2.6 Graphen

2.6.3 Minimal spannende Bäume

2.6.3.1 Generischer Algorithmus

2.6.3.2 Prims Algorithmus

2.6.3.3 Kruskals Algorithmus

Minimal Spannende Bäume

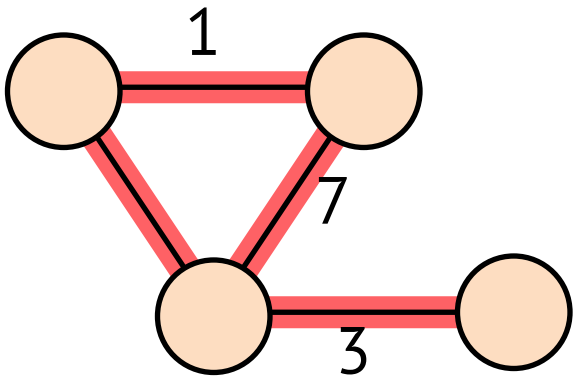
- Gegeben sei ein zusammenhängender, ungerichteter, gewichteter Graph $G=(V,E)$ mit Gewichtsfunktion $w:E \rightarrow R$.
- Gesucht ist ein Spannbaum $S \subseteq E$ der

$$w(S) = \sum_{e \in S} w(e)$$

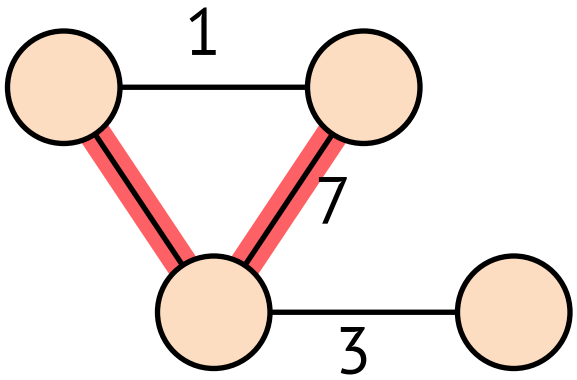
minimiert, der sogenannte MST.

Minimal Spannende Bäume

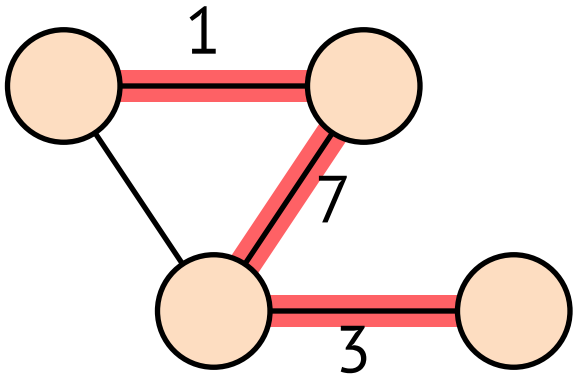
Kein Baum!



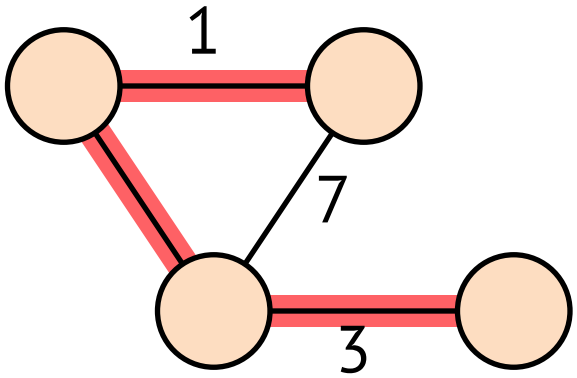
Nicht spannend!



Nicht minimal!



MST



Generischer Algorithmus

- Greedy-Strategie: Füge nach und nach Kanten zu einer anfangs leeren Kantenmenge A hinzu, und zwar unter Beachtung der folgenden Invariante:
  - A ist Teilmenge **eines** MST (*)

Generischer Algorithmus

- Sei A eine Menge von Kanten, die die Invariante (*) erfüllt. Eine Kante e heißt sicher für A, falls $A \cup \{e\}$ ebenfalls die Invariante (*) erfüllt.

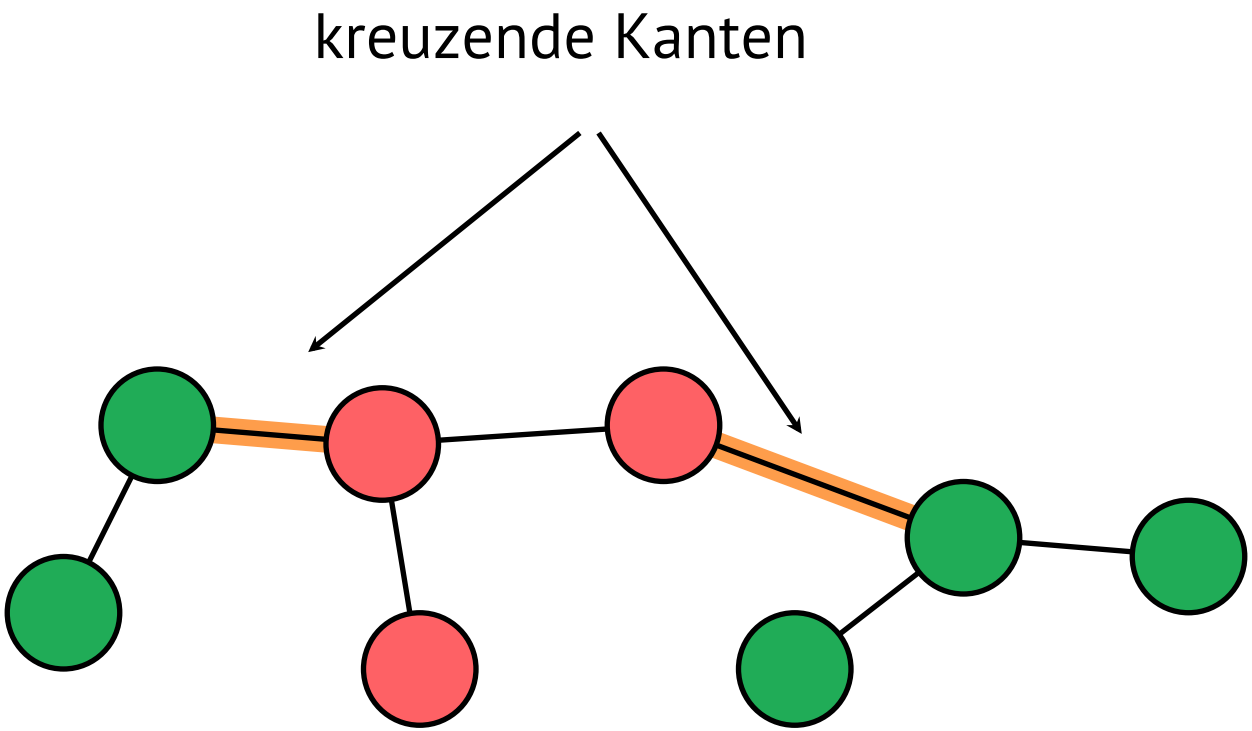
Generischer Algorithmus

- GenericMst(G, w)
  A ← ∅
  while A ist kein MST do
    suche eine sichere Kante e für A
    A ←A ∪{e}
  return A

Generischer Algorithmus

- Ein Schnitt eines ungerichteten Graphen $G=(V,E)$ ist eine Menge $S \subseteq V$
- Eine Kante (u,v) kreuzt den Schnitt S falls einer ihrer Endpunkte in S und der andere Endpunkt in $V-S$ liegt.

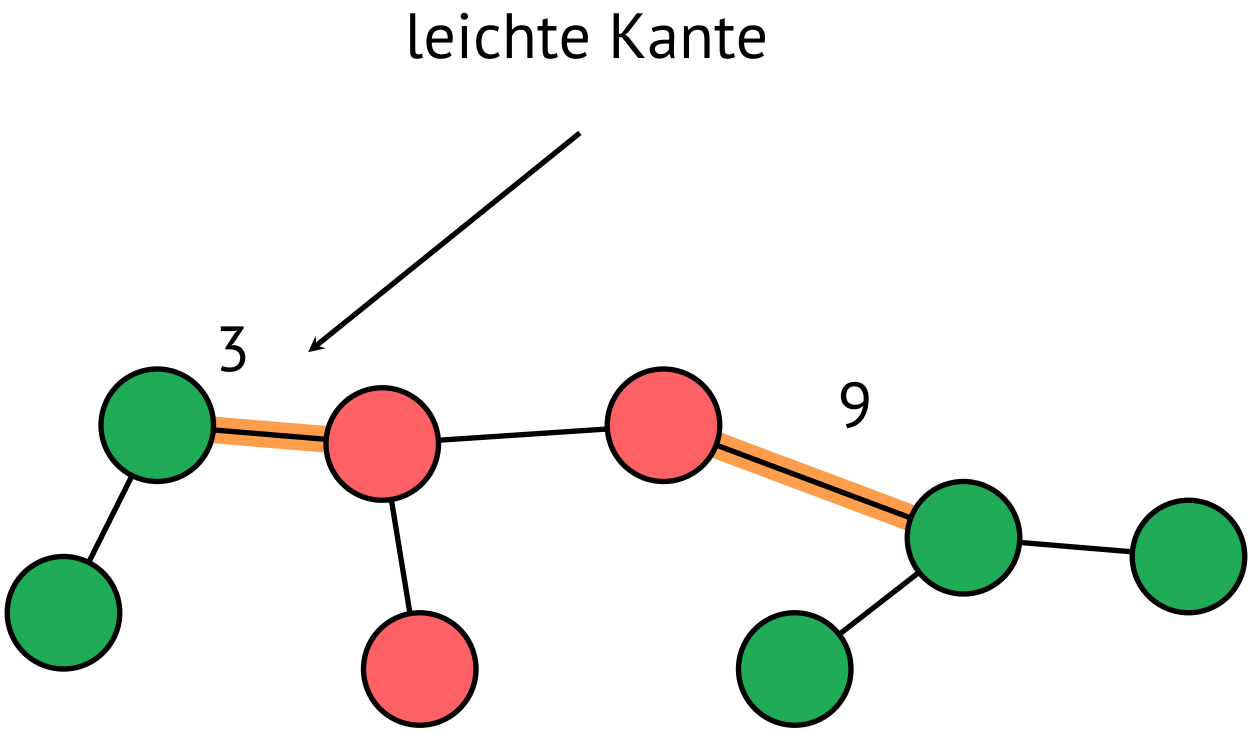
Generischer Algorithmus



Generischer Algorithmus

- Ein Schnitt respektiert A, falls keine Kante aus A den Schnitt kreuzt.
- Eine kreuzende Kante heißt leicht, falls ihr Gewicht minimal unter allen kreuzenden Kanten ist.

Generischer Algorithmus



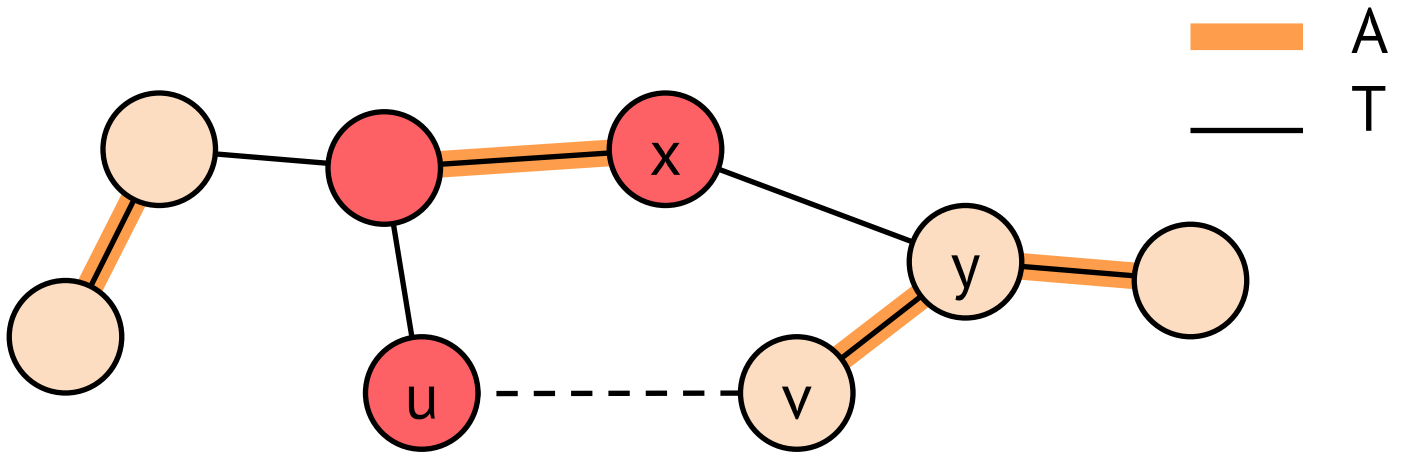
Generischer Algorithmus

- Satz: A erfülle die Invariante (*). S sei ein Schnitt der A respektiert und (u,v) eine leichte Kante, die S kreuzt. Dann ist (u,v) sicher für A.

Generischer Algorithmus

- Sei T ein MST der A enthält, aber nicht (u,v)

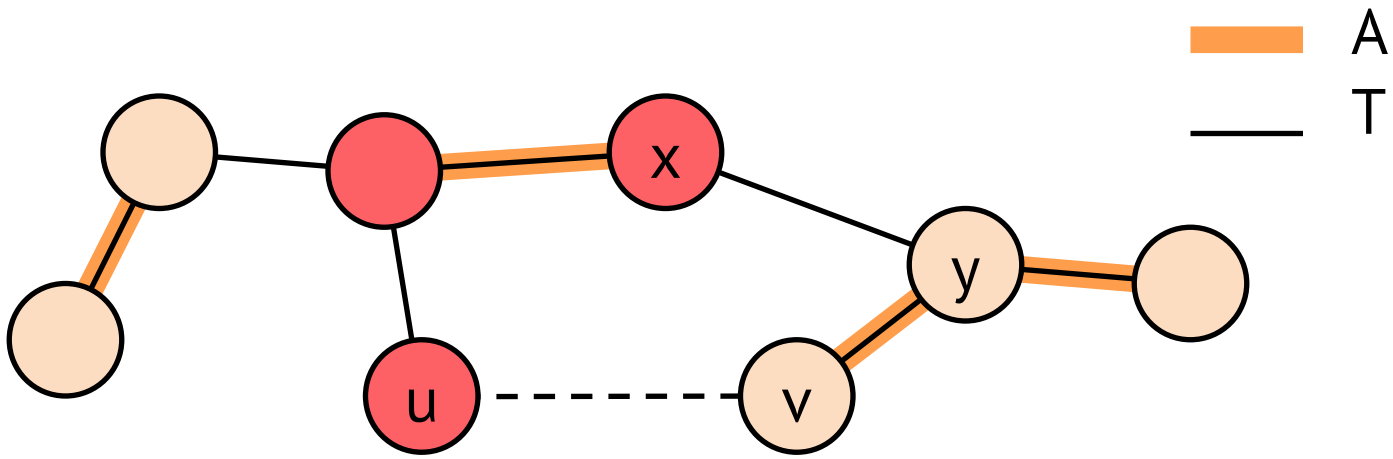
Würde er (u,v) enthalten,
so wären wir schon fertig!



Generischer Algorithmus

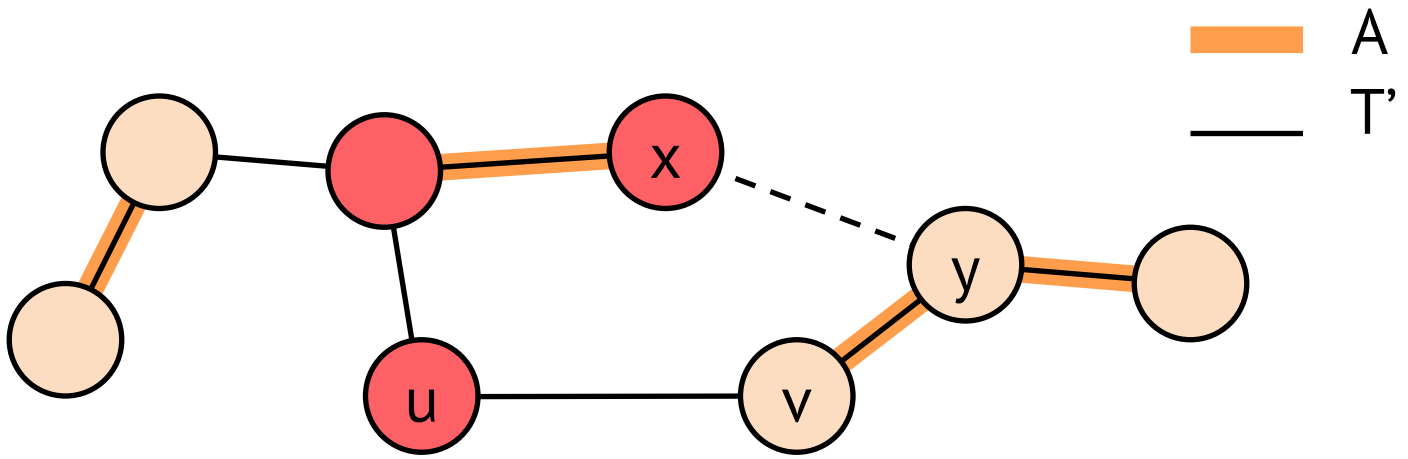
- Sei T ein MST der A enthält, aber nicht (u,v)
- Auf dem Weg von u nach v durch T gibt es eine Kante (x,y) die den Schnitt kreuzt

Der Weg existiert, da T ein spannend ist. Er ist eindeutig, da T ein Baum ist.



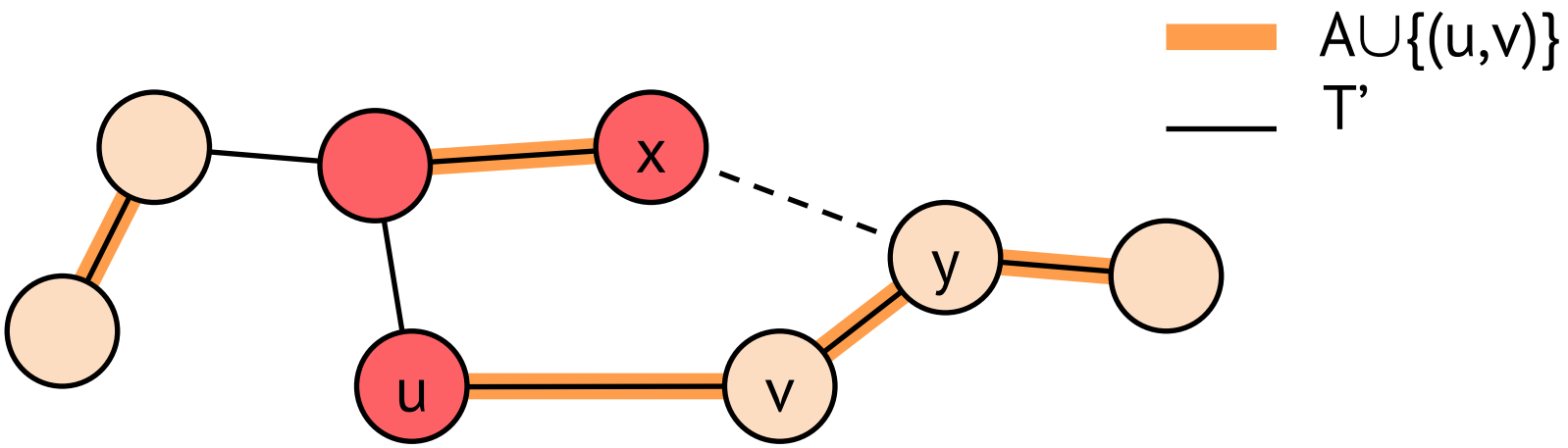
Generischer Algorithmus

- Sei T ein MST der A enthält, aber nicht (u,v)
- Auf dem Weg von u nach v durch T gibt es eine Kante (x,y) die den Schnitt kreuzt
- T' = T − {(x,y)} ∪ {(u,v)} ist ein MST, denn w(T') = w(T) − w(x,y) + w(u,v) ⩽ w(T)



Generischer Algorithmus

- Sei T ein MST der A enthält, aber nicht (u,v)
- Auf dem Weg von u nach v durch T gibt es eine Kante (x,y) die den Schnitt kreuzt
- T' = T − {(x,y)} ∪ {(u,v)} ist ein MST
- Wegen $A \subseteq T$ und $(x,y) \notin A$ folgt $A \cup \{(u,v)\} \subseteq T'$, also ist (u,v) sicher für A



2.6.3 Minimal spannende Bäume

2.6.3.1 Generischer Algorithmus

2.6.3.2 Prims Algorithmus

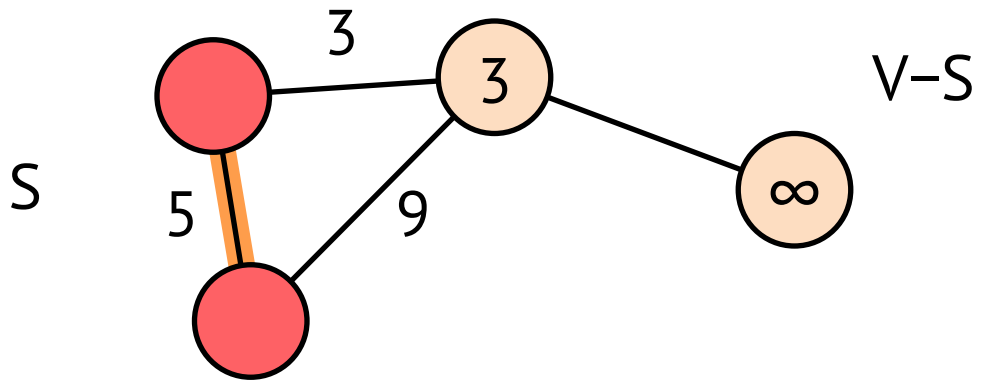
2.6.3.3 Kruskals Algorithmus

Prims Algorithmus

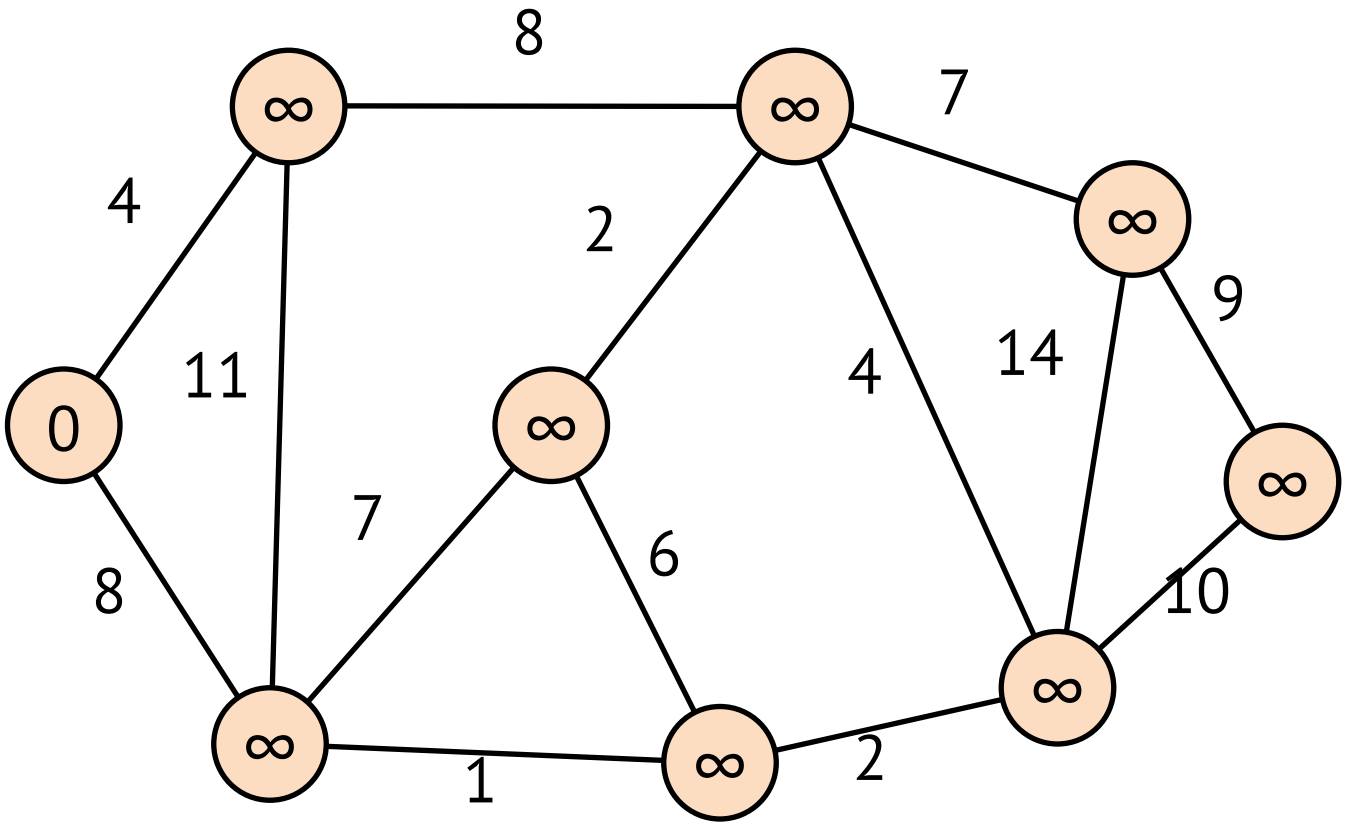
- Die Kanten in A bilden einen Baum. Jeder Knoten u erhält einen Zeiger $p[u]$ auf seinen Vater im MST.
- Der A respektierende Schnitt S ist gegeben durch $S = \{ u : (u,v) \in A \}$

Prims Algorithmus

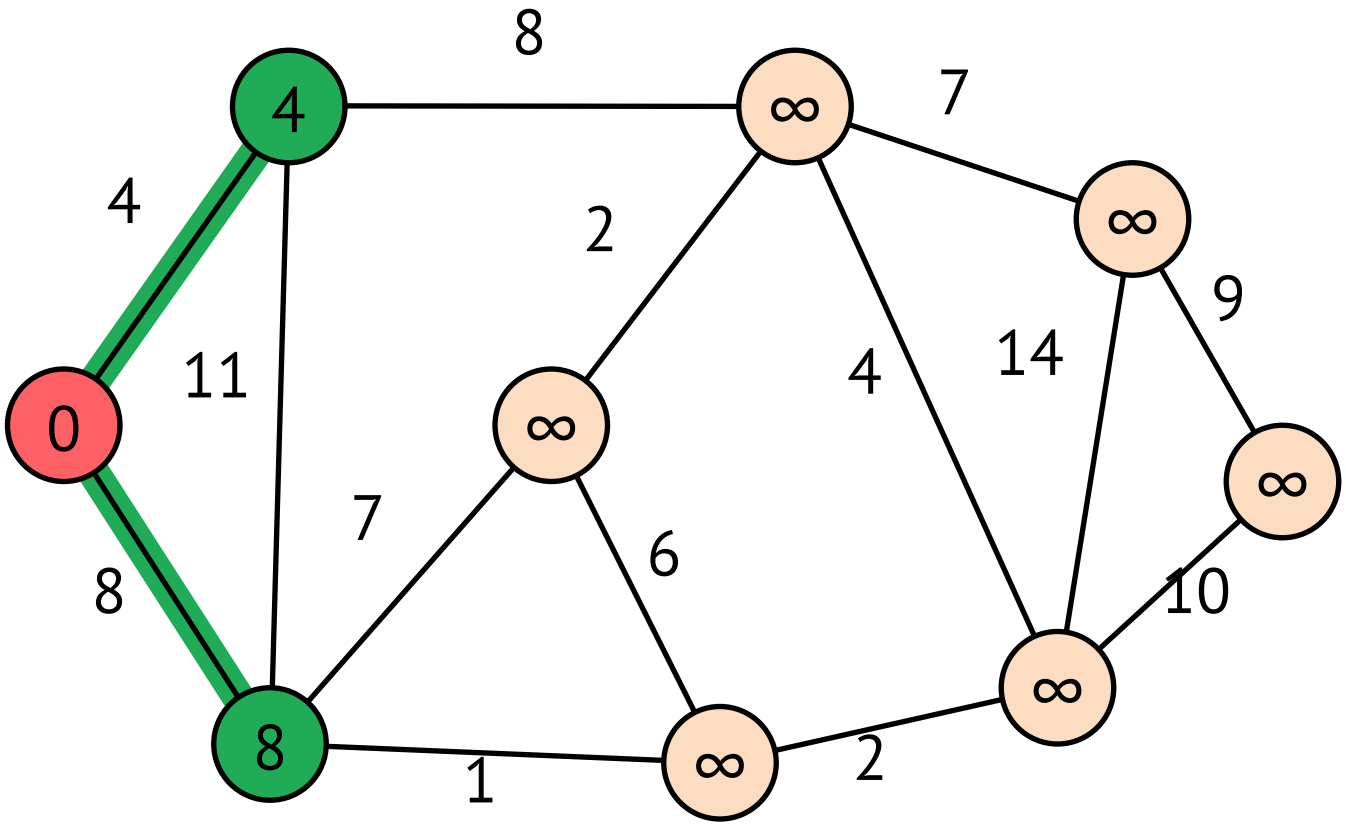
- Knoten u außerhalb von S erhalten einen Schlüssel, der das minimale Gewicht einer Kante angibt, die u mit S verbindet (∞∞ falls es solch eine Kante nicht gibt)



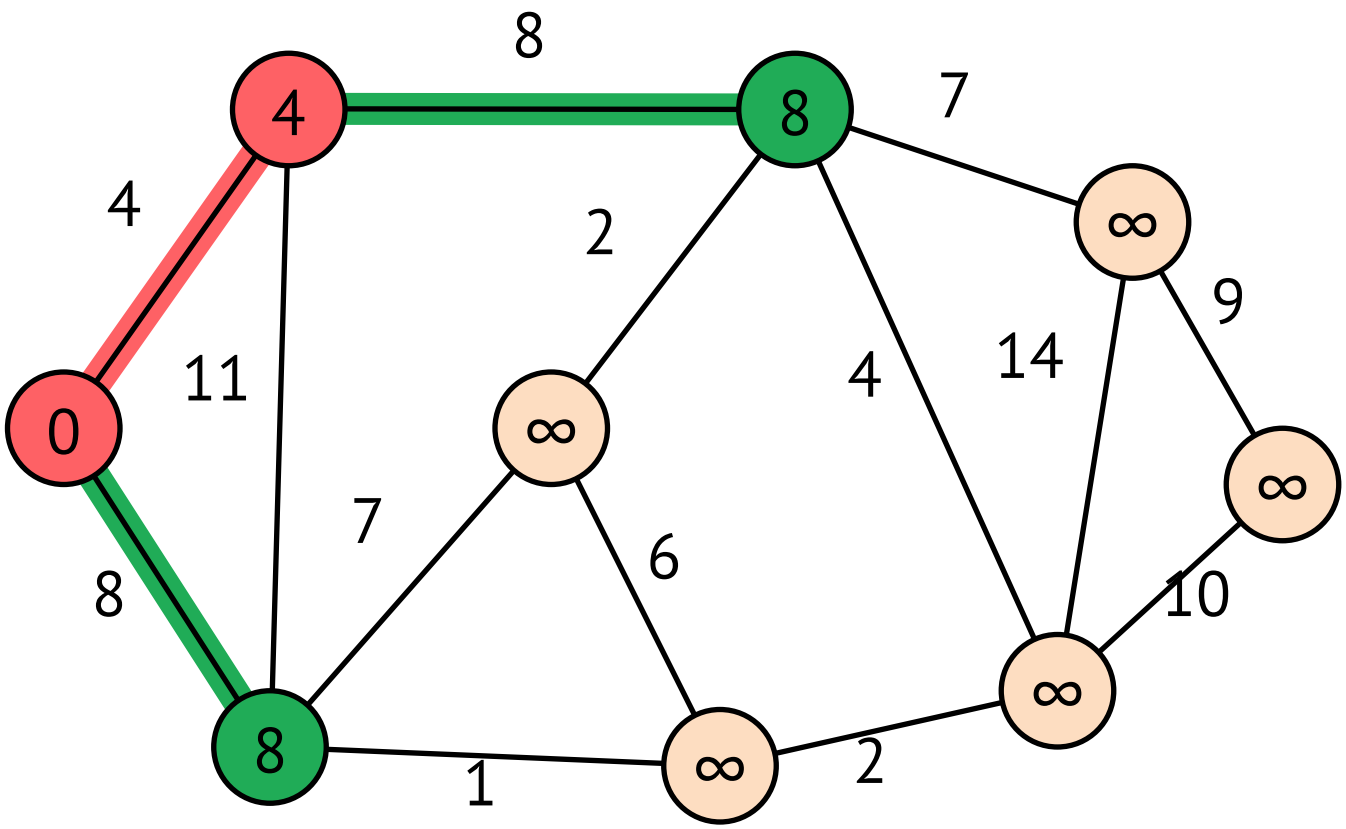
Prims Algorithmus



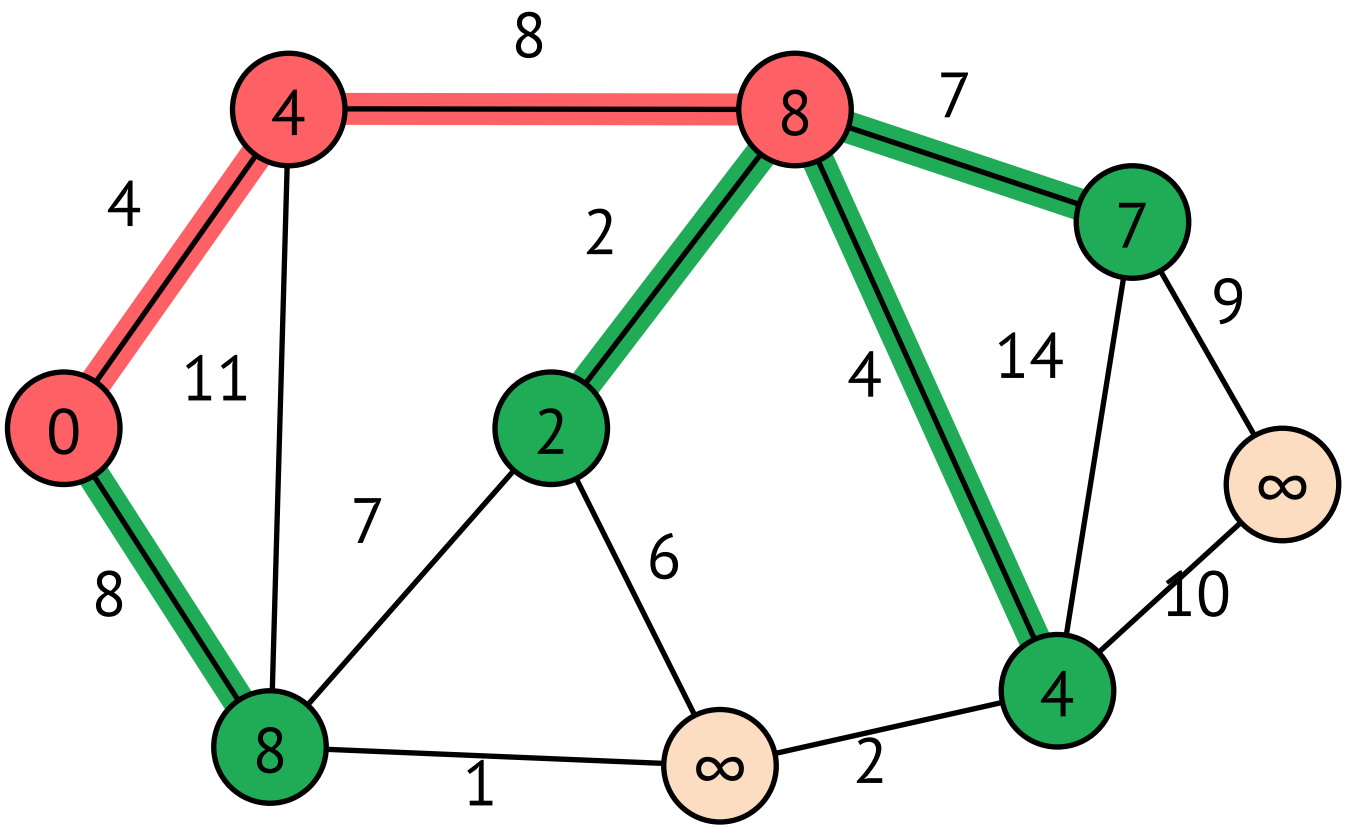
Prims Algorithmus



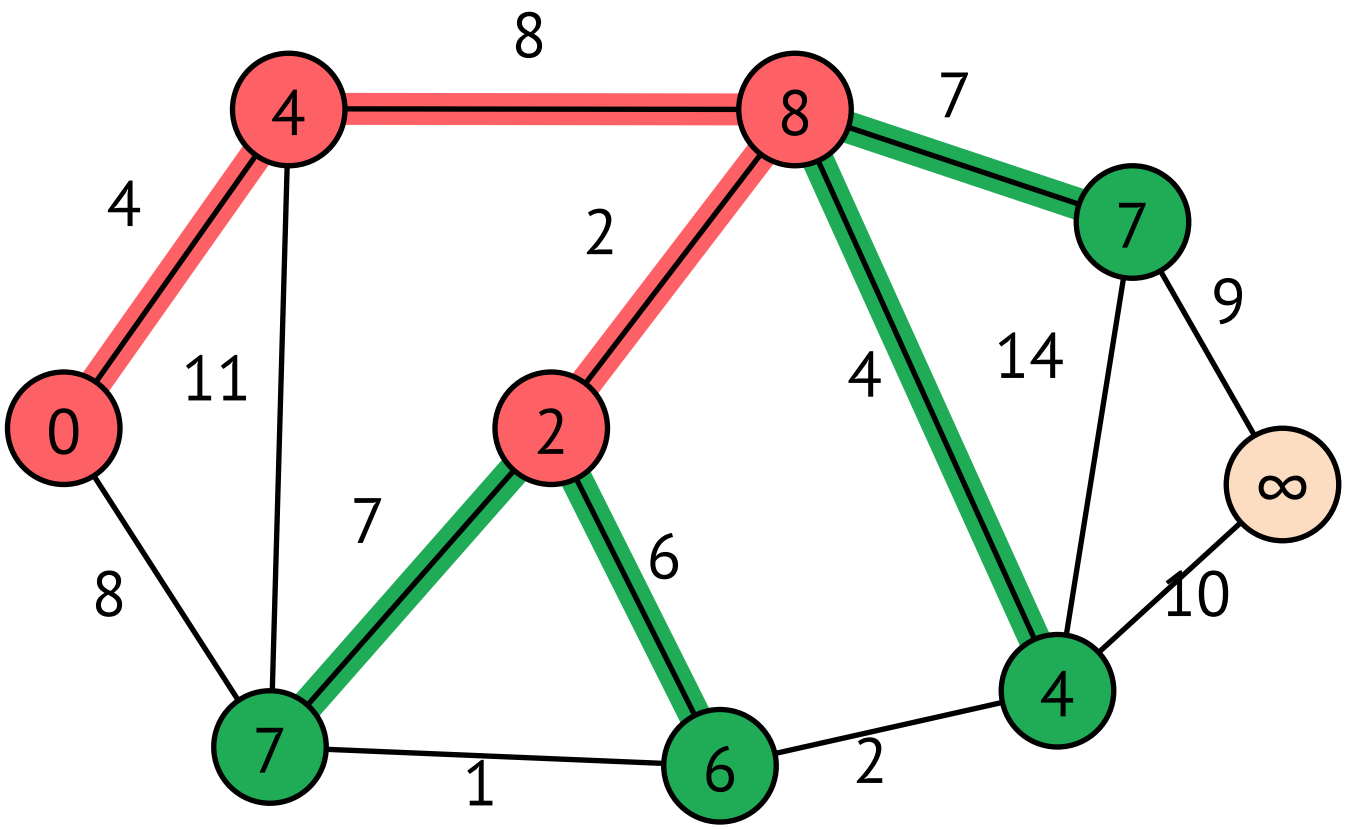
Prims Algorithmus



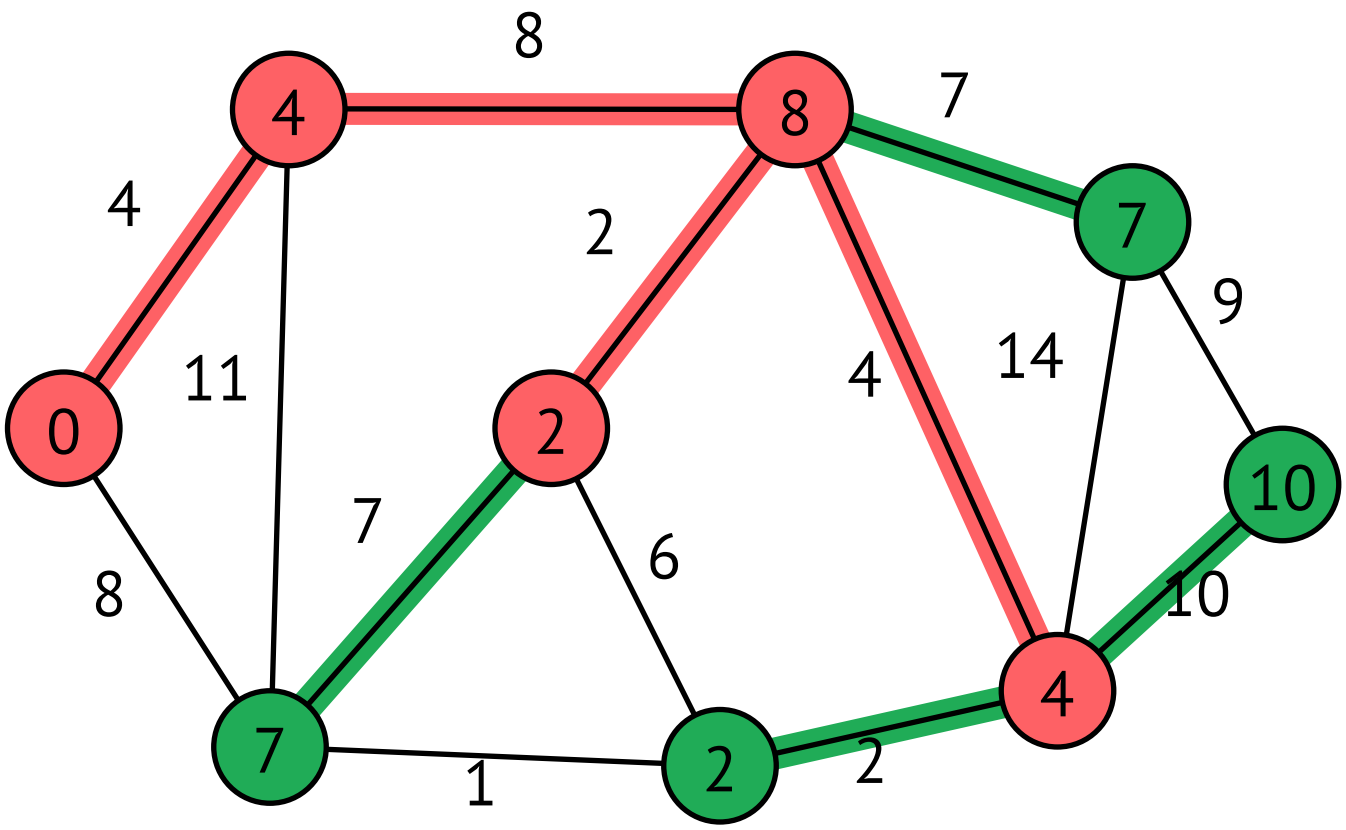
Prims Algorithmus



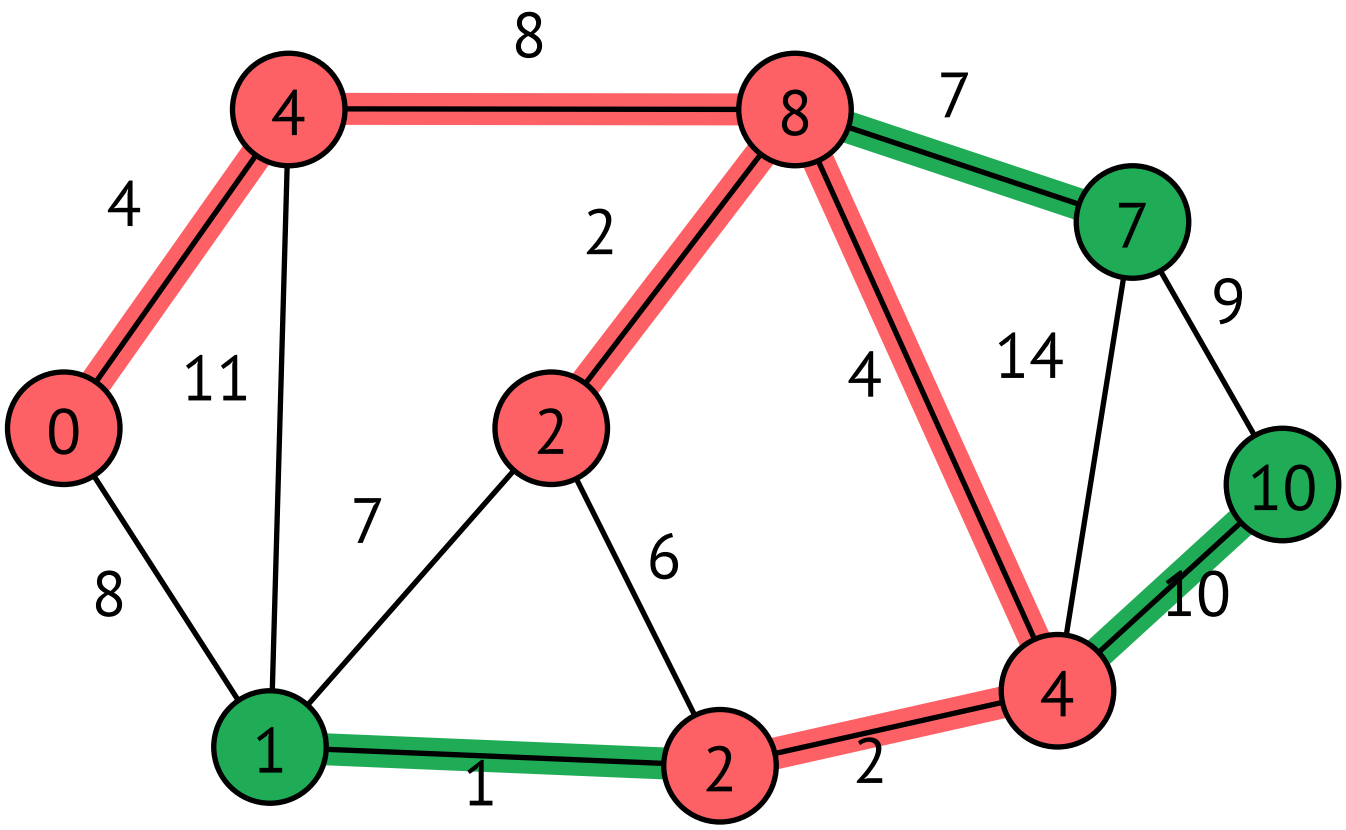
Prims Algorithmus



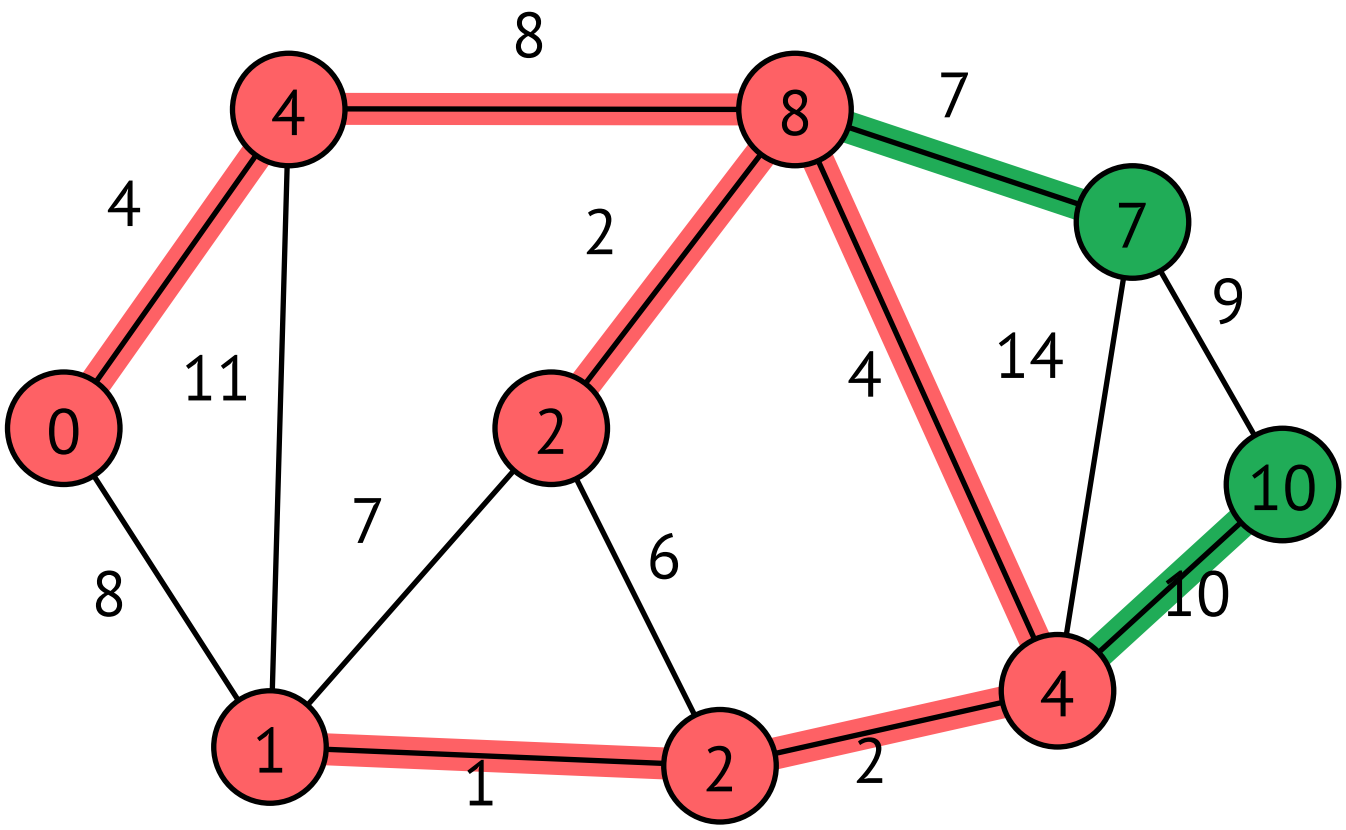
Prims Algorithmus



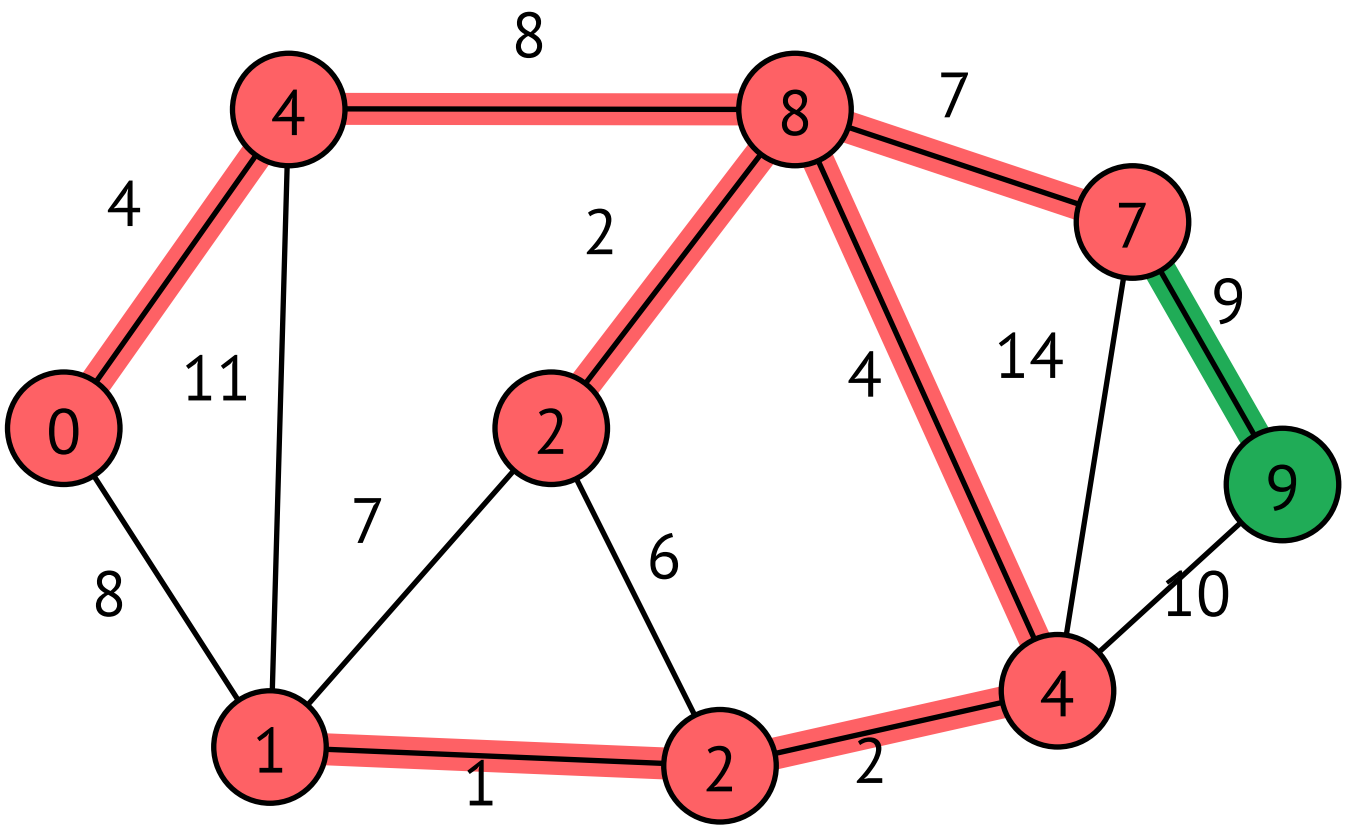
Prims Algorithmus



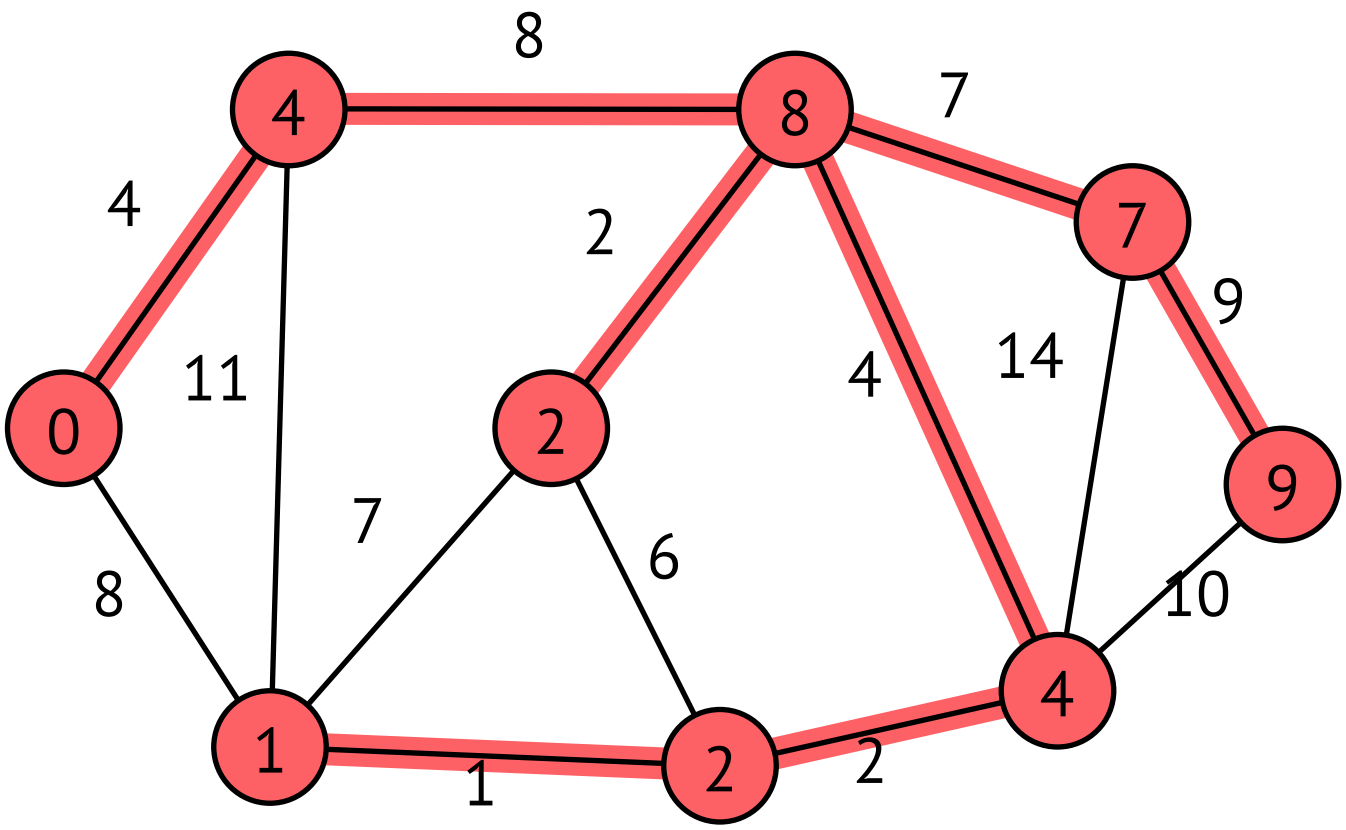
Prims Algorithmus



Prims Algorithmus



Prims Algorithmus



Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do
    key[u] ← ∞
    p[u] ← NIL
key[r] ← 0
Q ← V
while Q not empty do
    u ← ExtractMinimum(Q)
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
            p[v] ← u
            key[v] ← w(u,v)
```

Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do                 ⎤
    key[u] ← ∞                    ⎥  O(V)
    p[u] ← NIL                    ⎦
key[r] ← 0
Q ← V
while Q not empty do
    u ← ExtractMinimum(Q)
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
            p[v] ← u
            key[v] ← w(u,v)
```

Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do
    key[u] ← ∞
    p[u] ← NIL
key[r] ← 0          ┐
Q ← V               ┘   O(V) Heap Aufbau!
while Q not empty do
    u ← ExtractMinimum(Q)
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
            p[v] ← u
            key[v] ← w(u,v)
```

Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do
    key[u] ← ∞
    p[u] ← NIL
key[r] ← 0
Q ← V
while Q not empty do
    u ← ExtractMinimum(Q)
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
            p[v] ← u
            key[v] ← w(u,v)
```

V Läufe

Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do
    key[u] ← ∞
    p[u] ← NIL
key[r] ← 0
Q ← V
while Q not empty do
    u ← ExtractMinimum(Q) ]
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
            p[v] ← u
            key[v] ← w(u,v)
```

Entfernen eines Elements aus dem Heap

O(log V)

Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do
    key[u] ← ∞
    p[u] ← NIL
key[r] ← 0
Q ← V
while Q not empty do                          Durchläufe insgesamt
    u ← ExtractMinimum(Q)
    for each v ∈ Adj[u] do                    ⎫
        if v ∈ Q and w(u,v) < key[v] then     ⎬  O(E)
            p[v] ← u                          ⎪
            key[v] ← w(u,v)                   ⎭
```

Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do
    key[u] ← ∞
    p[u] ← NIL
key[r] ← 0
Q ← V
while Q not empty do
    u ← ExtractMinimum(Q)
    for each v ∈ Adj[u] do
        if v ∈ Q and w(u,v) < key[v] then
            p[v] ← u
            key[v] ← w(u,v)
```

Heap muss aktualisiert werden

] O(log V)

Prims Algorithmus

- PRIM(G,w,r)

```
for each u ∈ V do                                  ┐
    key[u] ← ∞                                     │
    p[u] ← NIL                                     │ O(V)
key[r] ← 0                                         │
Q ← V                                              ┘
while Q not empty do                               ┐
    u ← ExtractMinimum(Q)                          │
    for each v ∈ Adj[u] do                         │ O(V×log V
        if v ∈ Q and w(u,v) < key[v] then          │  + E×log V)
            p[v] ← u                               │
            key[v] ← w(u,v)                        ┘
```

Prims Algorithmus

- Nach der Ausführung von Prim(G,w,r) ist der MST gegeben durch
  $A = \{ (u,p[u]) : u \in V - \{r\} \}$
- Implementierung von Prims Algorithmus z.B. mit binärem Heap

Prims Algorithmus

- Implementiert man Prims Algorithmus mittels eines binären Heaps, so ist der Aufwand

$$O(E \times \log V)$$

- Implementiert man Prims Algorithmus mittels eines Fibonacci-Heaps, so ist der Aufwand

$$O(E + V \times \log V)$$

2.6.3 Minimal spannende Bäume

2.6.3.1 Generischer Algorithmus

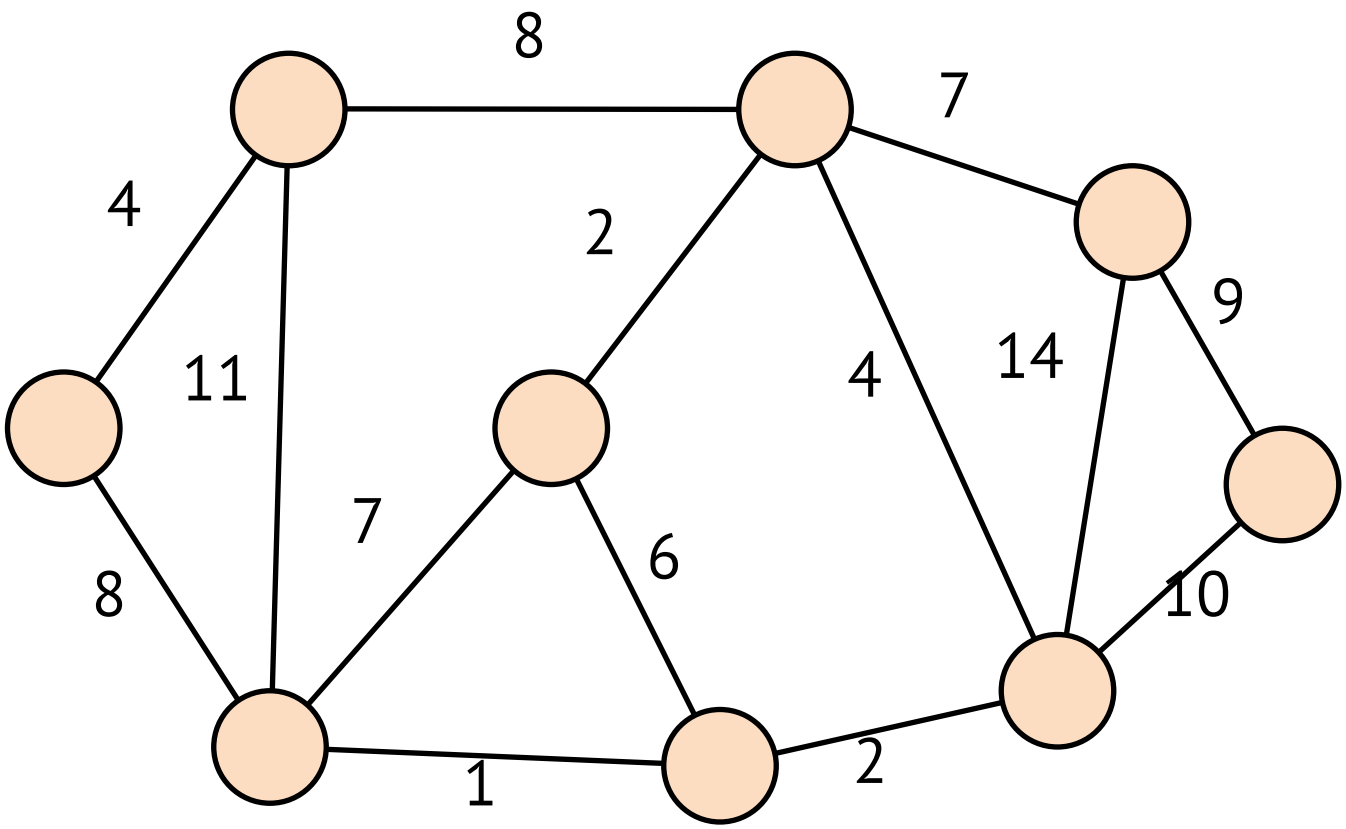
2.6.3.2 Prims Algorithmus

2.6.3.3 Kruskals Algorithmus

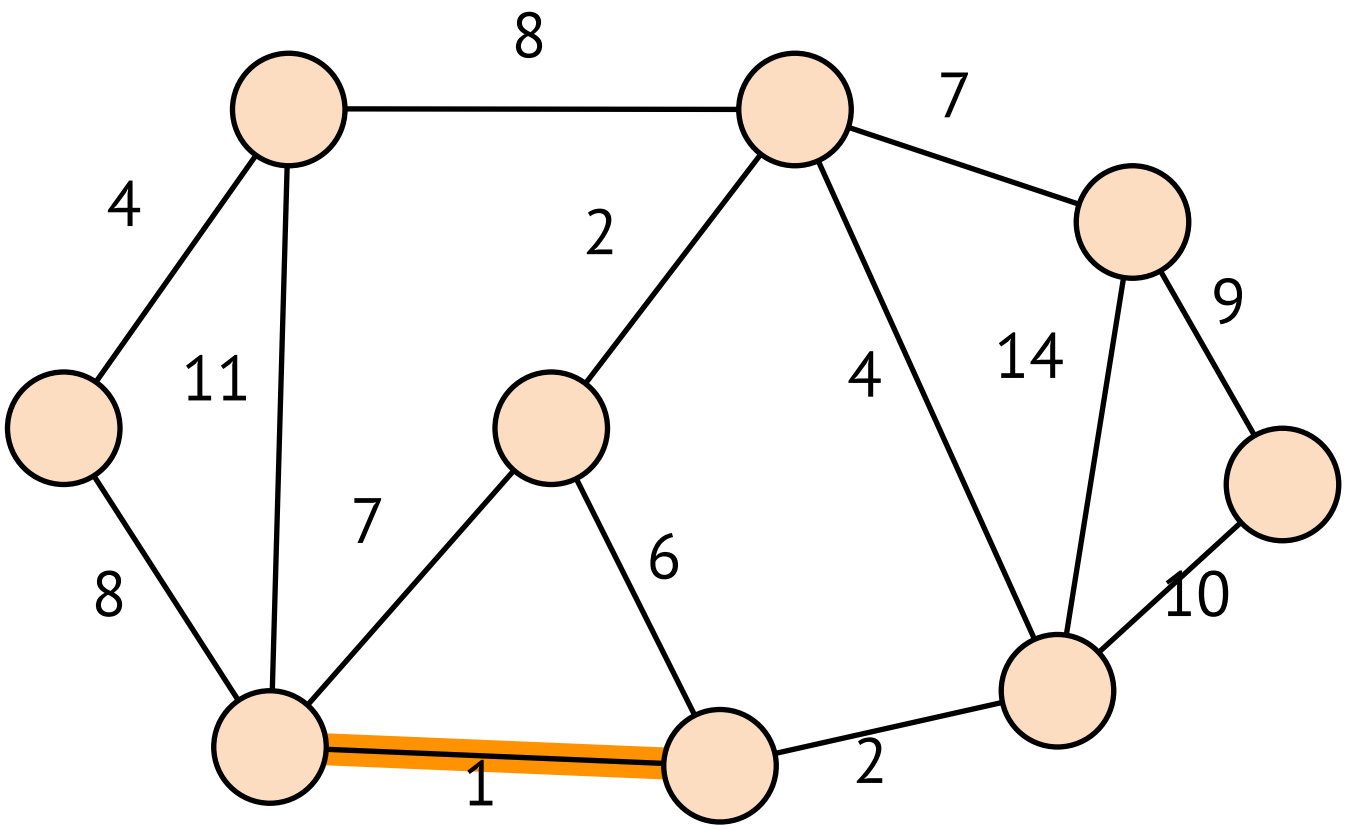
Kruskals Algorithmus

- Die Kanten in A bilden einen Wald. Jeder Knoten u in A erhält einen Zeiger p[u] auf seinen Vater im MST.
- Der A respektierende Schnitt S ist gegeben durch $S = \{ u : (u,v) \in A \}$
- In jedem Schritt wird die Kante mit kleinstem Gewicht gesucht, die zwei Bäume aus A verbindet. Diese Kante wird zu A hinzugefügt.

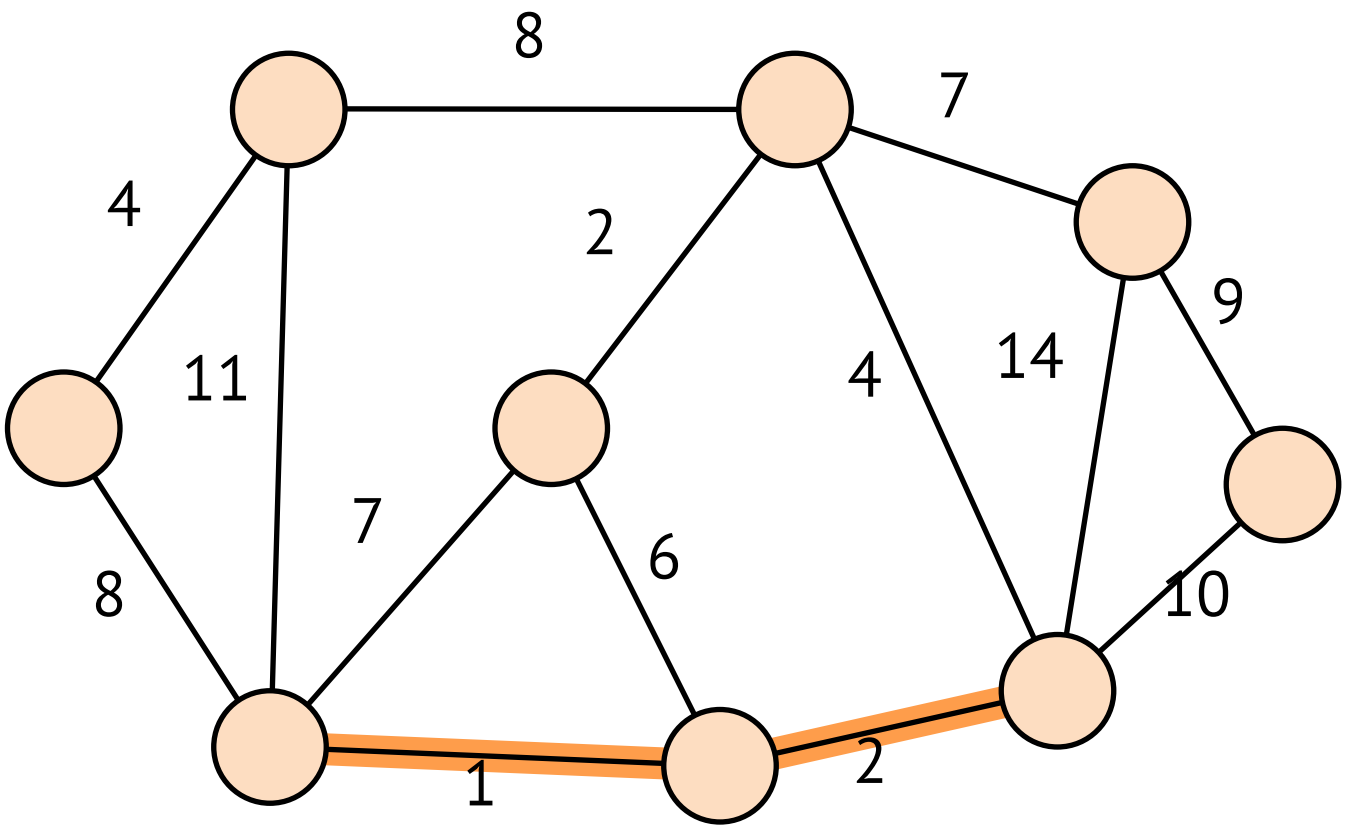
Kruskals Algorithmus



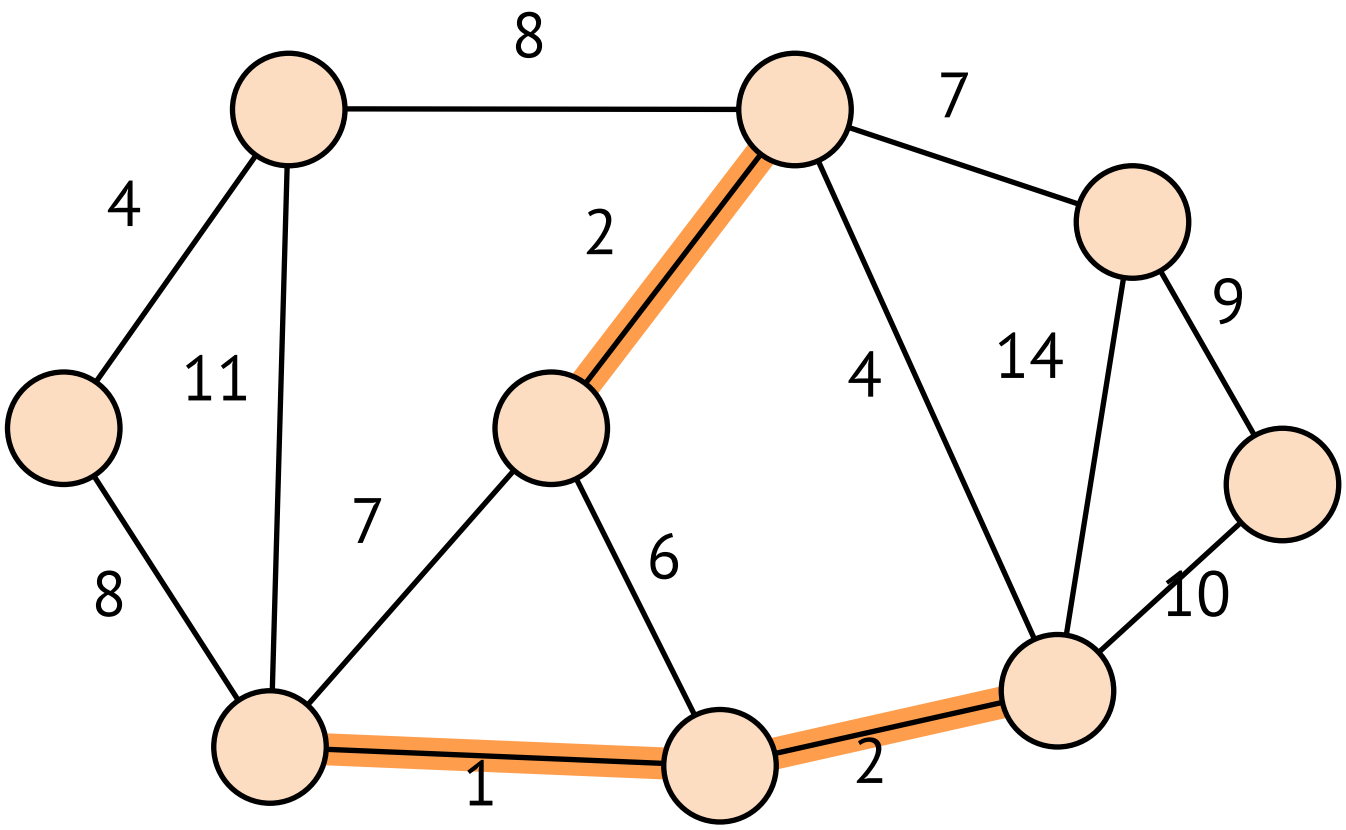
Kruskals Algorithmus



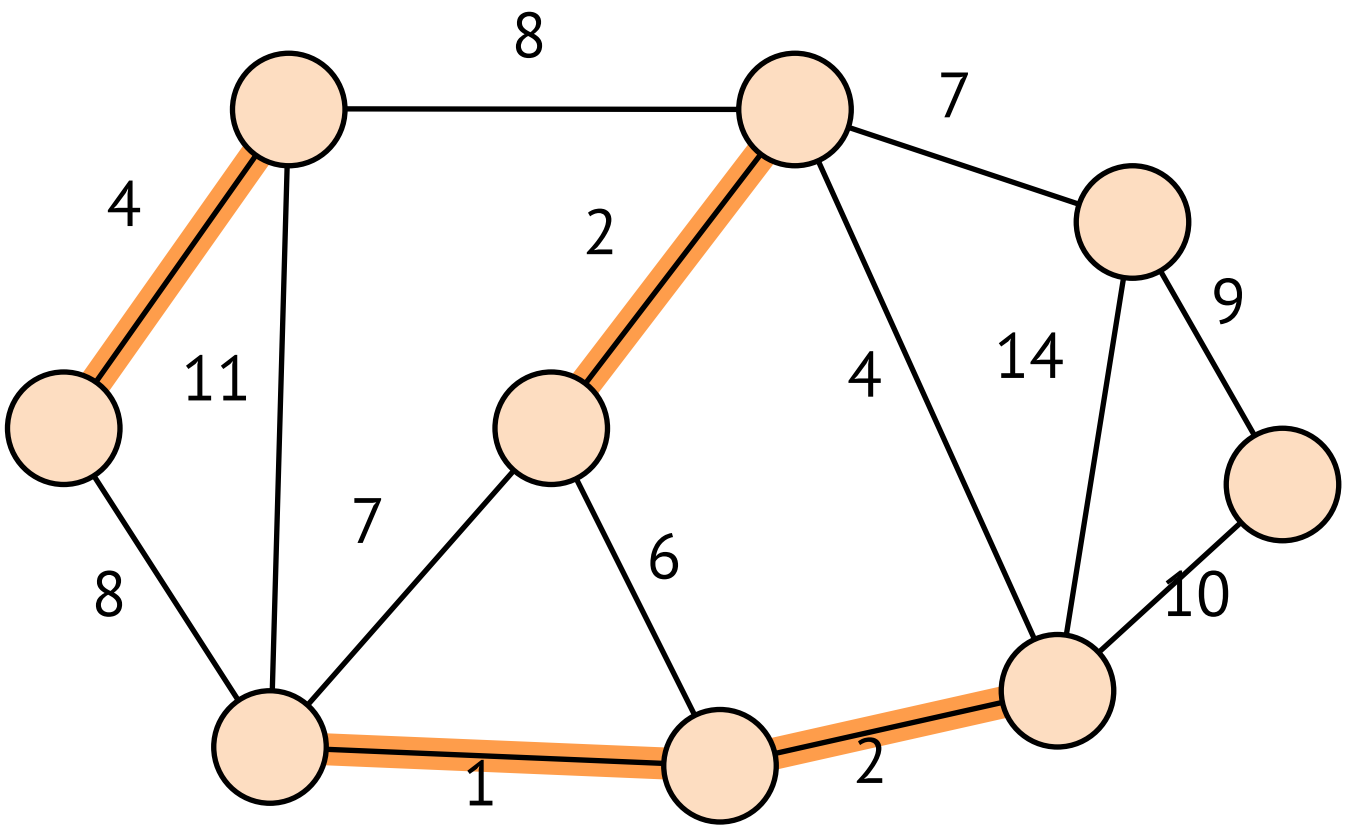
Kruskals Algorithmus



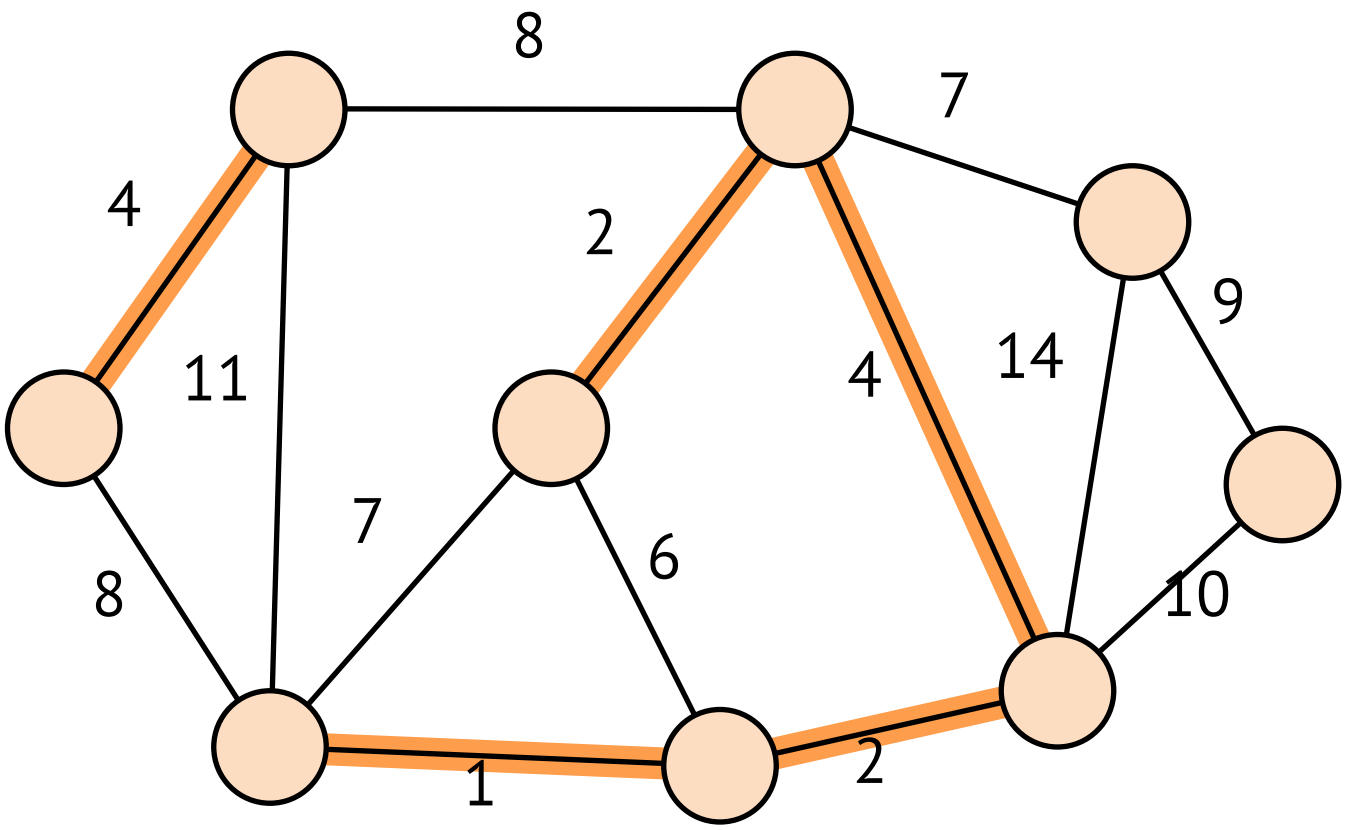
Kruskals Algorithmus



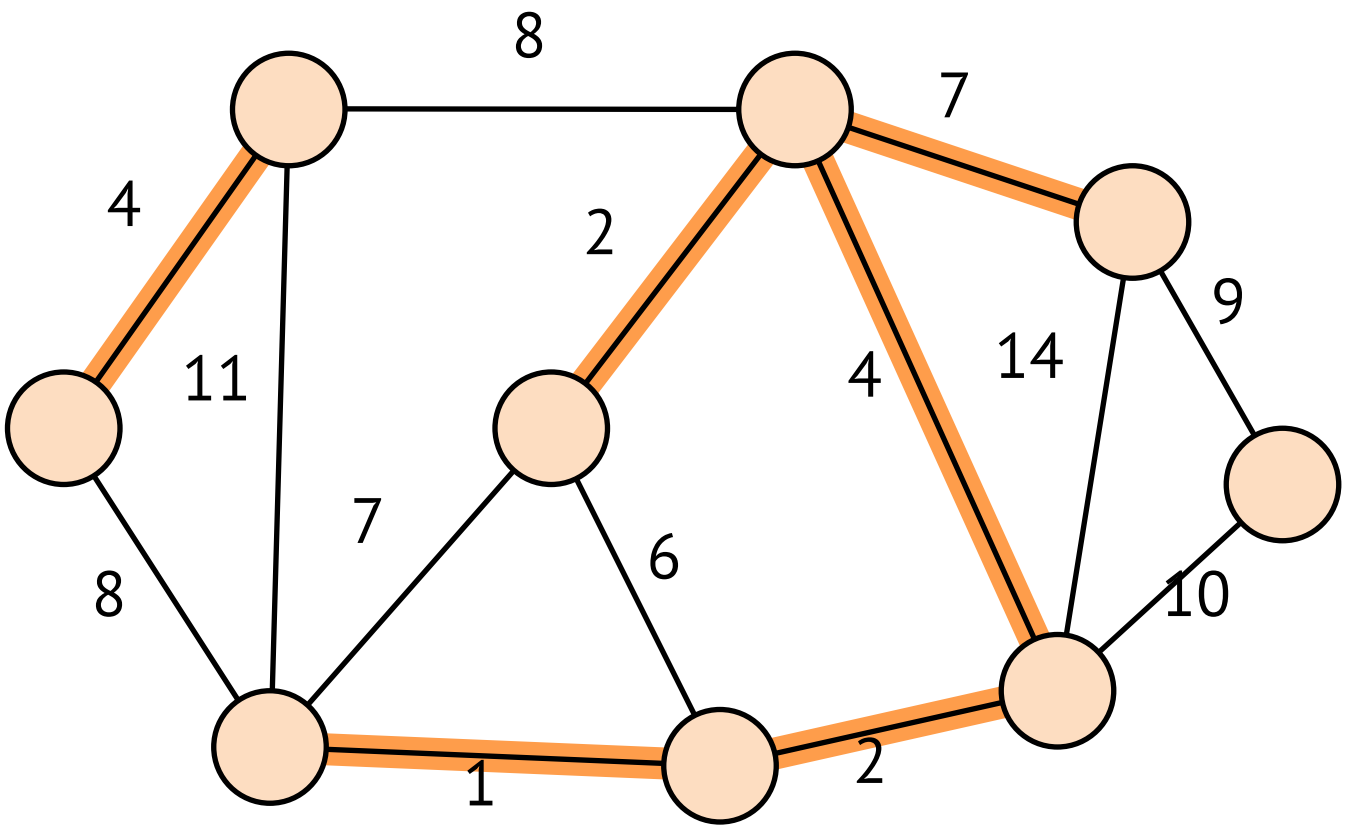
Kruskals Algorithmus



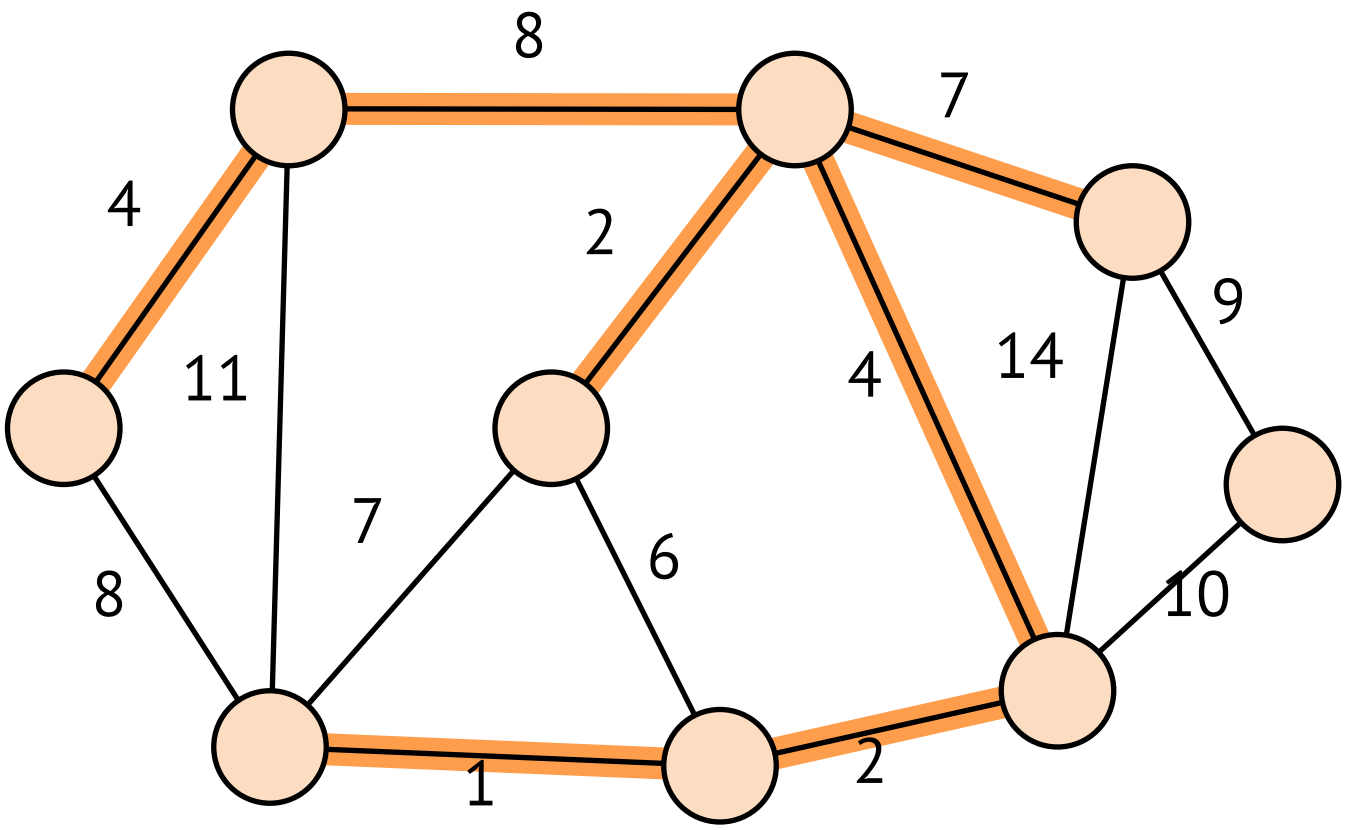
Kruskals Algorithmus



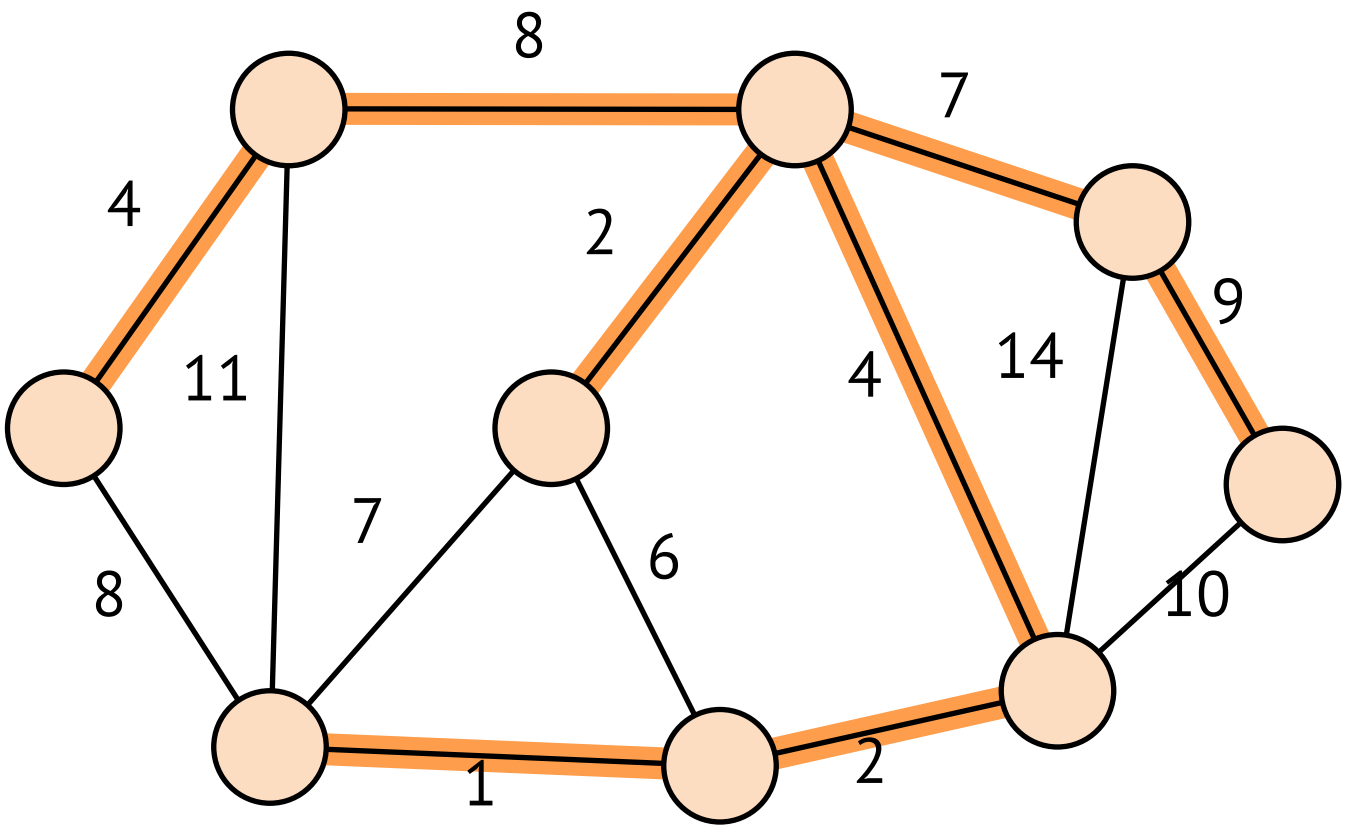
Kruskals Algorithmus



Kruskals Algorithmus



Kruskals Algorithmus



Kruskals Algorithmus

- Kruskal(G,w)

```
A ← ∅
for all v ∈ V do
    MakeSet(v)
sort E into non-decreasing order
for each (u,v) ∈ E do
    if Find(u) ≠ Find(v) then
        A ← A ∪ {(u,v)}
        Union(u,v)
return A
```

Kruskals Algorithmus

- Kruskal(G,w)

```
A ← ∅
for all v ∈ V do
    MakeSet(v)
sort E into non-decreasing order
for each (u,v) ∈ E do
    if Find(u) ≠ Find(v) then
        A ← A ∪ {(u,v)}
        Union(u,v)
return A
```

V × MakeSet()

O(E×log E)

O(E) × Find(), Union()

Kruskals Algorithmus

$E < V^2 \rightarrow \log E = O(\log V)$

- Aufwand
  - Sortieren: $O(E \times \log E) = O(E \times \log V)$
  - Union-Find
    - V × MakeSet()
    - $O(E)$ × FindSet(), Union()

    ➡$O((V+E) \times a(V)) = O(E \times \log V)$

  ➡Gesamtaufwand: $O(E \times \log V)$

Kruskals Algorithmus

- Aufwand
  - Sortieren: O(E log E) = O(E log V)
  - Union-Find
    - V × MakeSet()
    - O(E) × FindSet(), Union()
    - ➡O((V+E)×a(V)) = O(E×logV)
  - ➡Gesamtaufwand: O(E×log V)

E > V−1 und a(V) = O(log V)

Aufwand für Union-Find

Kruskals Algorithmus

- Aufwand
  - Sortieren: O(E×log E) = O(E×log V)
  - Union-Find
    - V × MakeSet()
    - O(E) × FindSet(), Union()

    ➡O((V+E)×a(V)) = O(E×logV)

  ➡Gesamtaufwand: O(E×log V)