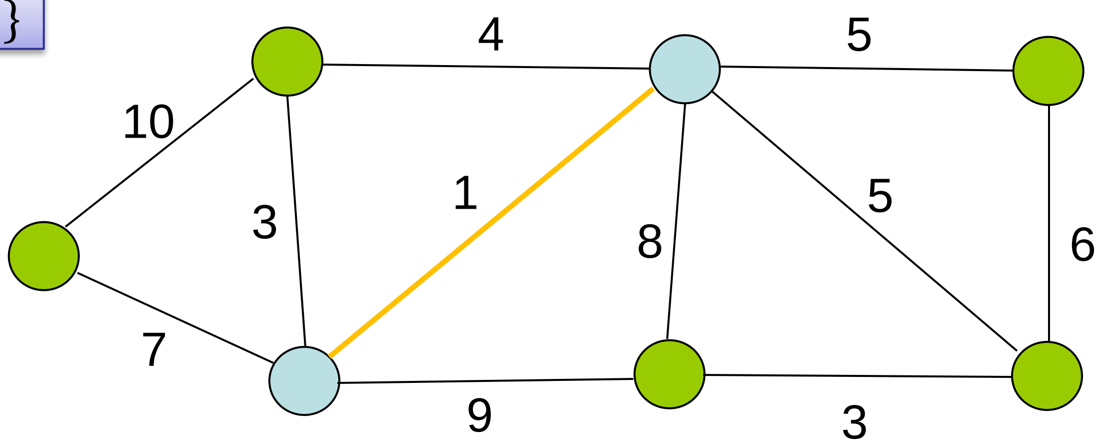
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $\boxed{Q \leftarrow Q - \{u\}}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

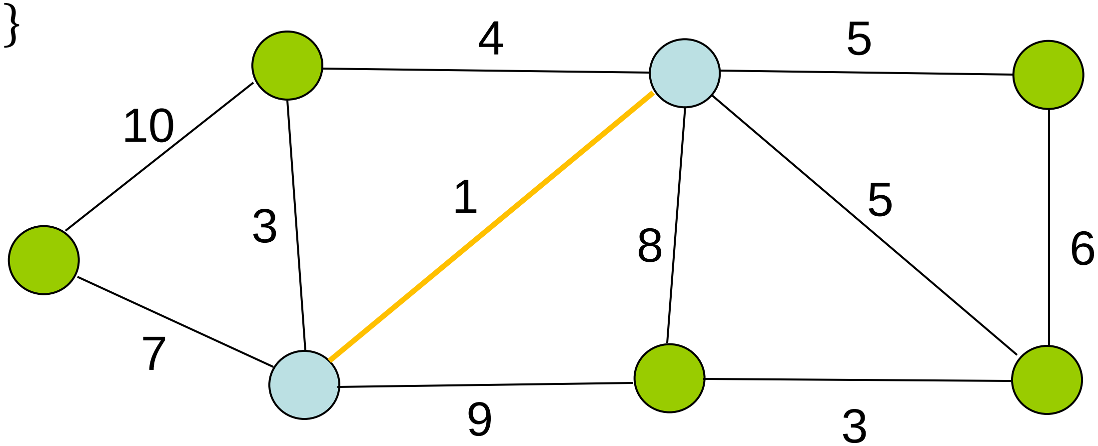
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $\boxed{A \leftarrow A \cup \{(u, v)\}}$
7. **return** A



Graphalgorithmen

Prim(G, r)

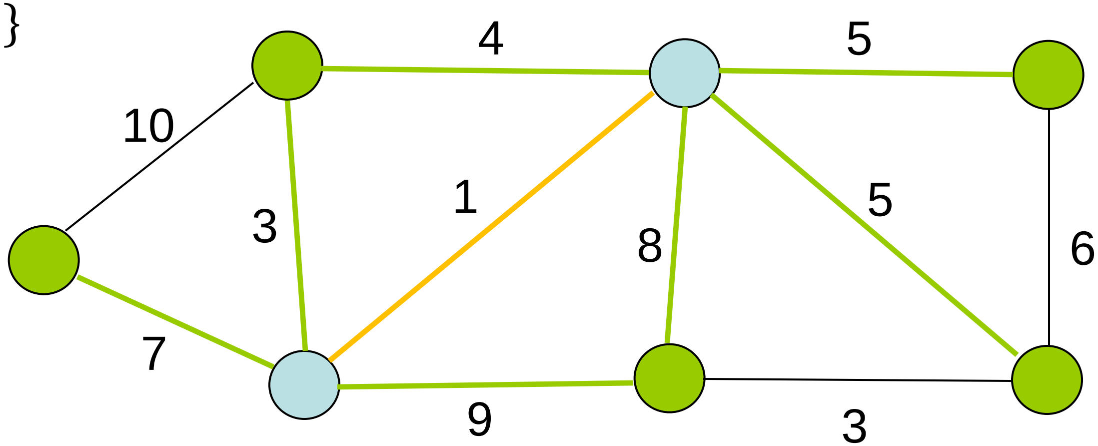
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. $\boxed{\textbf{while } Q \neq \emptyset}$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

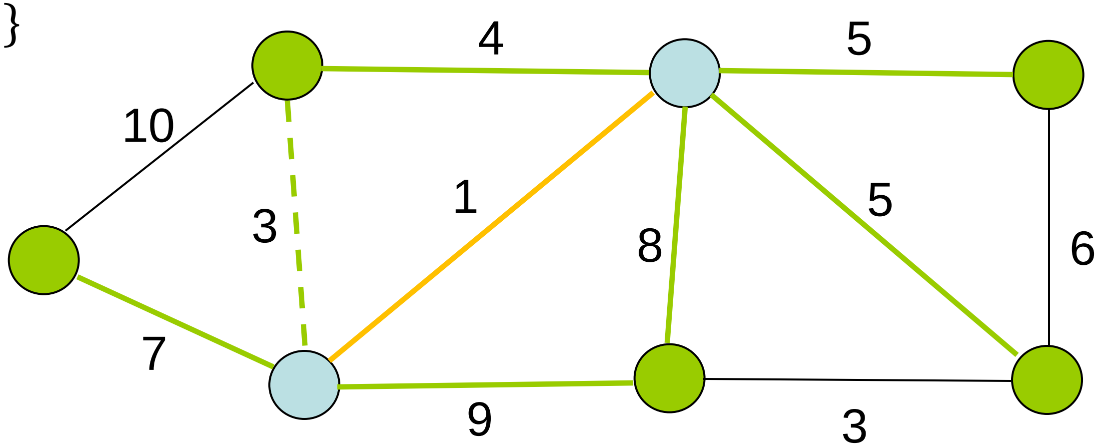
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

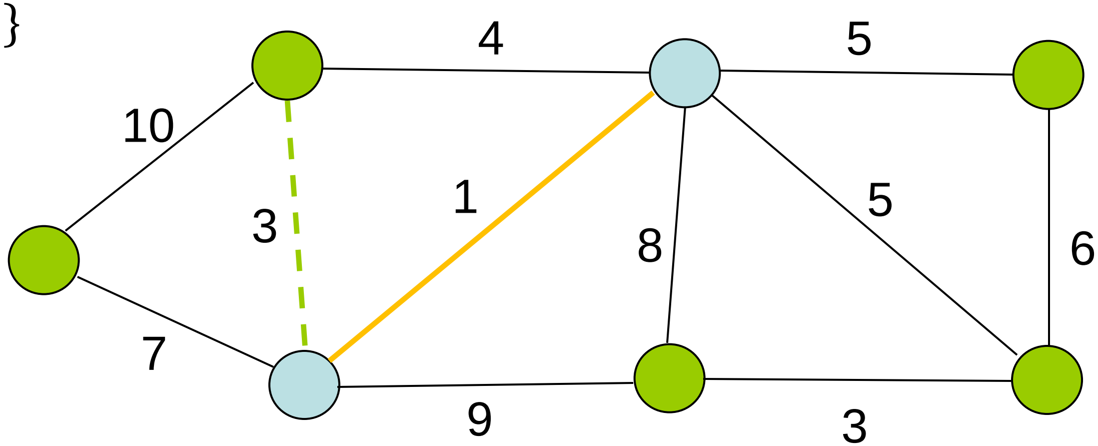
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $\quad Q \leftarrow Q - \{u\}$
6. $\quad A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

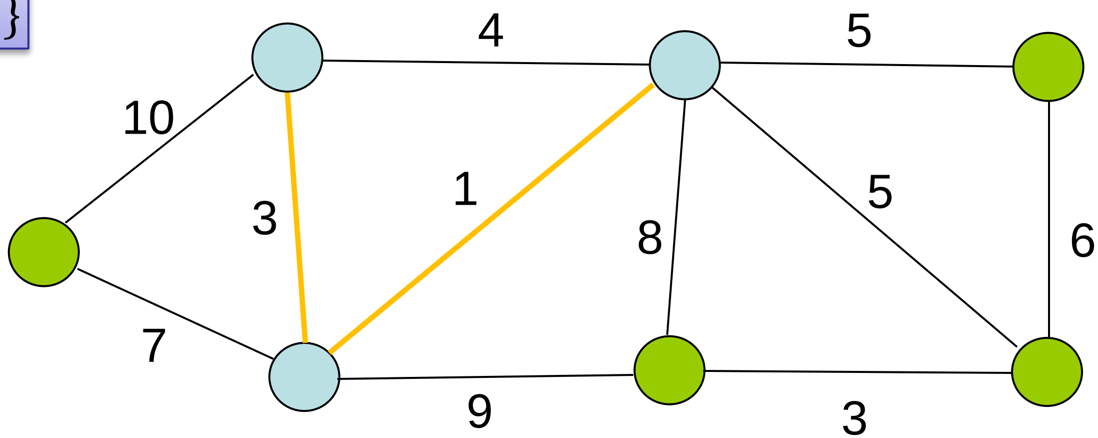
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

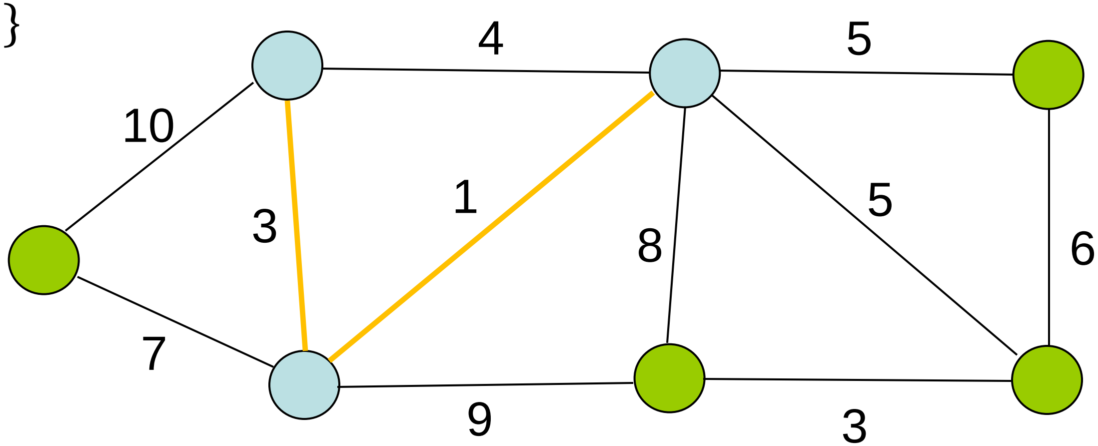
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

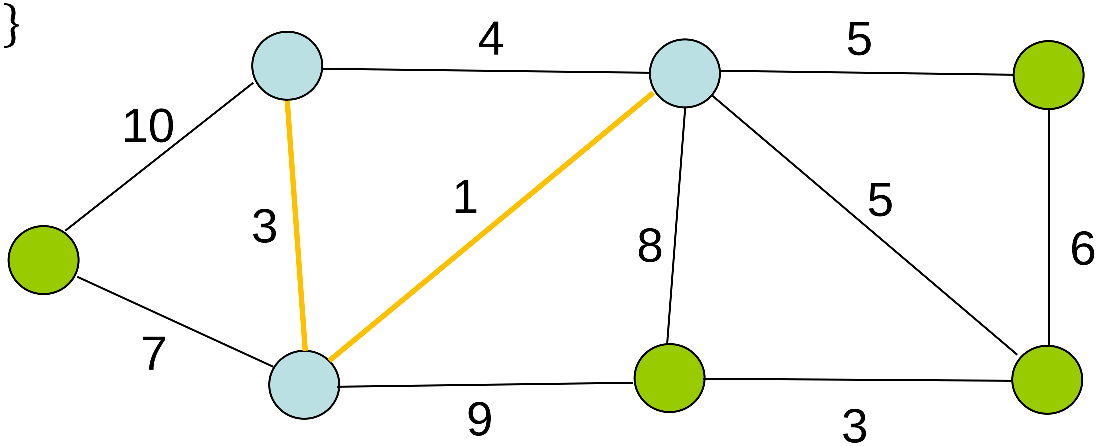
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. $\boxed{\textbf{while } Q \neq \emptyset}$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

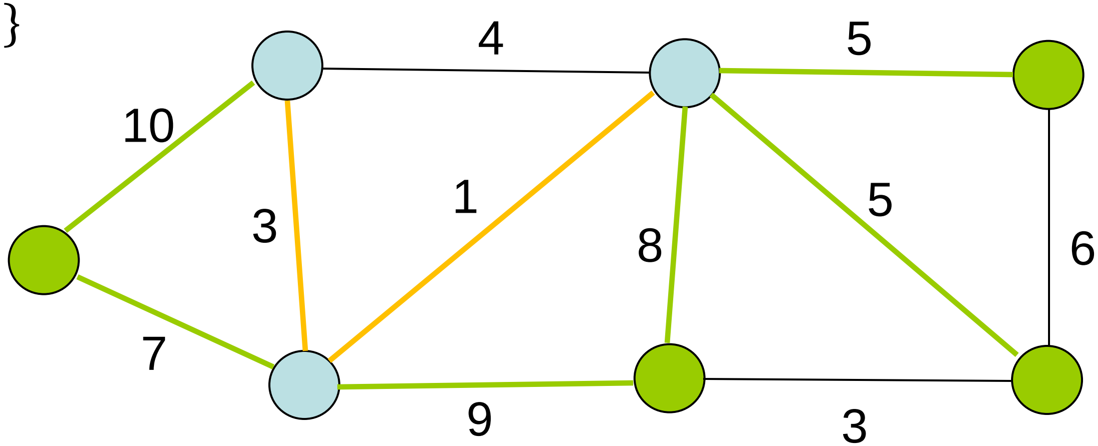
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

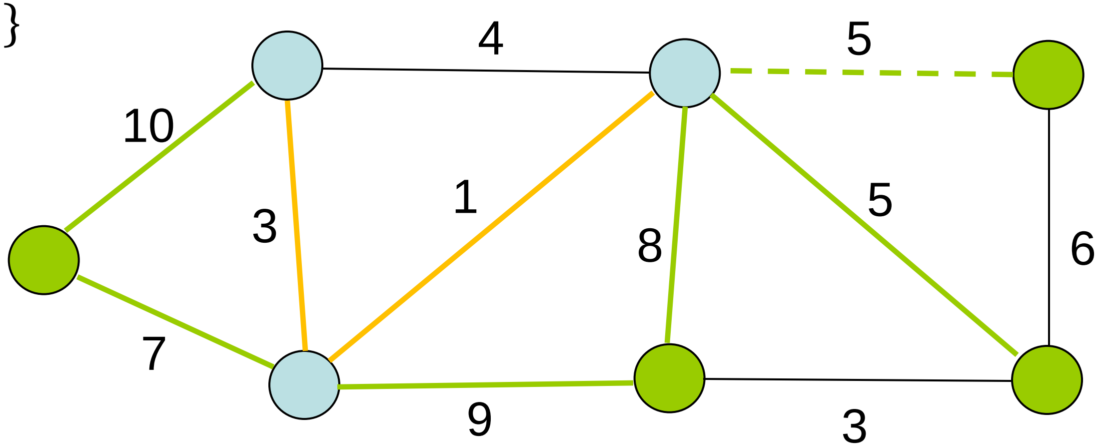
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

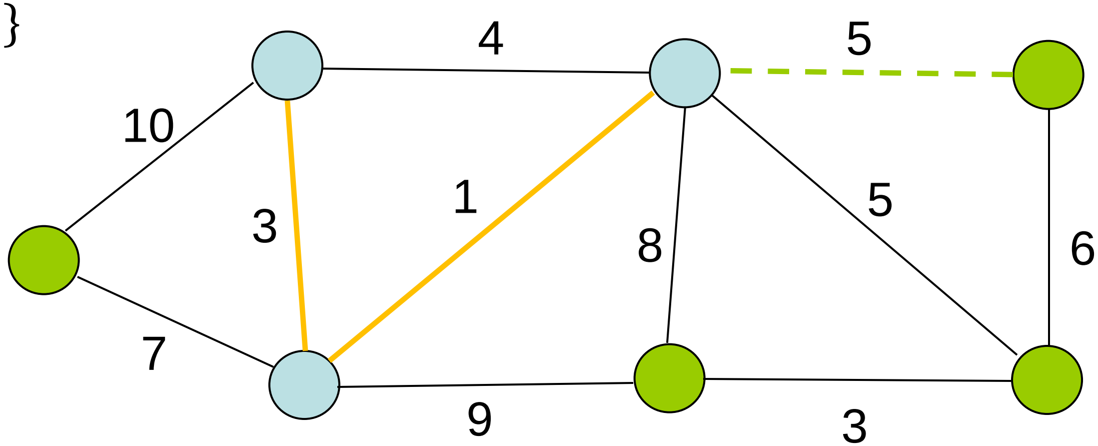
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

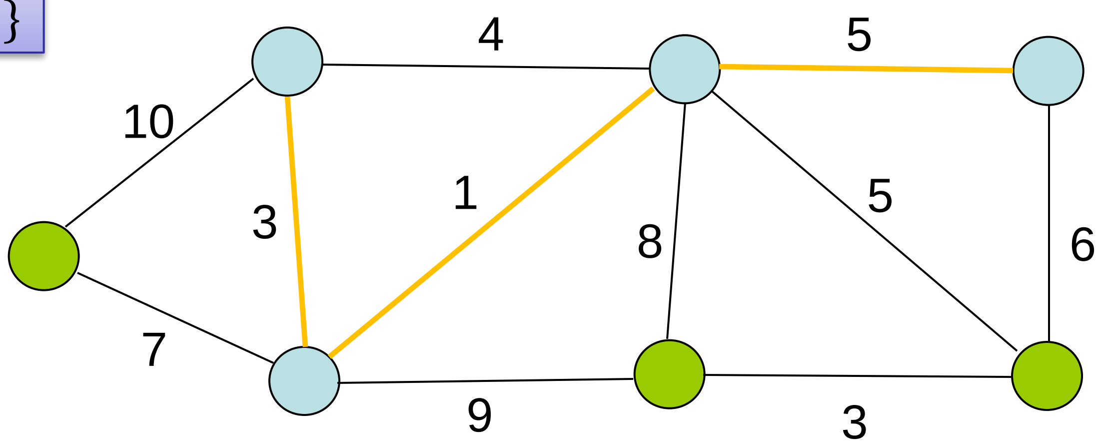
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

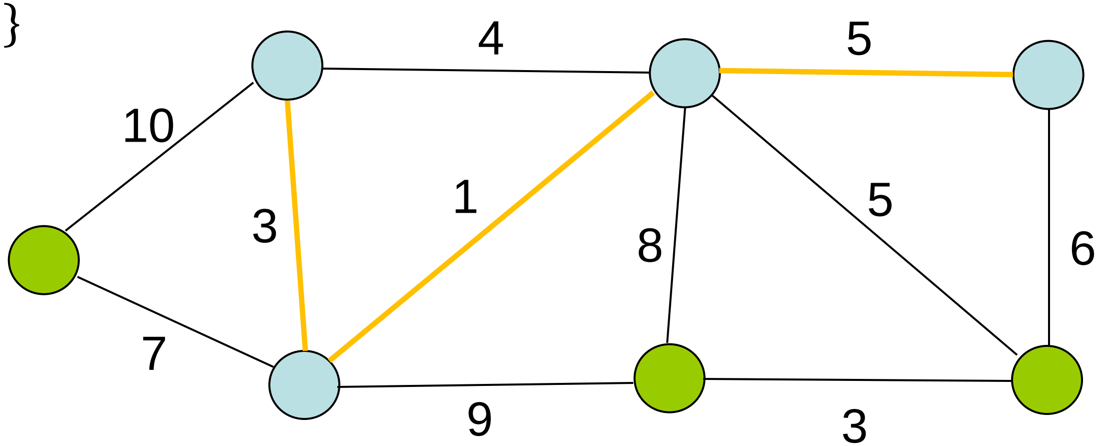
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

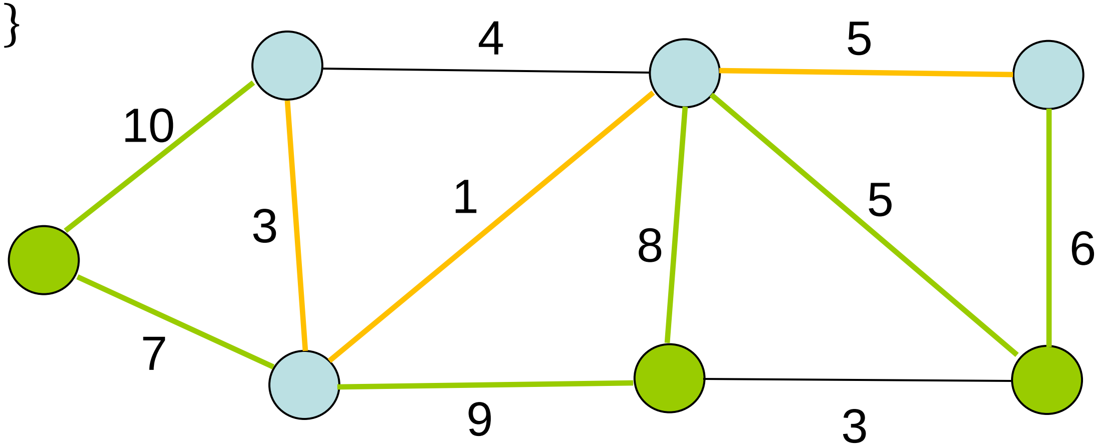
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. $\boxed{\textbf{while } Q \neq \emptyset}$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

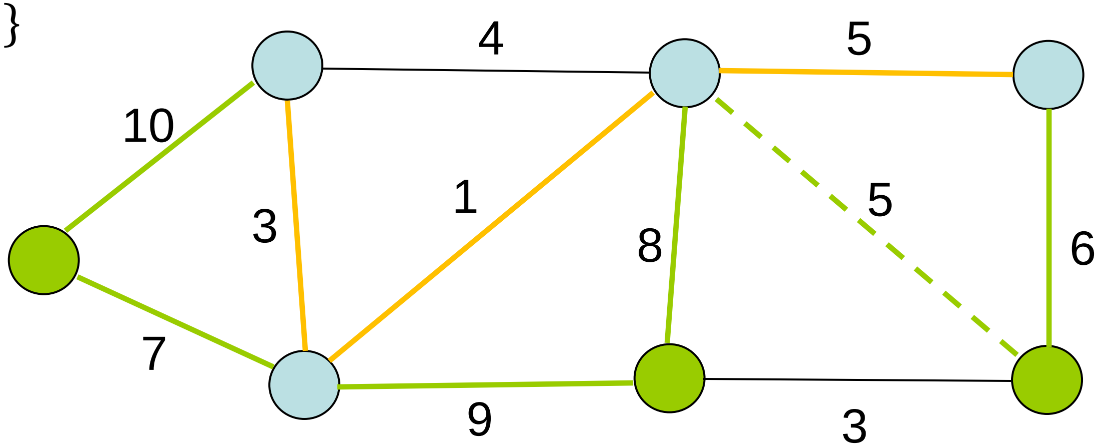
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

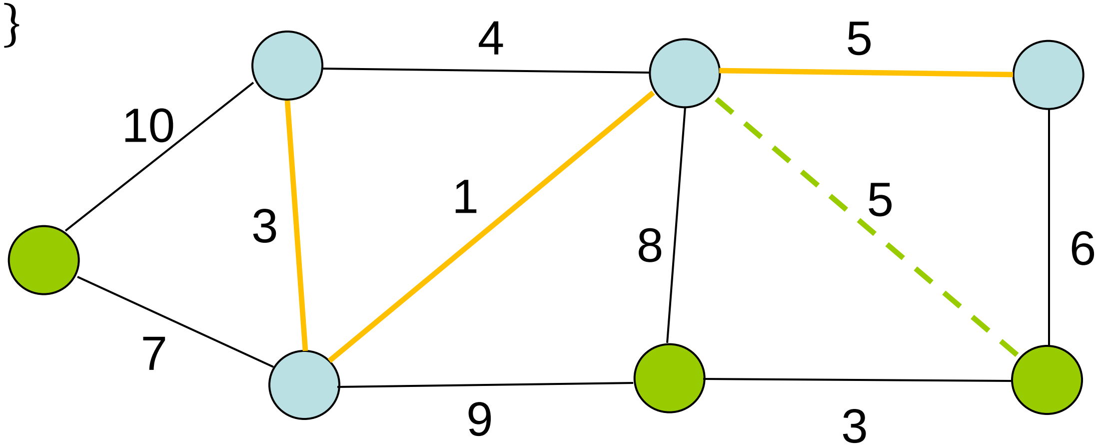
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $\quad Q \leftarrow Q - \{u\}$
6. $\quad A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

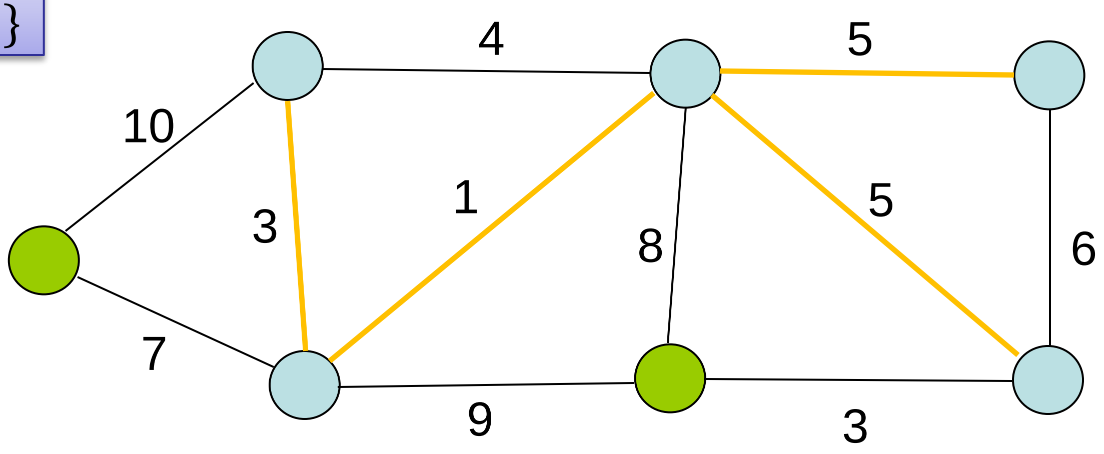
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

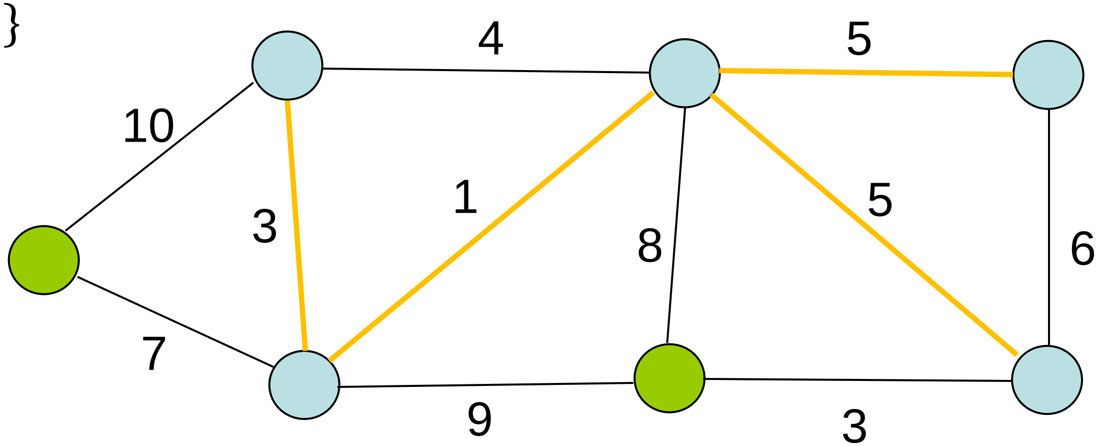
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

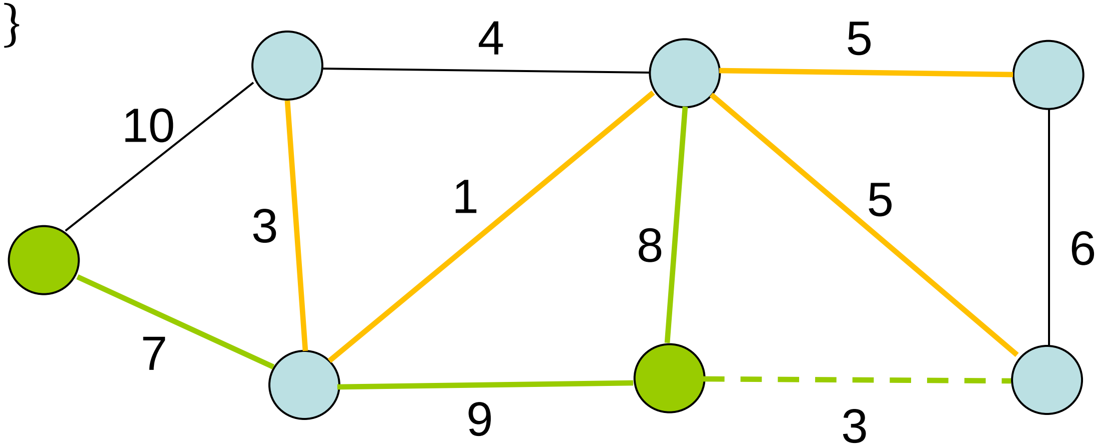
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

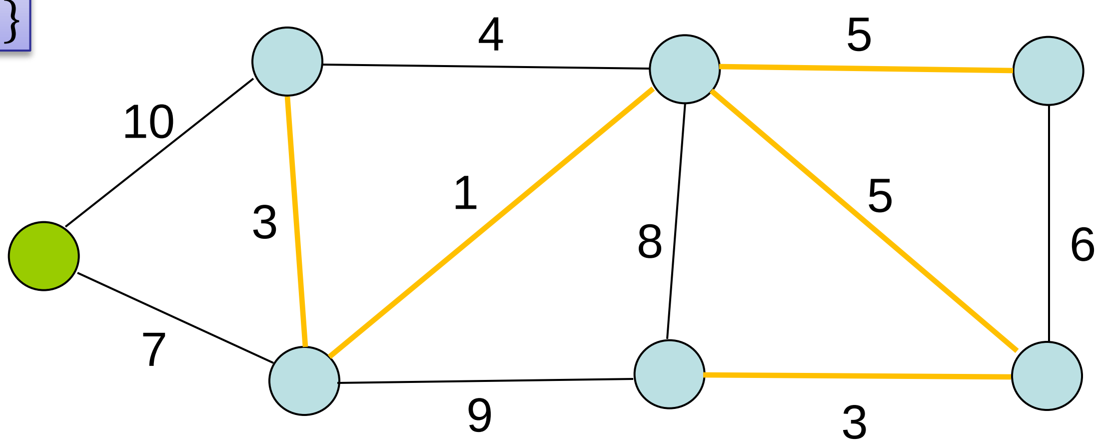
Prim(G, r)

1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

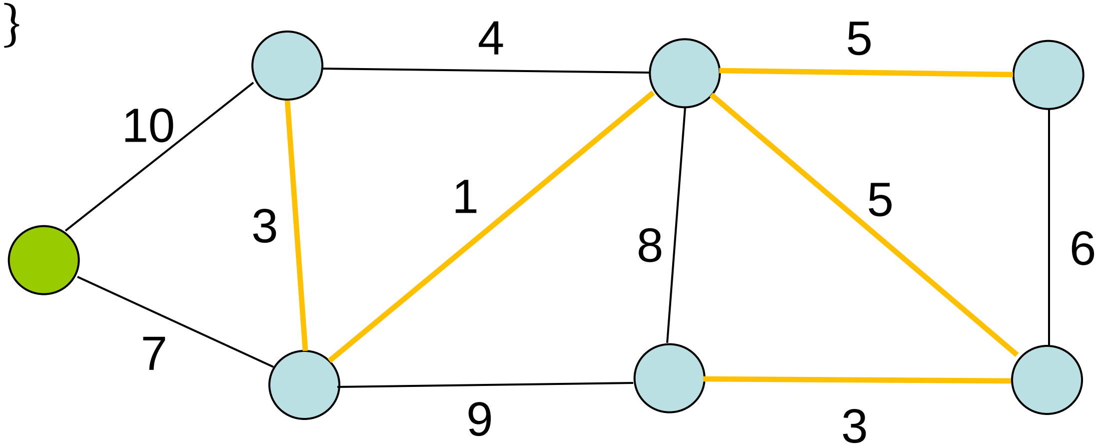
Prim(G, r)
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

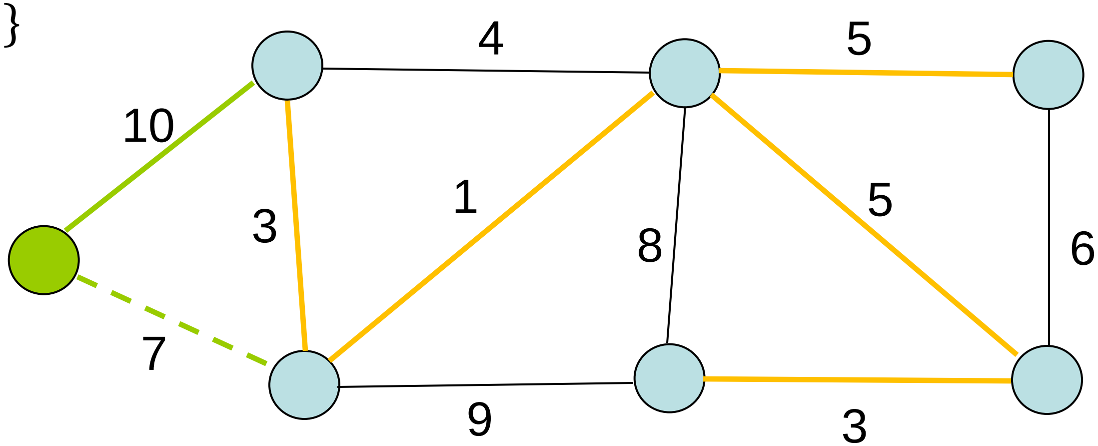
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. $\boxed{\textbf{while } Q \neq \emptyset}$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

Prim(G, r)

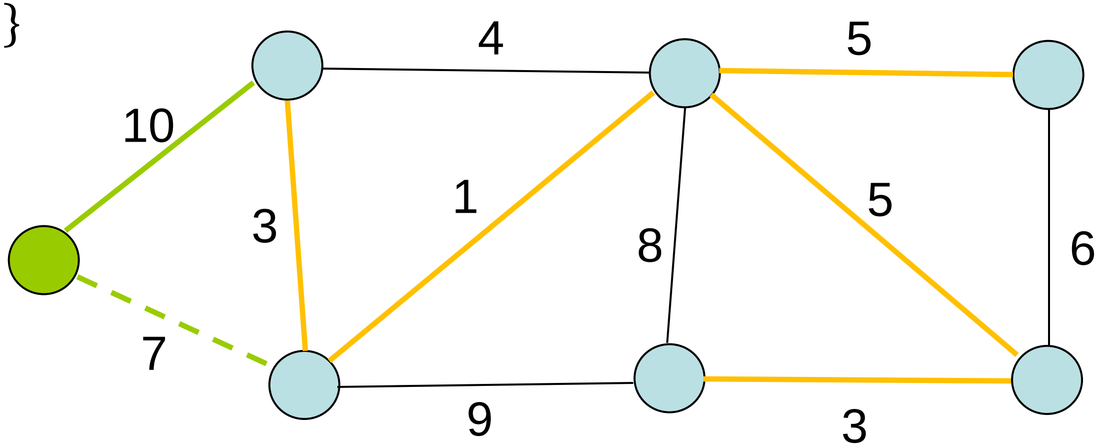
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $\quad Q \leftarrow Q - \{u\}$
6. $\quad A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

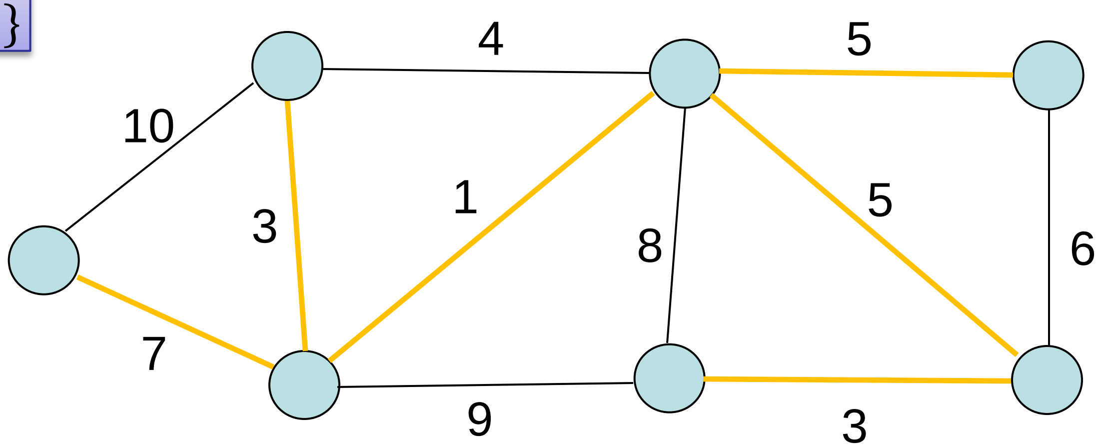
Prim(G, r)

1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



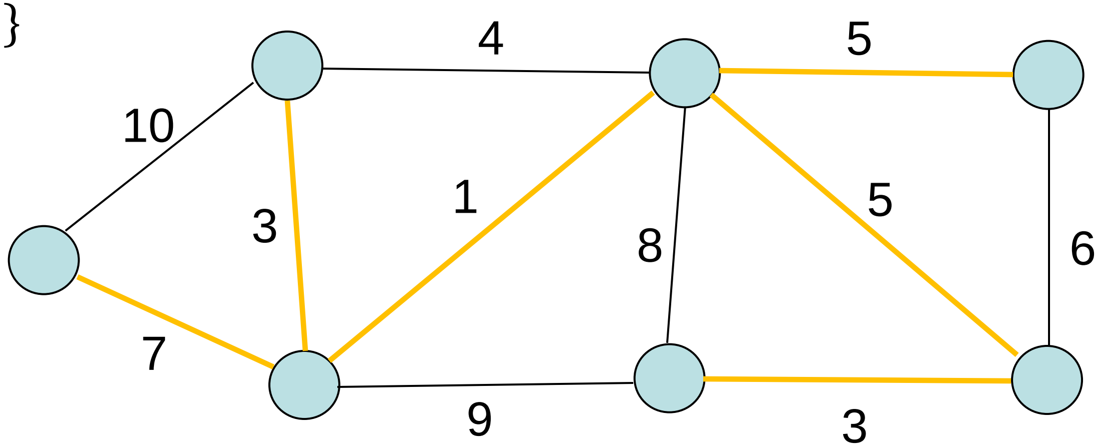
Graphalgorithmen

Prim(G, r)
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v) mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. **return** A



Graphalgorithmen

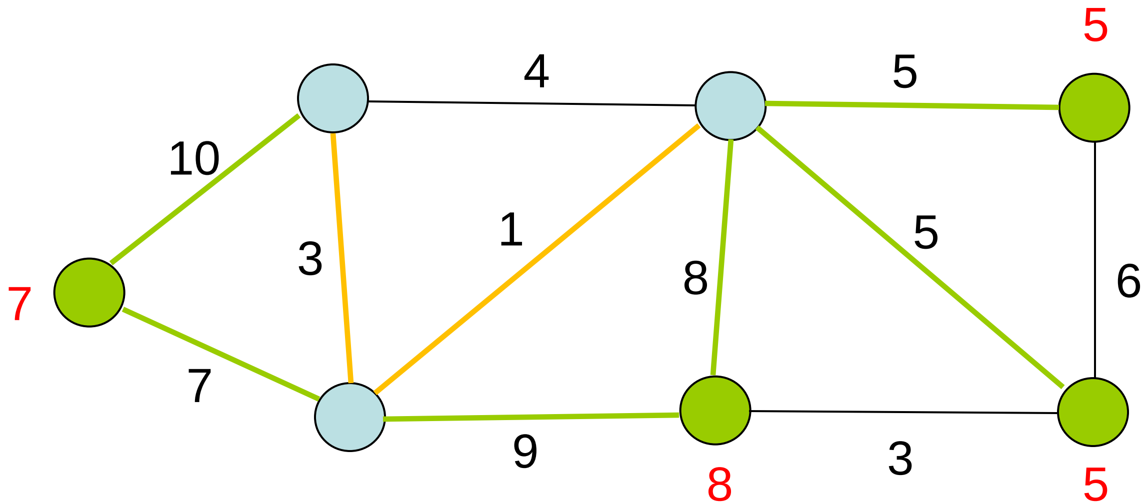
Prim(G, r)
1. $Q \leftarrow V - \{r\}$
2. $A \leftarrow \emptyset$
3. **while** $Q \neq \emptyset$
4. Finde Kante (u, v)mit minimalem Gewicht, die den Schnitt $(Q, V - Q)$ kreuzt, wobei $u \in Q$
5. $Q \leftarrow Q - \{u\}$
6. $A \leftarrow A \cup \{(u, v)\}$
7. $\boxed{\textbf{return } A}$



Graphalgorithmen

Prioritätenschlange

- Alle Knoten, die noch nicht zum Baum gehören, werden in Prioritätenschlange Q abgespeichert
- $\mathrm{key}[v]$: minimales Gewicht einer Kante, die v mit Baum verbindet
- $\mathrm{parent}[v]$: Vorgänger von v im Baum
- Menge A implizit gegeben durch $A = \{(v, \mathrm{parent}[v]) | v \in V - \{r\} - Q\}$



Graphalgorithmen

Prim(G, r)

$Q \leftarrow V$
for each vertex $u \in Q$ **do** key$[u] \leftarrow \infty$
key$[r] \leftarrow 0$, parent$[r] \leftarrow$ **nil**
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Extract-Min(Q)
    for each $v \in$ Adj$[u]$ **do**
        if $v \in Q$ **and** $w(u,v) <$ key$[v]$ **then**
            key$[v] \leftarrow w(u,v)$, parent$[v] \leftarrow u$

Graphalgorithmen

Zu Beginn ist jeder Knoten in der Schlange Q

Prim(G, r)

1. $Q \leftarrow V$
2. **for each** vertex $u \in Q$ **do** key$[u] \leftarrow \infty$
3. key$[r] \leftarrow 0$, parent$[r] \leftarrow$ **nil**
4. **while** $Q \neq \emptyset$ **do**
5. $\quad u \leftarrow$ Extract-Min(Q)
6. $\quad$ **for each** $v \in$ Adj$[u]$ **do**
7. $\quad\quad$ **if** $v \in Q$ **and** $w(u,v) <$ key$[v]$ **then**
8. $\quad\quad\quad$ key$[v] \leftarrow w(u,v)$, parent$[v] \leftarrow u$

Graphalgorithmen

$\text{Prim}(G, r)$

$Q \leftarrow V$

$\textbf{for each}$ vertex $u \in Q$ $\textbf{do}$ $\text{key}[u] \leftarrow \infty$

$\boxed{\text{key}[r] \leftarrow 0, \;\; \text{parent}[r] \leftarrow \textbf{nil}}$

$\textbf{while}$ $Q \neq \emptyset$ $\textbf{do}$

$\quad u \leftarrow \text{Extract-Min}(Q)$

$\quad \textbf{for each}$ $v \in \text{Adj}[u]$ $\textbf{do}$

$\quad\quad \textbf{if}$ $v \in Q$ $\textbf{and}$ $w(u, v) < \text{key}[v]$ $\textbf{then}$

$\quad\quad\quad \text{key}[v] \leftarrow w(u, v), \;\; \text{parent}[v] \leftarrow u$

r bildet die Wurzel des Spannbaums

Graphalgorithmen

$\mathrm{Prim}(G, r)$

$Q \leftarrow V$

for each vertex $u \in Q$ **do** $\mathrm{key}[u] \leftarrow \infty$

$\mathrm{key}[r] \leftarrow 0,\ \ \mathrm{parent}[r] \leftarrow$ **nil**

while $Q \neq \emptyset$ **do**

$\quad \boxed{u \leftarrow \text{Extract-Min}(Q)}$

$\quad$ **for each** $v \in \mathrm{Adj}[u]$ **do**

$\quad\quad$ **if** $v \in Q$ **and** $w(u, v) < \mathrm{key}[v]$ **then**

$\quad\quad\quad \mathrm{key}[v] \leftarrow w(u, v),\ \ \mathrm{parent}[v] \leftarrow u$

u ist inzident zu leichter Kante, die Schnitt $(Q, V - Q)$ kreuzt

Graphalgorithmen

Prim(G, r)

$Q \leftarrow V$
for each vertex $u \in Q$ **do** key$[u] \leftarrow \infty$
key$[r] \leftarrow 0$, parent$[r] \leftarrow$ **nil**
while $Q \neq \emptyset$ **do**
  $u \leftarrow$ Extract-Min(Q)
  for each $v \in$ Adj$[u]$ **do**
    if $v \in Q$ **and** $w(u, v) <$ key$[v]$ **then**
      key$[v] \leftarrow w(u, v)$, parent$[v] \leftarrow u$

Gehört v noch nicht zum Baum und könnte über u billiger erreicht werden als bisher bekannt?

Graphalgorithmen

Prim(G, r)

1. $Q \leftarrow V$
2. **for each** vertex $u \in Q$ **do** $\text{key}[u] \leftarrow \infty$
3. $\text{key}[r] \leftarrow 0$, $\text{parent}[r] \leftarrow$ **nil**
4. **while** $Q \neq \emptyset$ **do**
5. $\quad u \leftarrow \text{Extract-Min}(Q)$
6. $\quad$ **for each** $v \in \text{Adj}[u]$ **do**
7. $\quad\quad$ **if** $v \in Q$ **and** $w(u,v) < \text{key}[v]$ **then**
8. $\quad\quad\quad \boxed{\text{key}[v] \leftarrow w(u,v),\ \text{parent}[v] \leftarrow u}$

Wenn ja, dann ist u neuer Vorgänger von v und $\text{key}[v]$ gibt die neuen Kosten für v an

Graphalgorithmen

Implementierung

- Q als AVL Baum (oder alternativ: Binäre Halde)

Laufzeit

- Initialisierung von Q: $\mathbf{O}(|V|)$
- Ausführung der **while**-Schleife: $\mathbf{O}(|V|)$-mal
- Extract-Min: $\mathbf{O}(\log |V|)$
- Länge aller Adjazenzlisten: $\mathbf{O}(|E|)$
- Test $v \in Q$: $\mathbf{O}(1)$
- Änderung eines Schlüsselwertes: $\mathbf{O}(\log |V|)$
- Gesamtlaufzeit: $\mathbf{O}(|V| \log |V| + |E| \log |V|) = \mathbf{O}(|E| \log |V|)$

Graphalgorithmen

Satz 73

Der Algorithmus von Prim berechnet einen minimalen Spannbaum eines gewichteten, zusammenhängenden, ungerichteten Graphen in $\mathbf{O}(|E| \log |V|)$ Zeit.

Beweis

- Die Laufzeit haben wir bereits analysiert.
- Die Korrektheit folgt aus der Tatsache, dass wir nur sichere Kanten verwenden und dass der Algorithmus einen Baum berechnet

Graphalgorithmen

Graphtraversierung

- Breitensuche, Tiefensuche

Kürzeste Wege

- Breitensuche, Algorithmus von Dijkstra, Algorithmus von Floyd-Warshall

Minimale Spannbäume

- Algorithmen von Kruskal und Prim