Algorithmentheorie

12 – Spannende Bäume minimalen Gewichts

Prof. Dr. S. Albers

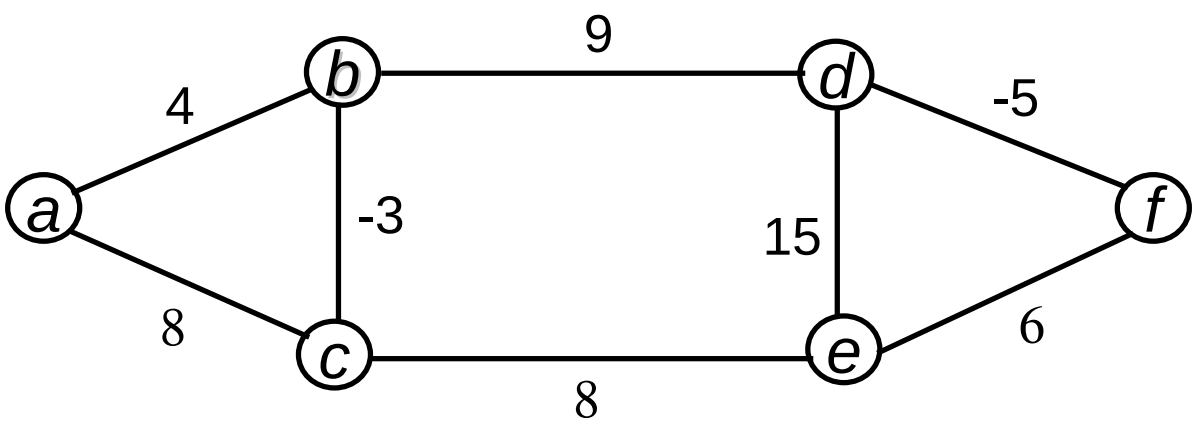
1. Minimale spannende Bäume

Ungerichteter Graph $G = (V, E)$ Kostenfunktion $c: E \rightarrow R$

Sei $T \subseteq E$ ein Baum (zusammenhängender azyklischer Teilgraph)

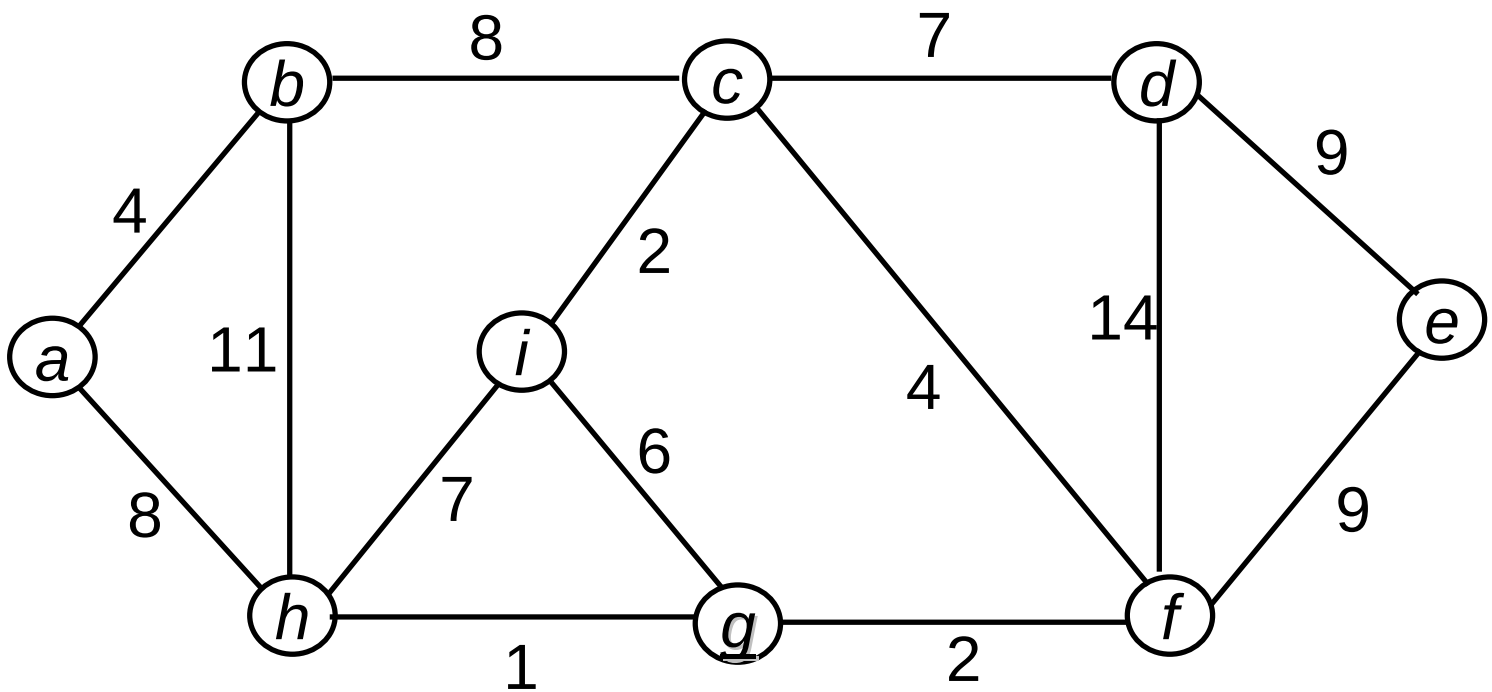
Kosten von T :

$$c(T) = \sum_{(u,v) \in T} c(u,v)$$



Minimale spannende Bäume

Ein minimaler spannender Baum ist ein Baum $T \subseteq E$, der alle Knoten in V verbindet und minimale Kosten hat.



Das Wachsen von min. spannenden Bäumen

Invariante: Verwalte Menge $A \subseteq E$, die Teilmenge eines mininalen spannenden Baums ist.

Definition: Eine Kante $(u,v) \in E \setminus A$ ist sicher für A, wenn

$A \cup \{(u,v)\}$ Teilmenge eines minimalen spannenden Baums ist.

Greedy-Ansatz

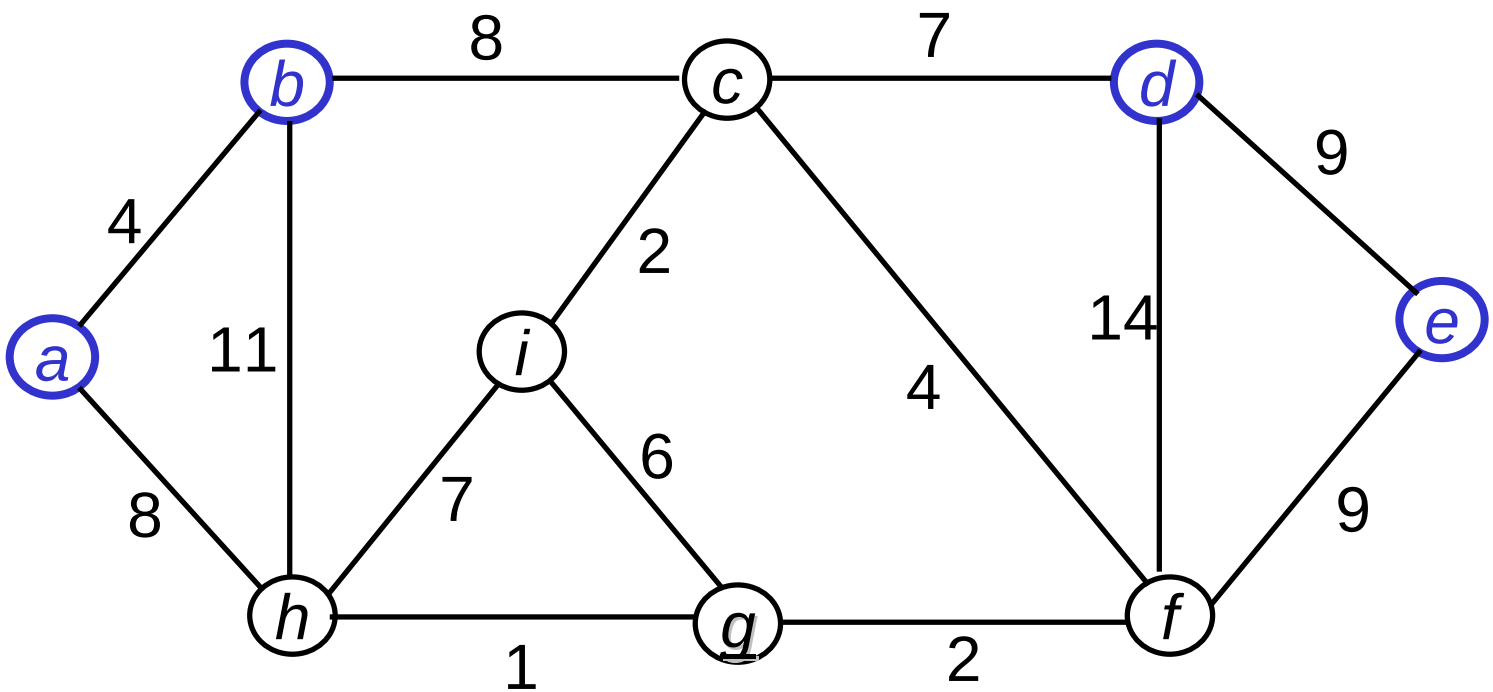
Algorithmus Generischer-MST(G,c);

1. $A \leftarrow \emptyset$;
2. **while** A ist kein spannender Baum **do**
3. Finde sichere Kante (u,v) für A;
4. $A \leftarrow A \cup \{(u,v)\}$;
5. **endwhile;**

2. Schnitte

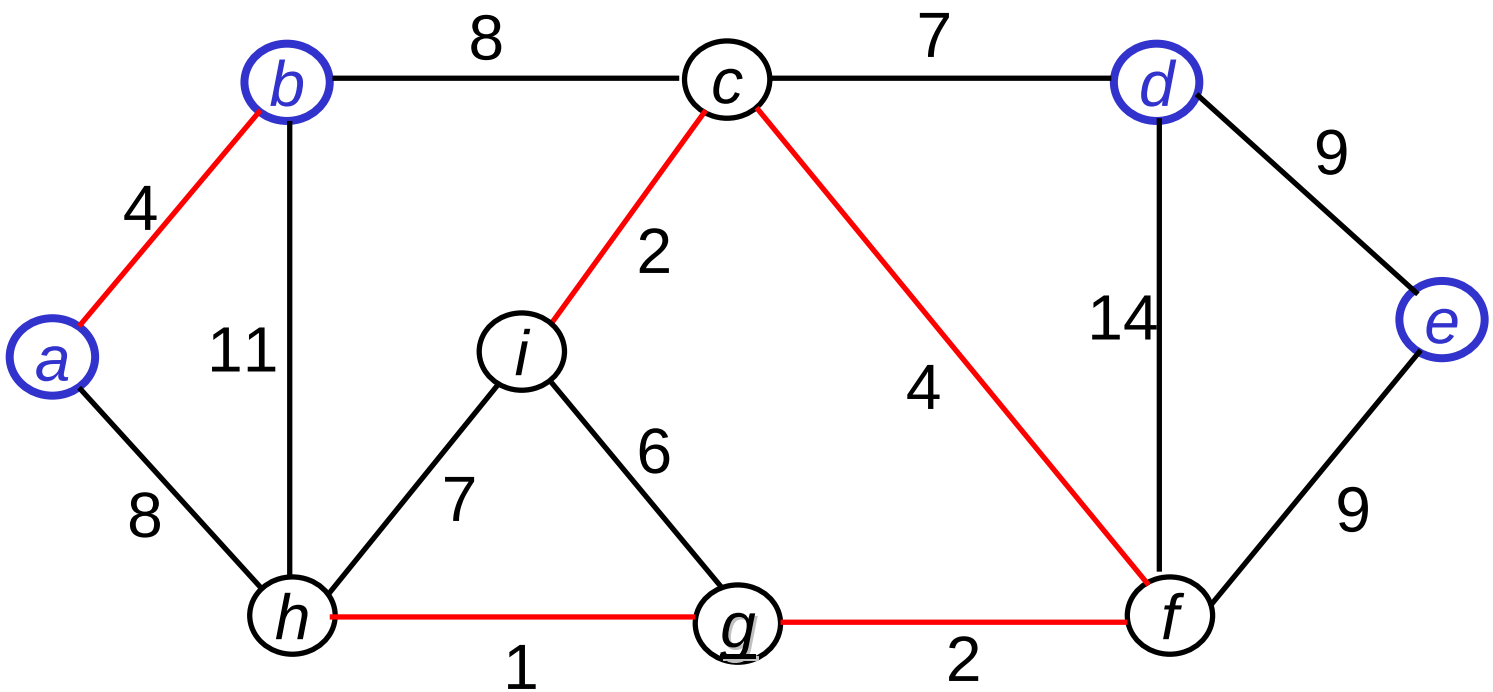
Ein Schnitt $(S, V \setminus S)$ ist eine Partition von V.

Eine Kante (u,v) schneidet $(S, V \setminus S)$, wenn ein Endpunkt in S und der andere in $V \setminus S$ liegt.



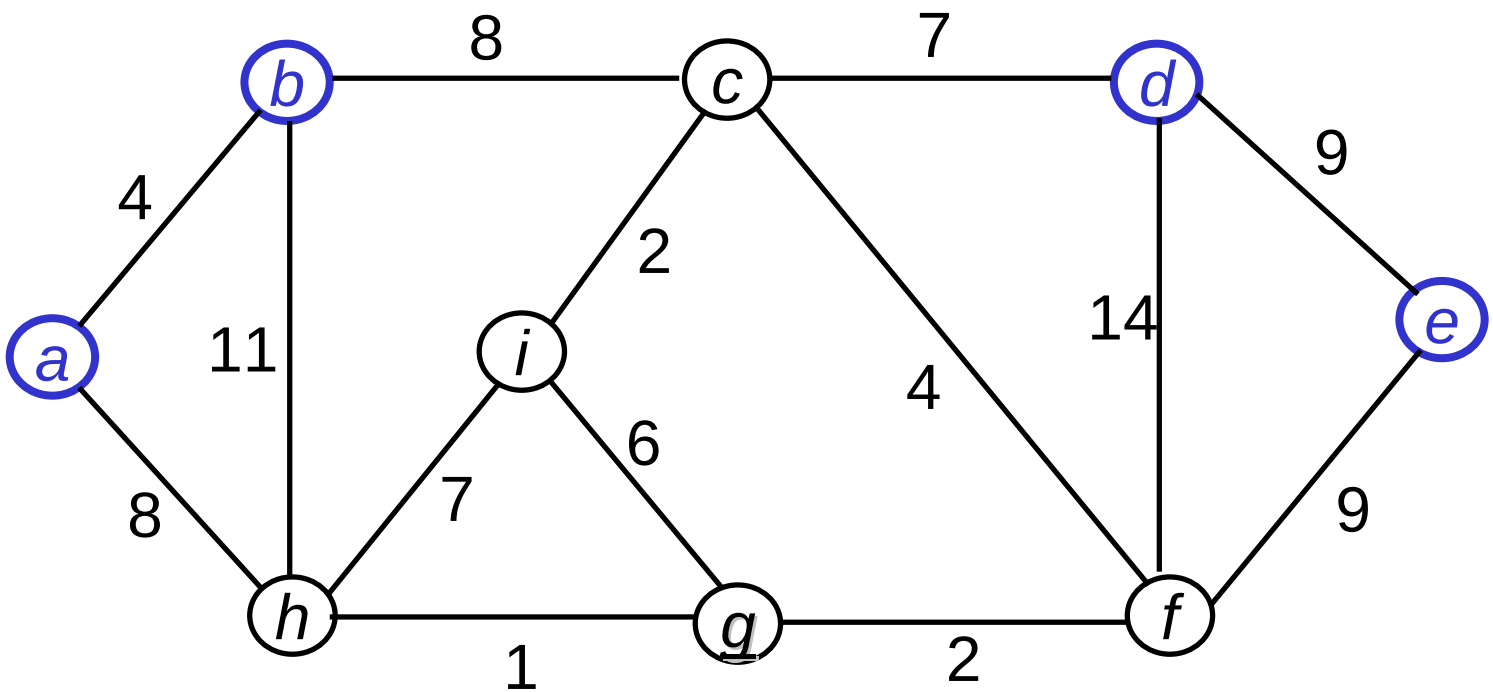
Schnitte

Ein Schnitt respektiert *A*, wenn keine Kante aus *A* den Schnitt schneidet.



Schnitte

Eine Kante heißt minimal bzgl. eines Schnitts, wenn sie den Schnitt schneidet und unter allen solchen Kanten minimale Kosten hat.



3. Sichere Kanten

Satz: Sei A Teilmenge eines min. spannenden Baums T und $(S, V \setminus S)$ ein Schnitt, der A respektiert. Ist (u,v) eine minimale Kante bzgl. $(S, V \setminus S)$, dann ist (u,v) sicher für A.

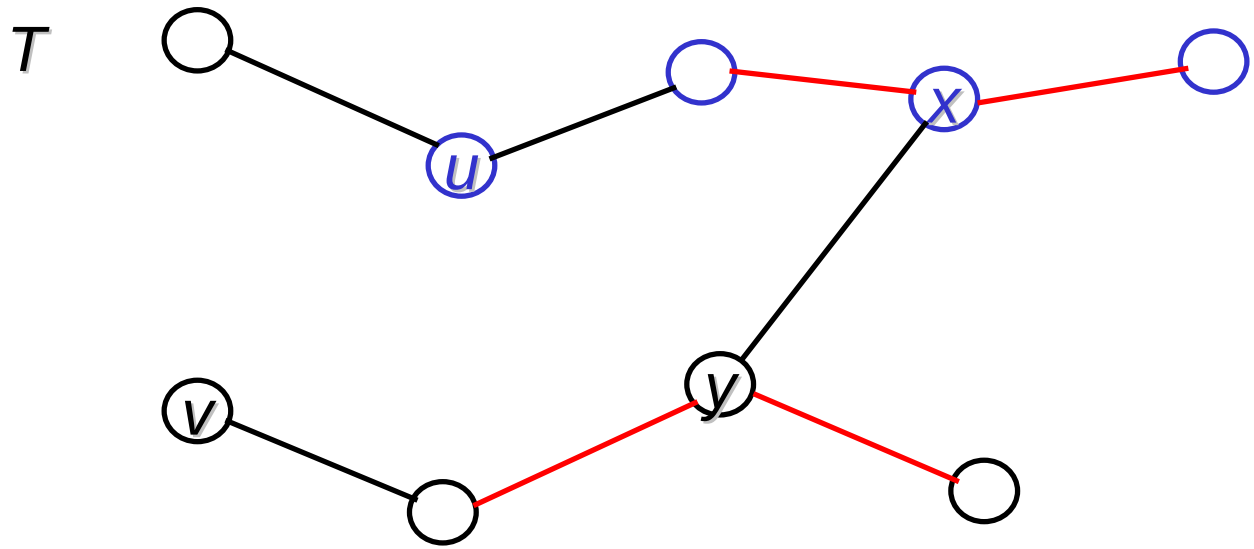
Beweis:

1. Fall: $(u,v) \in T$: ok

2. Fall: $(u,v) \notin T$

   Wir konstruieren min. spannenden Baum T' mit $(u,v) \in T'$ und $A \subseteq T'$

Sichere Kanten



Durch Hinzufügen von (u,v) zu T erhalten wir einen Zyklus.

Auf diesem gibt es (mindestens) eine Kante (x,y), die ebenfalls den Schnitt schneidet.

Sichere Kanten

$$T' = T \setminus \{(x,y)\} \cup \{(u,v)\}$$

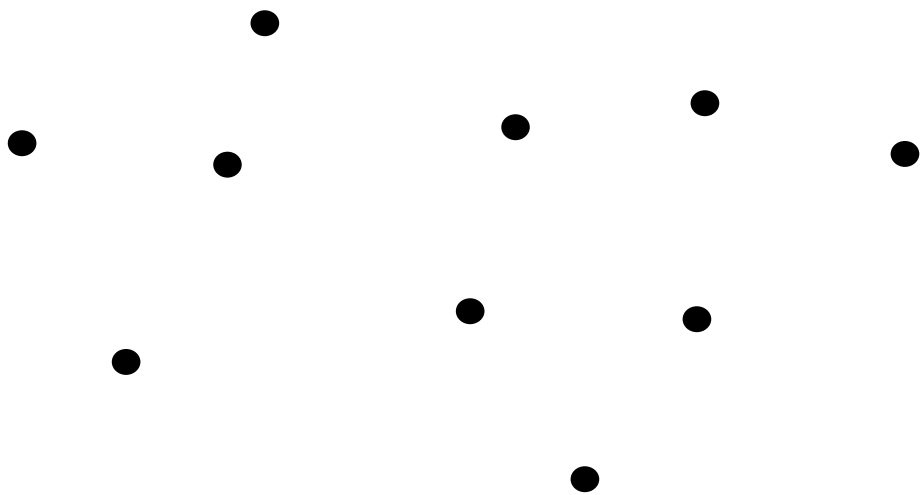
ist ein min. spannender Baum, denn

$$c(T') = c(T) - c(x,y) + c(u,v) \leq c(T)$$

4. Der Graph G_A

$G_A = (V, A)$

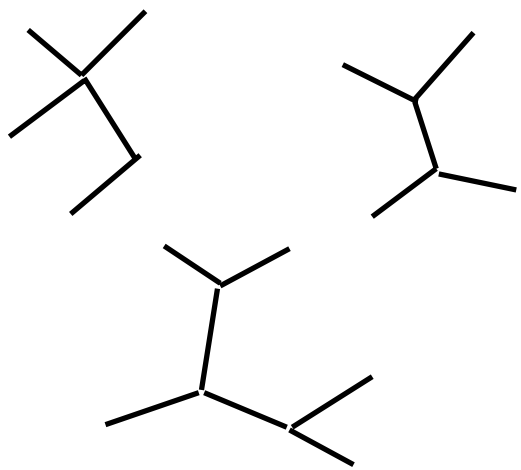
- ist ein Wald, d.h. eine Menge von Bäumen
- anfangs, wenn $A = \varnothing$, ist jeder Baum ein einzelner Knoten
- eine sichere Kante verbindet verschiedene Bäume



Der Graph G_A

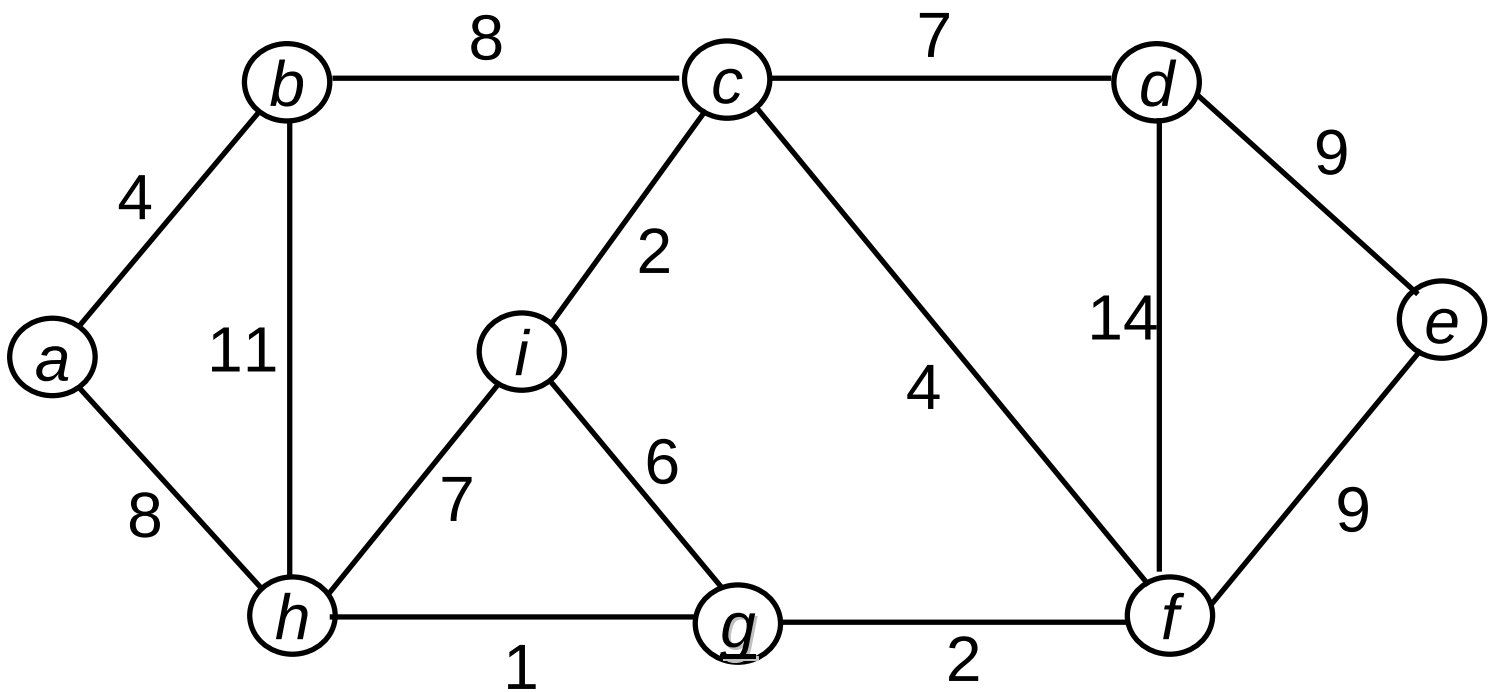
Korollar: Sei B ein Baum in $G_A = (V, A)$. Ist (u,v) eine Kante minimaler Kosten, die B und einen anderen Baum in G_A verbindet, dann ist (u,v) sicher für A.

Beweis: $(B, V \setminus B)$ respektiert A und
(u,v) ist eine min. Kante für diesen Schnitt.



5. Algorithmus von Kruskal

Wähle stets eine Kante minimaler Kosten, die zwei Bäume B_1 und B_2 in G_A verbindet.



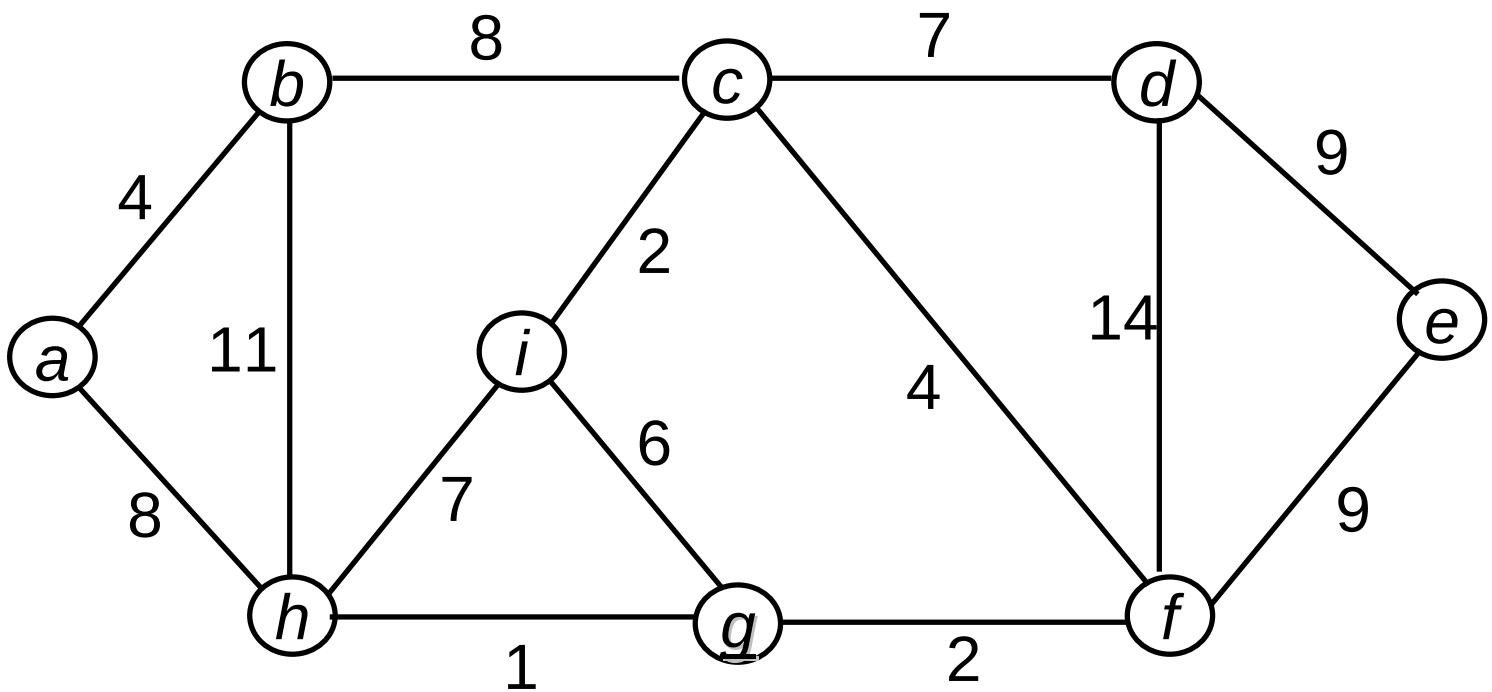
Algorithmus von Kruskal

1. $A \leftarrow \emptyset$;
2. **for all** $v \in V$ **do** $B_v \leftarrow \{ v \}$; **endfor;**
3. Erzeuge eine Liste L der Kanten in E, welche gemäß nicht-fallenden Kantenkosten sortiert ist;
4. **for all** (u,v) in L **do**
5. $B_1 \leftarrow$ FIND(u); $B_2 \leftarrow$ FIND(v);
6. **if** $B_1 \neq B_2$ **then**
7. $A \leftarrow A \cup \{(u,v)\}$; UNION ($B_1$, B_2, B_1);
8. **endif;**
9. **endfor;**

Laufzeit: $O(m\ \alpha(m,n) + m + n \log n)$

6. Algorithmus von Prim

A ist immer ein einziger Baum. Starte mit einem beliebigen Wurzelknoten *w*. Füge in jedem Schritt eine minimale Kante hinzu, die einen Knoten in *A* mit einem Knoten in $V \setminus A$ verbindet.



Implementierung

Q : Prioritätswarteschlange, die alle Knoten $v \in V \setminus A$ enthält.

Schlüssel von v: min. Kosten einer Kante, die v mit einem Knoten aus A verbindet.

Für einen Knoten v ist $p[v]$ der Elter-Knoten von v in dem Baum.

$A = \{ (v, p[v]) : v \in V - \{w\} - Q\}$

Algorithmus von Prim

1. **for all** $v \in V$ **do** Insert(Q, ∞, v); **endfor;**
2. Wähle einen Knoten $w \in V$ als Wurzel;
3. DecreaseKey(Q, 0, w); $p[w] \leftarrow$ nil;
4. **while** $\neg$Empty(Q) **do**
5. (d, u) $\leftarrow$ DeleteMin(Q);
6. **for all** (u,v) $\in E$ **do**
7. **if** $v \in Q$ und $c(u,v) <$ Schlüssel von v **then**
8. DecreaseKey(Q, $c(u,v)$, v); $p[v] \leftarrow u$;
9. **endif;**
10. **endfor;**
11. **endwhile;**

Laufzeit: $O(n \log n + m)$