# Datenstrukturen und Algorithmen

## Vorlesung 14: Minimale Spannbäume

Prof. Dr. Erika Ábrahám

Theorie Hybrider Systeme
Informatik 2

http://ths.rwth-aachen.de/teaching/ss-14/
datenstrukturen-und-algorithmen/

Diese Präsentation verwendet in Teilen Folien von Joost-Pieter Katoen.

27. Juni 2014

RWTH AACHEN UNIVERSITY

# Übersicht

Minimale Spannbäume

# Übersicht

# Probleme auf kantengewichteten Graphen

Betrachte einen gewichteten Graphen, wobei den *Kanten* ein Gewicht zugeordnet ist.

# Probleme auf kantengewichteten Graphen

Betrachte einen gewichteten Graphen, wobei den *Kanten* ein Gewicht zugeordnet ist.

**Beispiel (Optimierungsprobleme auf Graphen)**

# Probleme auf kantengewichteten Graphen

Betrachte einen gewichteten Graphen, wobei den *Kanten* ein Gewicht zugeordnet ist.

**Beispiel (Optimierungsprobleme auf Graphen)**

- Finde den minimalen Spannbaum (minimal spanning tree) in einem ungerichteten Graphen.

# Probleme auf kantengewichteten Graphen

Betrachte einen gewichteten Graphen, wobei den *Kanten* ein Gewicht zugeordnet ist.

**Beispiel (Optimierungsprobleme auf Graphen)**

- Finde den minimalen Spannbaum (minimal spanning tree) in einem ungerichteten Graphen.
- Finde den kürzesten Weg (shortest path) in einem gerichteten oder ungerichteten Graphen.

# Probleme auf kantengewichteten Graphen

Betrachte einen gewichteten Graphen, wobei den *Kanten* ein Gewicht zugeordnet ist.

**Beispiel (Optimierungsprobleme auf Graphen)**

- Finde den minimalen Spannbaum (minimal spanning tree) in einem ungerichteten Graphen.
- Finde den kürzesten Weg (shortest path) in einem gerichteten oder ungerichteten Graphen.

„Minimal“ und „kürzester“ beziehen sich hierbei auf die besuchten Gewichte.

# Probleme auf kantengewichteten Graphen

Betrachte einen gewichteten Graphen, wobei den *Kanten* ein Gewicht zugeordnet ist.

**Beispiel (Optimierungsprobleme auf Graphen)**

- Finde den minimalen Spannbaum (minimal spanning tree) in einem ungerichteten Graphen.
- Finde den kürzesten Weg (shortest path) in einem gerichteten oder ungerichteten Graphen.

„Minimal“ und „kürzester“ beziehen sich hierbei auf die besuchten Gewichte. Die Gewichte können als Kosten für die Benutzung der Kante aufgefasst werden.

# Probleme auf kantengewichteten Graphen

Betrachte einen gewichteten Graphen, wobei den *Kanten* ein Gewicht zugeordnet ist.

**Beispiel (Optimierungsprobleme auf Graphen)**

- Finde den minimalen Spannbaum (minimal spanning tree) in einem ungerichteten Graphen.
- Finde den kürzesten Weg (shortest path) in einem gerichteten oder ungerichteten Graphen.

„Minimal“ und „kürzester“ beziehen sich hierbei auf die besuchten Gewichte. Die Gewichte können als Kosten für die Benutzung der Kante aufgefasst werden.

Diese Probleme können durch Greedy-Algorithmen gelöst werden.

Minimale Spannbäume

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

Minimale Spannbäume

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- Treffe in jedem Schritt eine Entscheidung, die bezüglich eines lokalen Kriteriums optimal ist.

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- Treffe in jedem Schritt eine Entscheidung, die bezüglich eines lokalen Kriteriums optimal ist.
- Dieses Kriterium sollte günstig ($\rightarrow$ Komplexität) auswertbar sein.

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- Treffe in jedem Schritt eine Entscheidung, die bezüglich eines lokalen Kriteriums optimal ist.
- Dieses Kriterium sollte günstig ($\rightarrow$ Komplexität) auswertbar sein.
- Nachdem eine Wahl getroffen wurde, kann sie nicht mehr rückgängig gemacht werden.

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- Treffe in jedem Schritt eine Entscheidung, die bezüglich eines lokalen Kriteriums optimal ist.
- Dieses Kriterium sollte günstig ($\rightarrow$ Komplexität) auswertbar sein.
- Nachdem eine Wahl getroffen wurde, kann sie nicht mehr rückgängig gemacht werden.

Mit Greedy-Methoden ist nicht garantiert, dass immer die beste Lösung gefunden wird, denn

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- Treffe in jedem Schritt eine Entscheidung, die bezüglich eines lokalen Kriteriums optimal ist.
- Dieses Kriterium sollte günstig ($\rightarrow$ Komplexität) auswertbar sein.
- Nachdem eine Wahl getroffen wurde, kann sie nicht mehr rückgängig gemacht werden.

Mit Greedy-Methoden ist nicht garantiert, dass immer die beste Lösung gefunden wird, denn

- immer das lokale Optimum zu nehmen führt nicht automatisch auch zum globalen Optimum.

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- Treffe in jedem Schritt eine Entscheidung, die bezüglich eines lokalen Kriteriums optimal ist.
- Dieses Kriterium sollte günstig ($\rightarrow$ Komplexität) auswertbar sein.
- Nachdem eine Wahl getroffen wurde, kann sie nicht mehr rückgängig gemacht werden.

Mit Greedy-Methoden ist nicht garantiert, dass immer die beste Lösung gefunden wird, denn

- immer das lokale Optimum zu nehmen führt nicht automatisch auch zum globalen Optimum.
- In einigen Fällen, wie dem minimalen Spannbaum und dem Kürzesten-Wege-Problem, wird aber *immer* die optimale Lösung gefunden.

Minimale Spannbäume

# Greedy?

## Beispiel

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- Das Problem des Handlungsreisenden (Traveling Salesman Problem, TSP)

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- Das Problem des Handlungsreisenden (Traveling Salesman Problem, TSP)

Greedy kann gut sein:

- Behälterproblem (Bin Packing) ($\leqslant$ 2x Optimum)

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- Das Problem des Handlungsreisenden (Traveling Salesman Problem, TSP)

Greedy kann gut sein:

- Behälterproblem (Bin Packing) ($\leqslant$ 2x Optimum)

Greedy kann optimal sein:

- Minimaler Spannbaum, Kürzester-Weg-Problem.

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- Das Problem des Handlungsreisenden (Traveling Salesman Problem, TSP)

Greedy kann gut sein:

- Behälterproblem (Bin Packing) ($\leqslant$ 2x Optimum)

Greedy kann optimal sein:

- Minimaler Spannbaum, Kürzester-Weg-Problem.

Wann ist eine Greedy-Lösungsstrategie optimal?

- Optimale Lösung setzt sich aus optimalen Teilproblemen zusammen
- Unabhängigkeit von anderen Teillösungen

# Was ist ein minimaler Spannbaum?

## Spannbaum

Ein Spannbaum eines ungerichteten, zusammenhängenden Graphen $G$ ist

- ein Teilgraph von $G$, der ein *ungerichteter Baum* ist und alle Knoten von $G$ enthält.

# Was ist ein minimaler Spannbaum?

## Spannbaum

Ein Spannbaum eines ungerichteten, zusammenhängenden Graphen $G$ ist

- ein Teilgraph von $G$, der ein *ungerichteter Baum* ist und alle Knoten von $G$ enthält.

## Gewicht eines Graphen

Das Gewicht $W(G')$ eines Teilgraphen $G' = (V', E')$ vom gewichteten Graph $G$ ist:

- $W(G') = \sum_{(u,v) \in E'} W(u, v)$.

# Was ist ein minimaler Spannbaum?

## Spannbaum

Ein Spannbaum eines ungerichteten, zusammenhängenden Graphen $G$ ist

- ein Teilgraph von $G$, der ein *ungerichteter Baum* ist und alle Knoten von $G$ enthält.

## Gewicht eines Graphen

Das Gewicht $W(G')$ eines Teilgraphen $G' = (V', E')$ vom gewichteten Graph $G$ ist:

- $W(G') = \sum_{(u,v) \in E'} W(u, v)$.

## Minimaler Spannbaum

Ein Spannbaum mit minimalem Gewicht heißt minimaler Spannbaum (minimum spanning tree, MST).

# Anwendungen

## Problem

*Finde einen MST eines gewichteten, ungerichteten, zusammenhängenden Graphen.*

# Anwendungen

## Problem

*Finde einen MST eines gewichteten, ungerichteten, zusammenhängenden Graphen.*

## Beispiel

- Finde einen kostengünstigsten Weg, um eine Menge von Flughafenterminals, Städten, ... zu verbinden.

# Anwendungen

## Problem

*Finde einen MST eines gewichteten, ungerichteten, zusammenhängenden Graphen.*

## Beispiel

- Finde einen kostengünstigsten Weg, um eine Menge von Flughafenterminals, Städten, ... zu verbinden.
- Grundlage für viele andere Probleme, etwa Routing-Probleme („Wegfindung“).

# Anwendungen

## Problem

*Finde einen MST eines gewichteten, ungerichteten, zusammenhängenden Graphen.*

## Beispiel

- Finde einen kostengünstigsten Weg, um eine Menge von Flughafenterminals, Städten, ... zu verbinden.
- Grundlage für viele andere Probleme, etwa Routing-Probleme („Wegfindung“).
- Bestandteil von Approximationsalgorithmen für das TSP Problem.

# Anwendungen

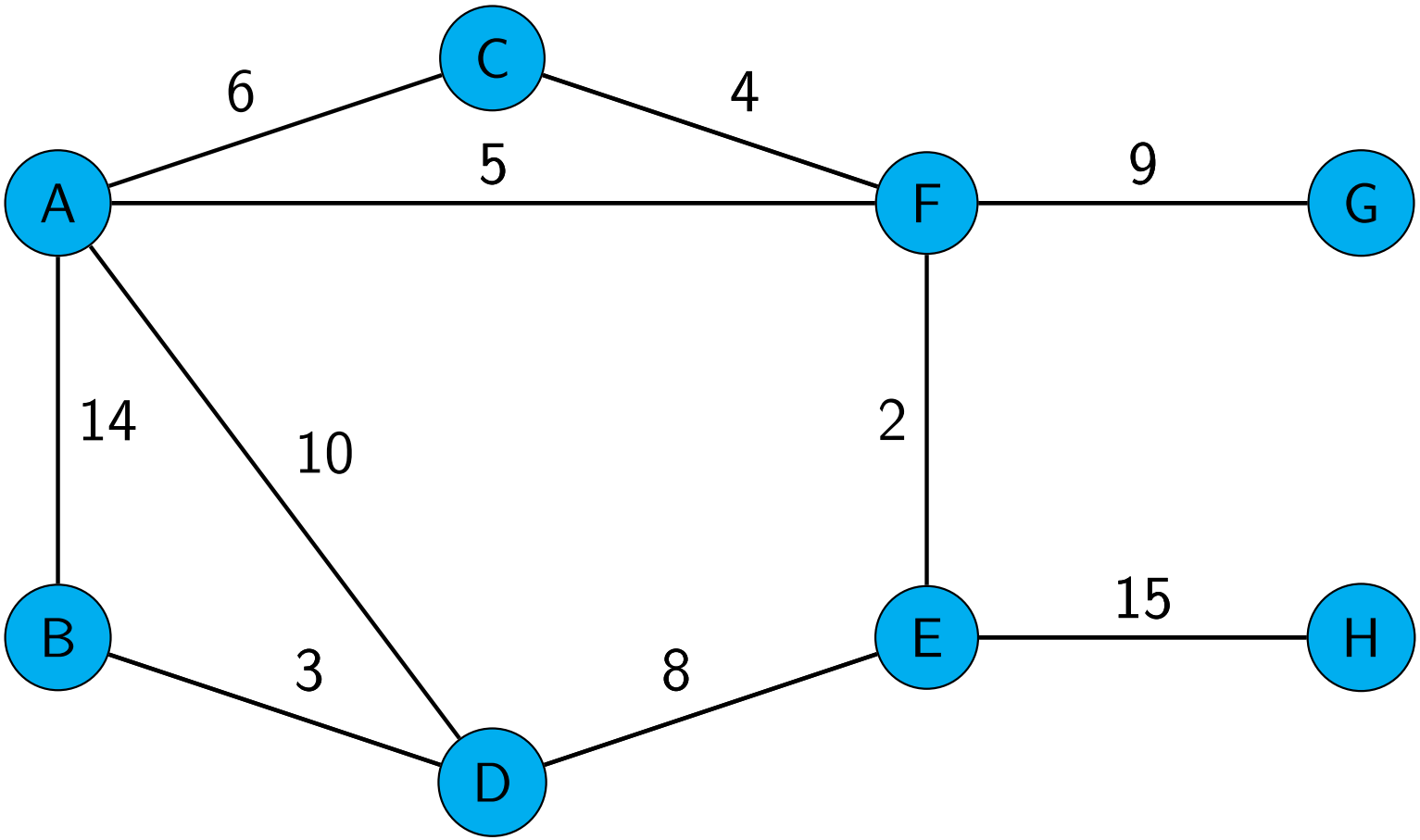
## Problem

*Finde einen MST eines gewichteten, ungerichteten, zusammenhängenden Graphen.*

## Beispiel

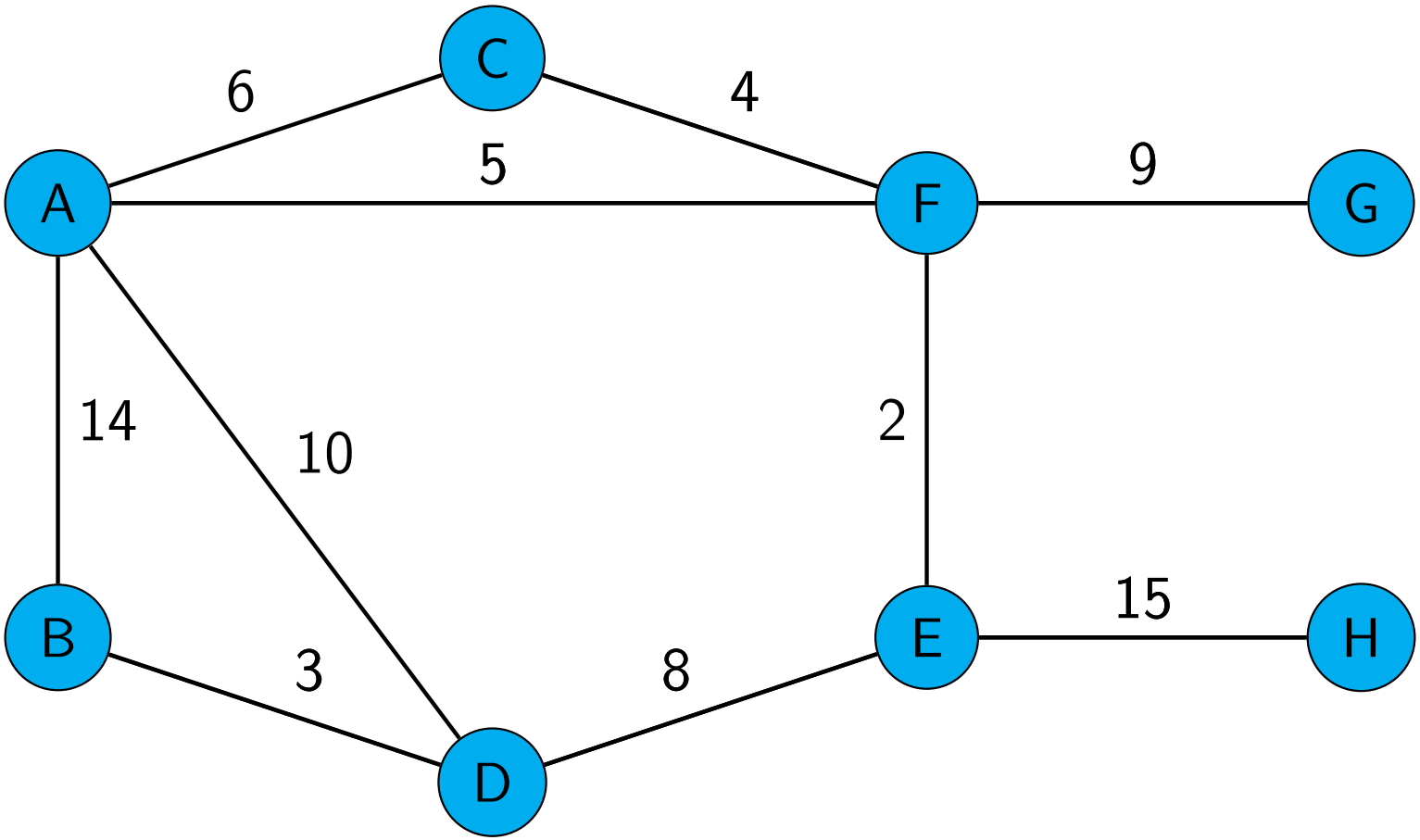
- Finde einen kostengünstigsten Weg, um eine Menge von Flughafenterminals, Städten, ... zu verbinden.
- Grundlage für viele andere Probleme, etwa Routing-Probleme („Wegfindung“).
- Bestandteil von Approximationsalgorithmen für das TSP Problem.
- Verdrahtung von Schaltungen mit geringstem Energieverbrauch.

# Minimaler Spannbaum – Beispiel



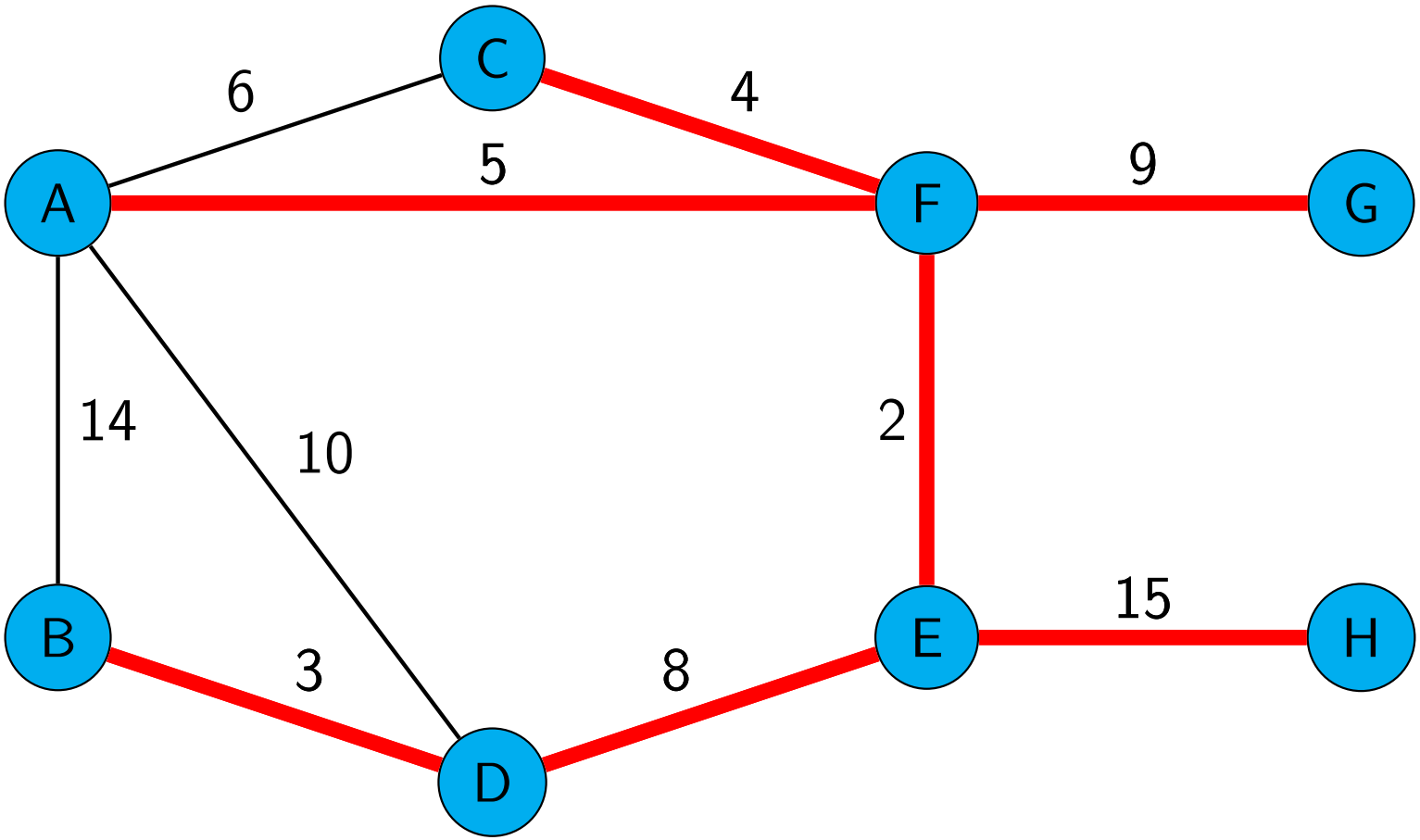
Minimale Spannbäume

# Minimaler Spannbaum – Beispiel



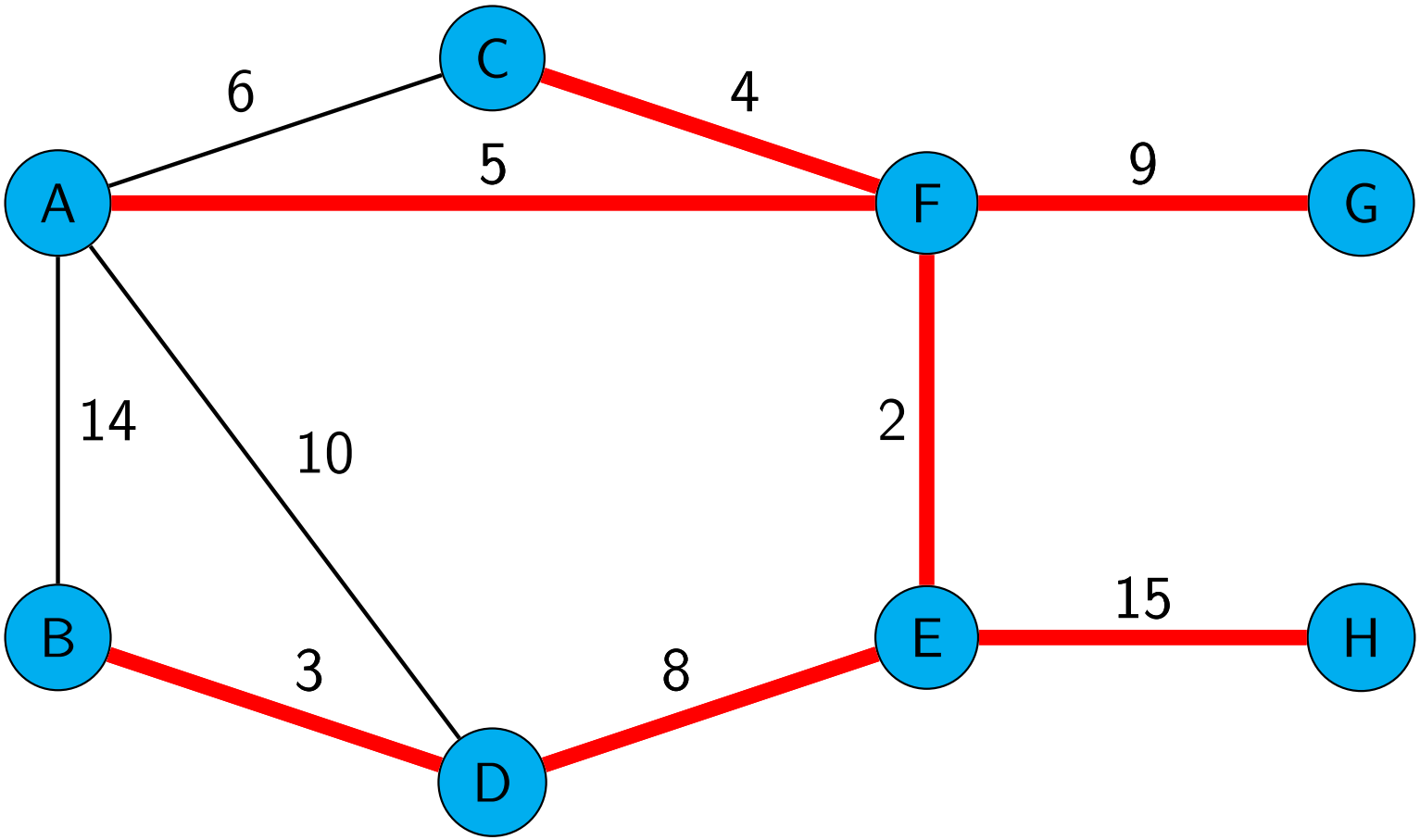

Was ist ein minimaler Spannbaum?

# Minimaler Spannbaum – Beispiel




Das ist *ein* minimaler Spannbaum (mit Gesamtgewicht 46).

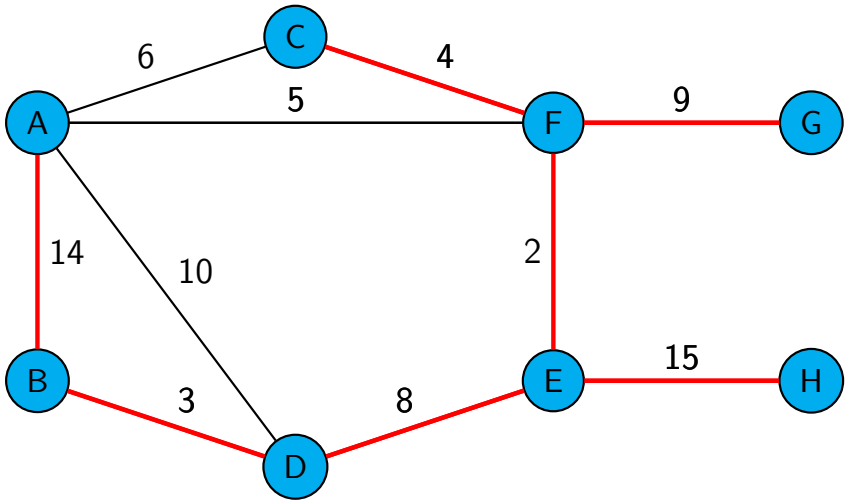
# Minimaler Spannbaum – Beispiel



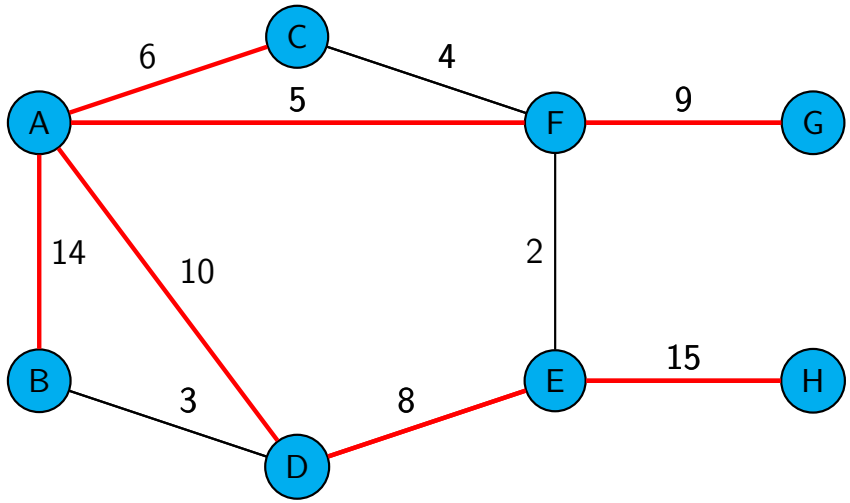

Das ist *ein* minimaler Spannbaum (mit Gesamtgewicht 46).
In diesem Fall ist es auch der einzige.

Minimale Spannbäume

# Tiefen- oder Breitensuche?




Tiefensuchbaum (von A gestartet)
Gesamtgewicht: 55




Breitensuchbaum (von A gestartet)
Gesamtgewicht: 67

Der Tiefensuchbaum und der Breitensuchbaum sind zwar Spannbäume, aber nicht notwendigerweise MSTs.

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

Grundkonzept:

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

Grundkonzept:

- Fange mit einen Baum mit nur einem Knoten an, indem ein beliebiger Knoten des Graphens ausgewählt wird.

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

Grundkonzept:

- Fange mit einen Baum mit nur einem Knoten an, indem ein beliebiger Knoten des Graphens ausgewählt wird.
- Finde die günstigste Kante (d. h. mit minimalem Gewicht), die den bisherigen Baum verlässt.

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

Grundkonzept:

- Fange mit einen Baum mit nur einem Knoten an, indem ein beliebiger Knoten des Graphens ausgewählt wird.
- Finde die günstigste Kante (d. h. mit minimalem Gewicht), die den bisherigen Baum verlässt.
- Füge den über diese Kante erreichten (Rand-)Knoten dem Baum hinzu, zusammen mit der Kante.

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

Grundkonzept:

- Fange mit einen Baum mit nur einem Knoten an, indem ein beliebiger Knoten des Graphens ausgewählt wird.
- Finde die günstigste Kante (d. h. mit minimalem Gewicht), die den bisherigen Baum verlässt.
- Füge den über diese Kante erreichten (Rand-)Knoten dem Baum hinzu, zusammen mit der Kante.
- Fahre fort, bis keine weiteren Randknoten mehr vorhanden sind.

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

Grundkonzept:

- Fange mit einen Baum mit nur einem Knoten an, indem ein beliebiger Knoten des Graphens ausgewählt wird.
- Finde die günstigste Kante (d. h. mit minimalem Gewicht), die den bisherigen Baum verlässt.
- Füge den über diese Kante erreichten (Rand-)Knoten dem Baum hinzu, zusammen mit der Kante.
- Fahre fort, bis keine weiteren Randknoten mehr vorhanden sind.

Ist das korrekt?

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien ein:

Baum-Knoten : Knoten, die Teil vom bis jetzt konstruierten Baum sind.

Rand-knoten: Nicht im Baum, jedoch adjazent zu Knoten im Baum.

Ungesehene Knoten: Alle anderen Knoten.

Grundkonzept:

- Fange mit einen Baum mit nur einem Knoten an, indem ein beliebiger Knoten des Graphens ausgewählt wird.
- Finde die günstigste Kante (d. h. mit minimalem Gewicht), die den bisherigen Baum verlässt.
- Füge den über diese Kante erreichten (Rand-)Knoten dem Baum hinzu, zusammen mit der Kante.
- Fahre fort, bis keine weiteren Randknoten mehr vorhanden sind.

Ist das korrekt? Und wenn, was ist die Komplexität?

Minimale Spannbäume

# Prim's Algorithmus: Implementierung

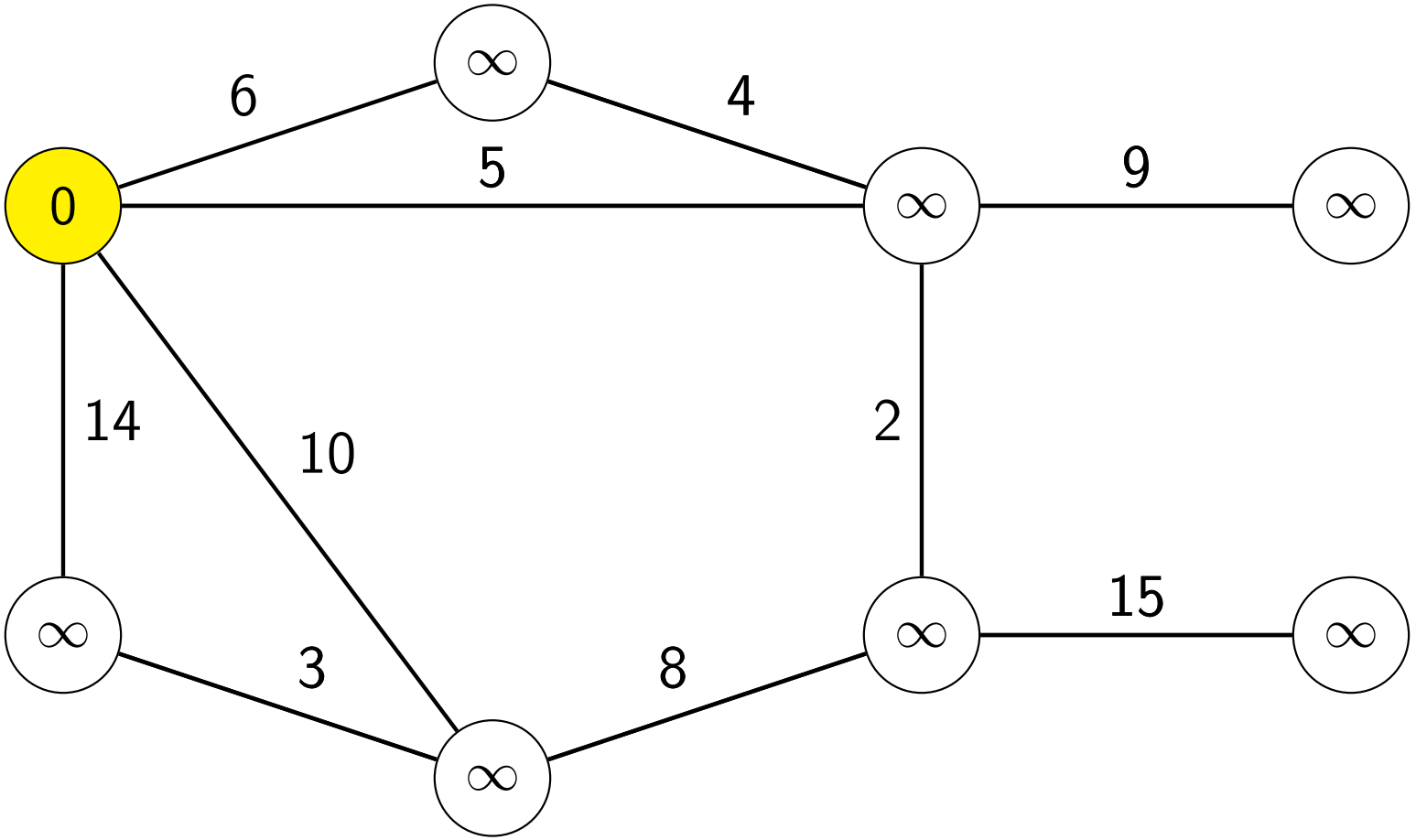
```
1  // Parameter: ungerichteter gewichteter Graph mit n Knoten und
       ein Startknoten
2  void primMST(int adj[n], int n, int start) {
3    for (int i = 0; i < n; i++) {
4       key[i] = inf; p[i] = null;
5    }
6    key[start] = 0; //start ist einziger Randknoten mit Kosten 0
7    Q = {0,...,n-1};
8    while (Q not empty) {
9      //verschiebe den billigsten Randknoten in den Baum
10     v = extractMin(Q);
11     // aktualisiere Kosten für Randknoten
12     for each ((u,w) in adj[v]) {
13       if (u in Q and w < key[u]) {
14         p[u] = v;
15         key[u] = w;
16       }
17     }
18   }
```

// extractMin(Q) bestimmt ein Element e aus Q mit
// minimalem Schlüssel key[e], entfernt e aus Q, und
// gibt e zurück

edge

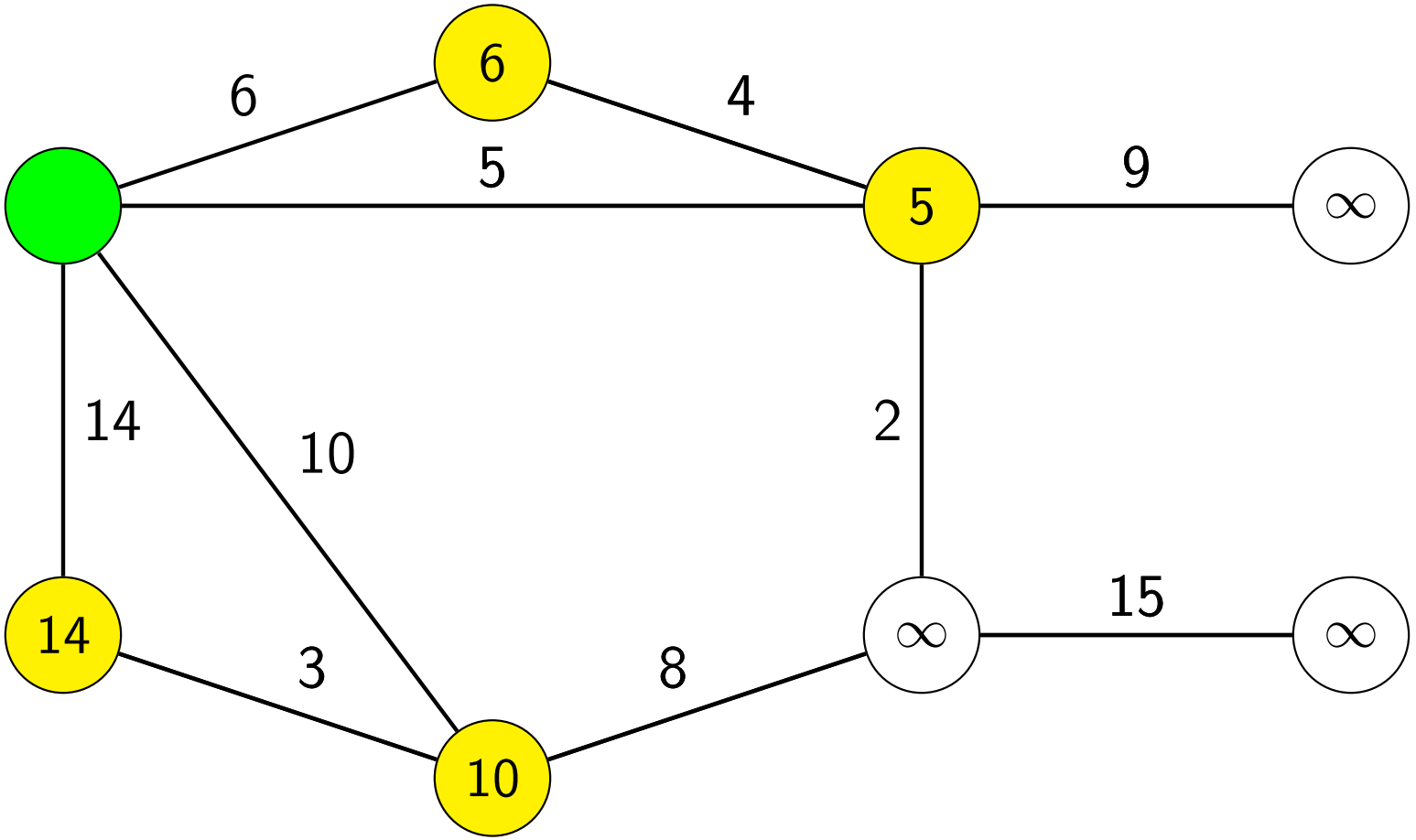
int u = edge.target; float w = edge.weight;

# Prim's Algorithmus – Beispiel



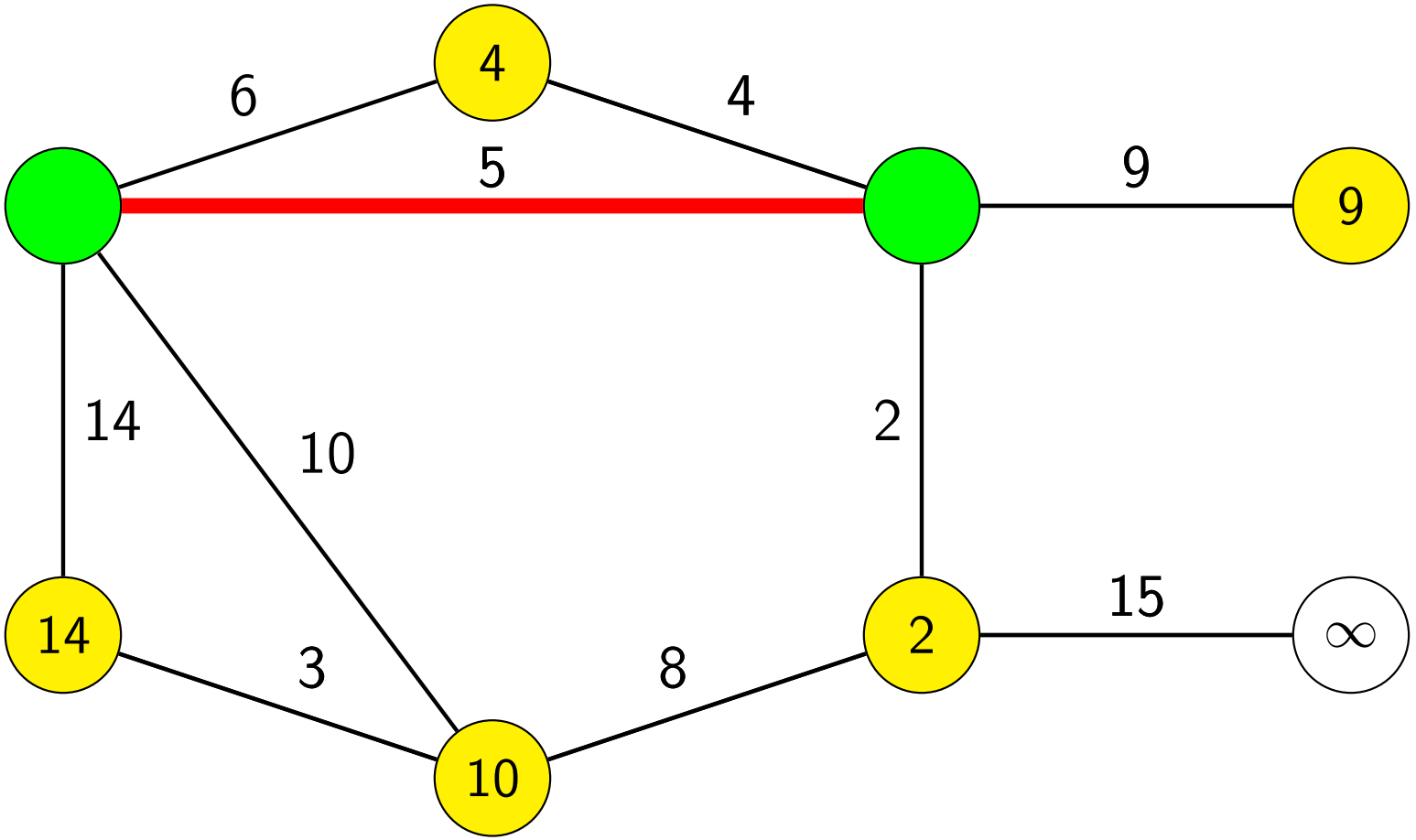
Minimale Spannbäume

# Prim's Algorithmus – Beispiel



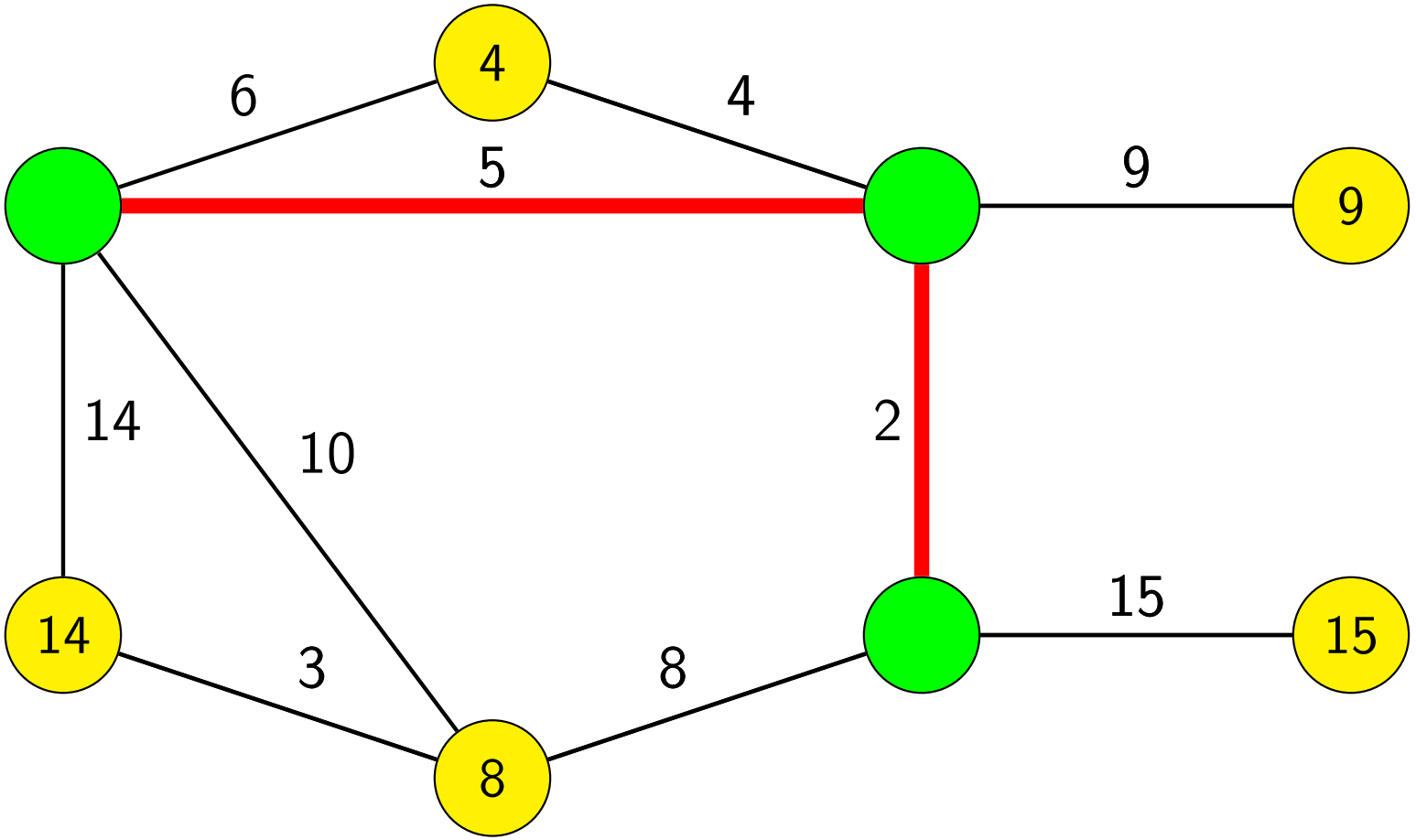
Minimale Spannbäume

# Prim's Algorithmus – Beispiel



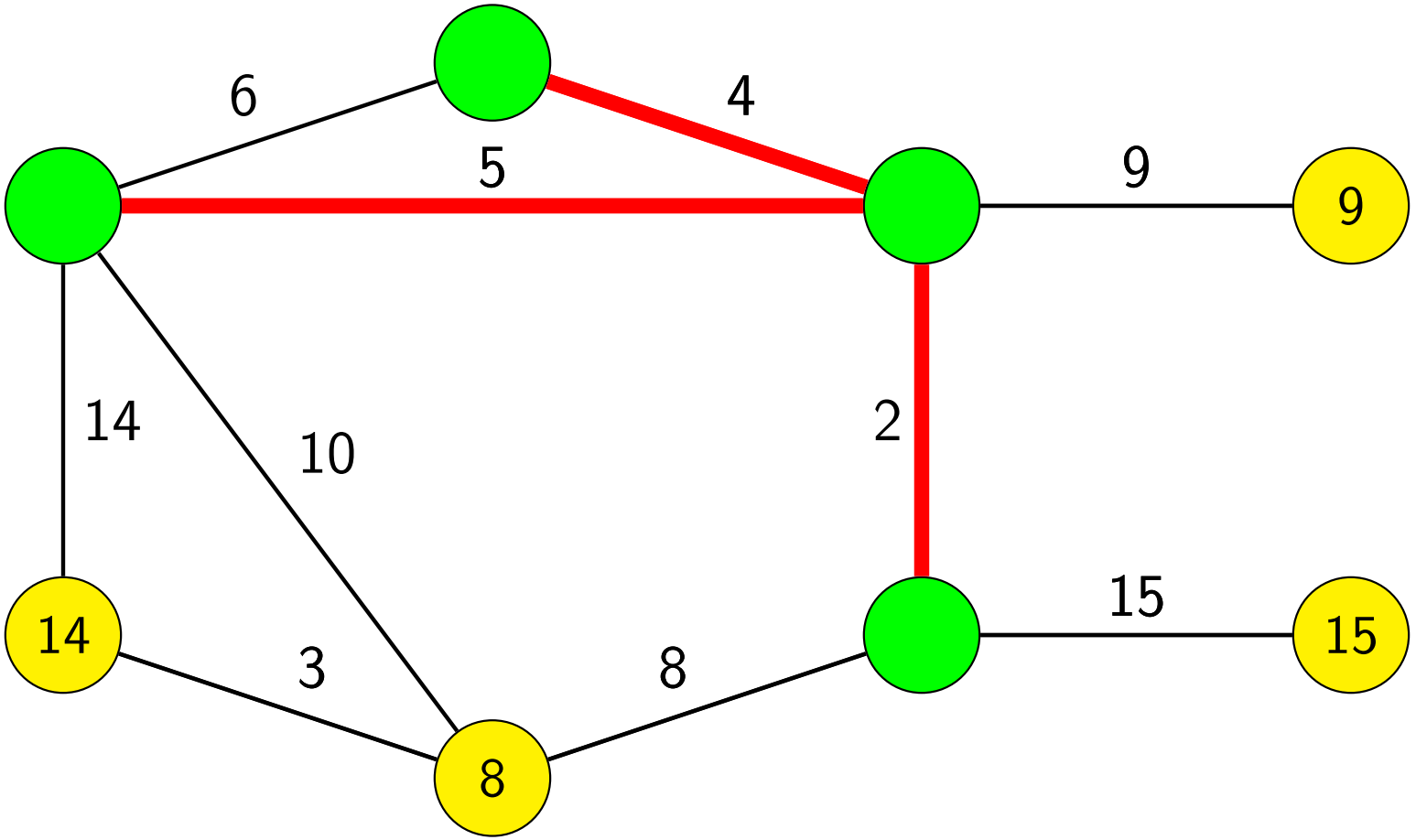
Minimale Spannbäume

# Prim's Algorithmus – Beispiel

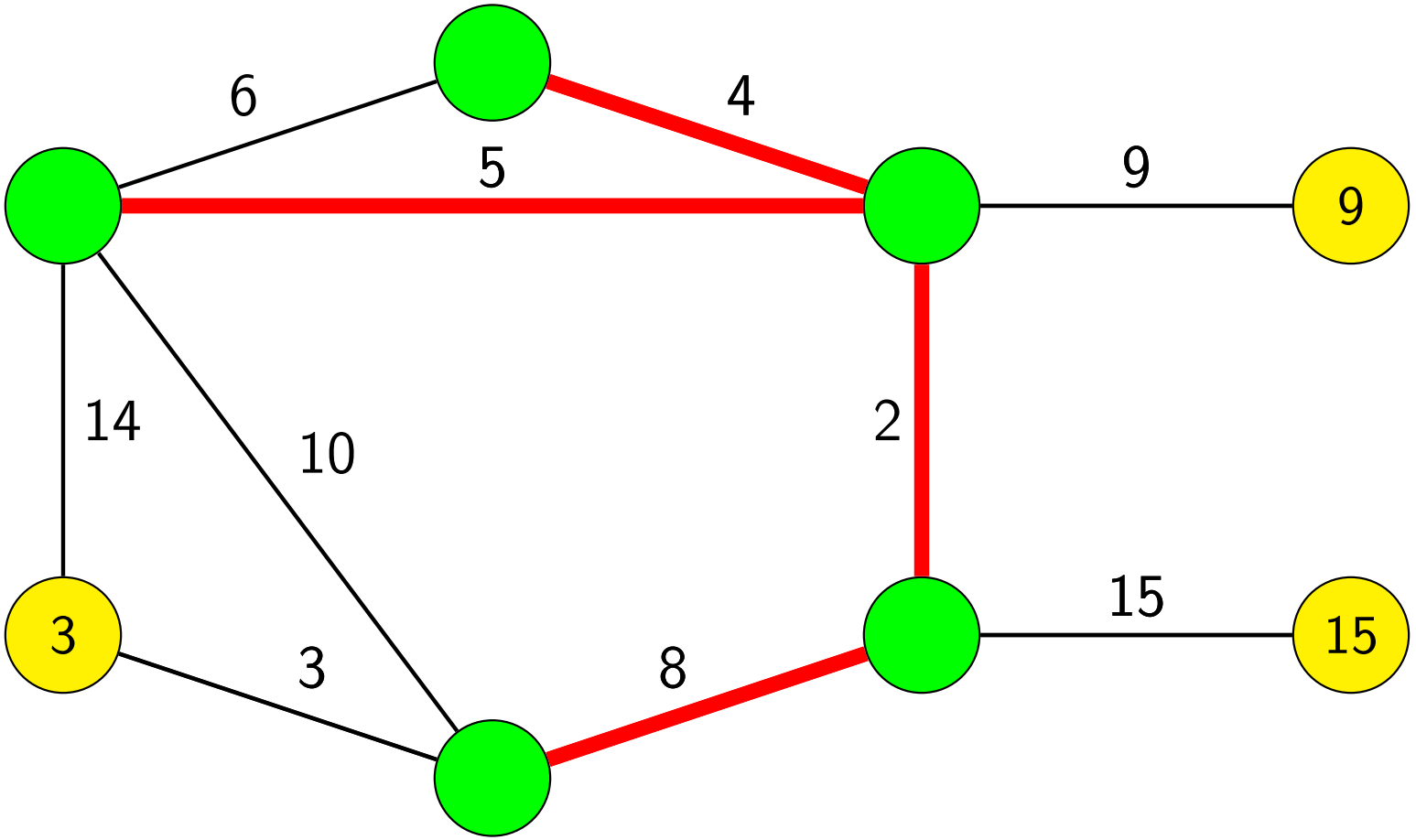


Minimale Spannbäume

# Prim's Algorithmus – Beispiel

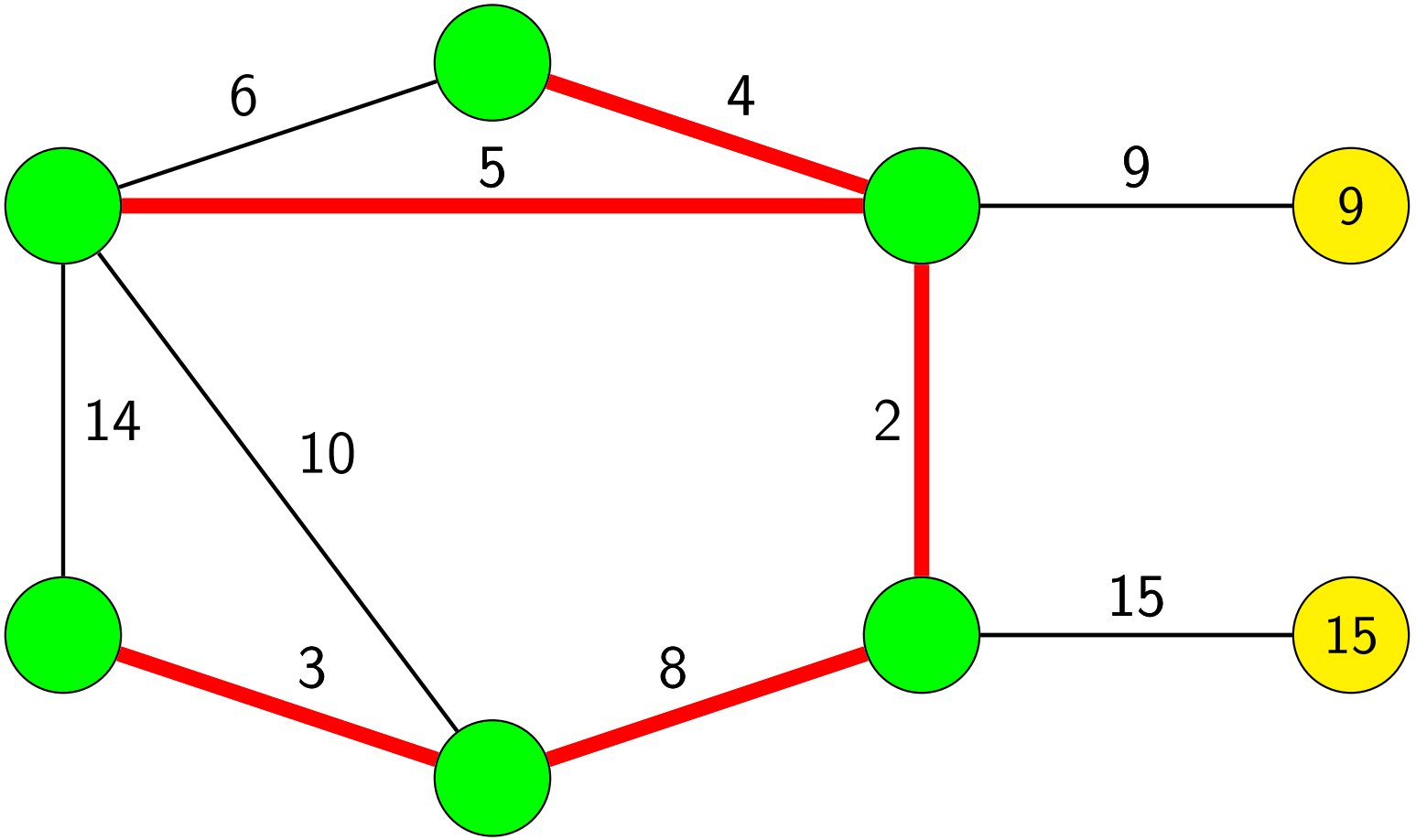


# Prim's Algorithmus – Beispiel



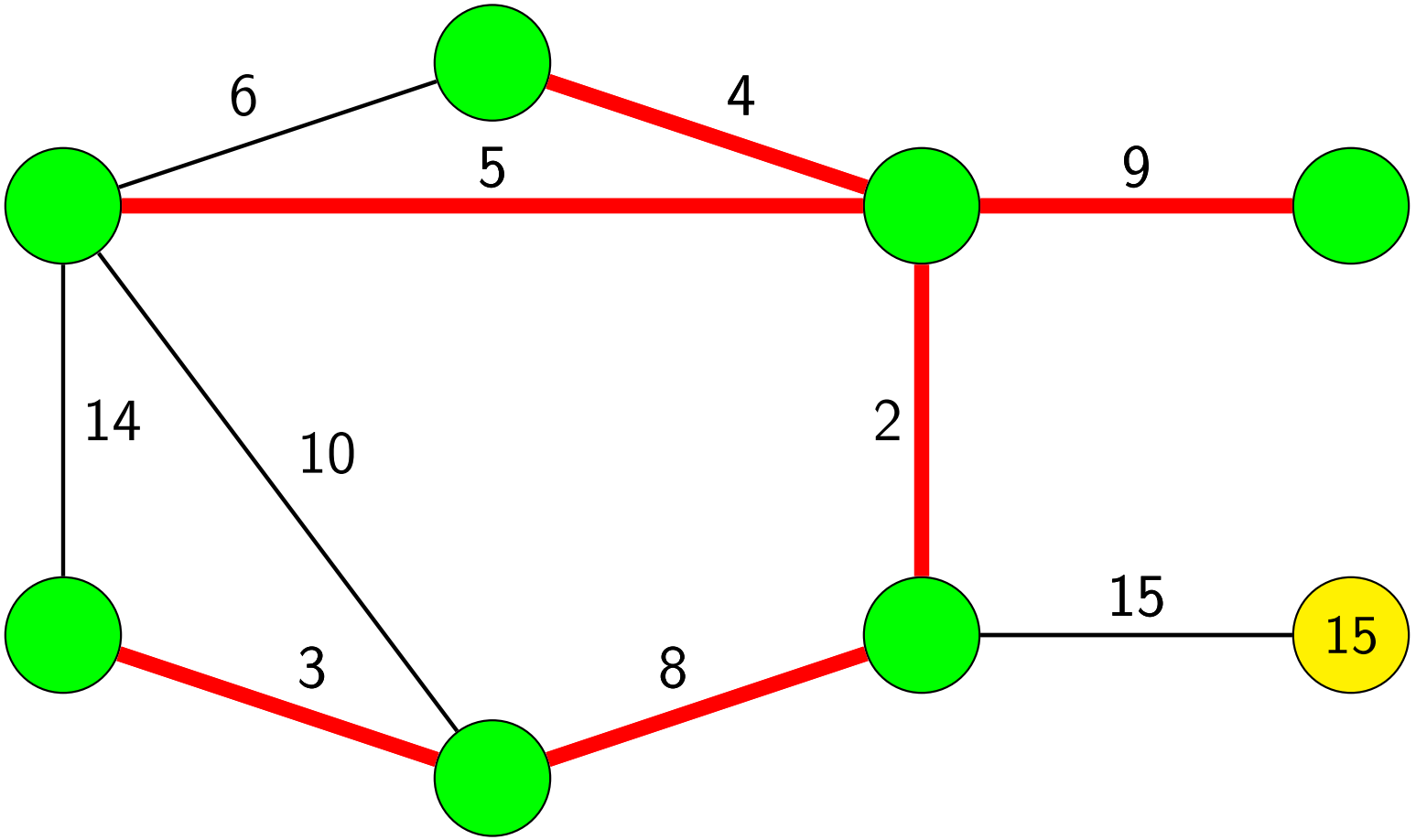
Minimale Spannbäume

# Prim's Algorithmus – Beispiel



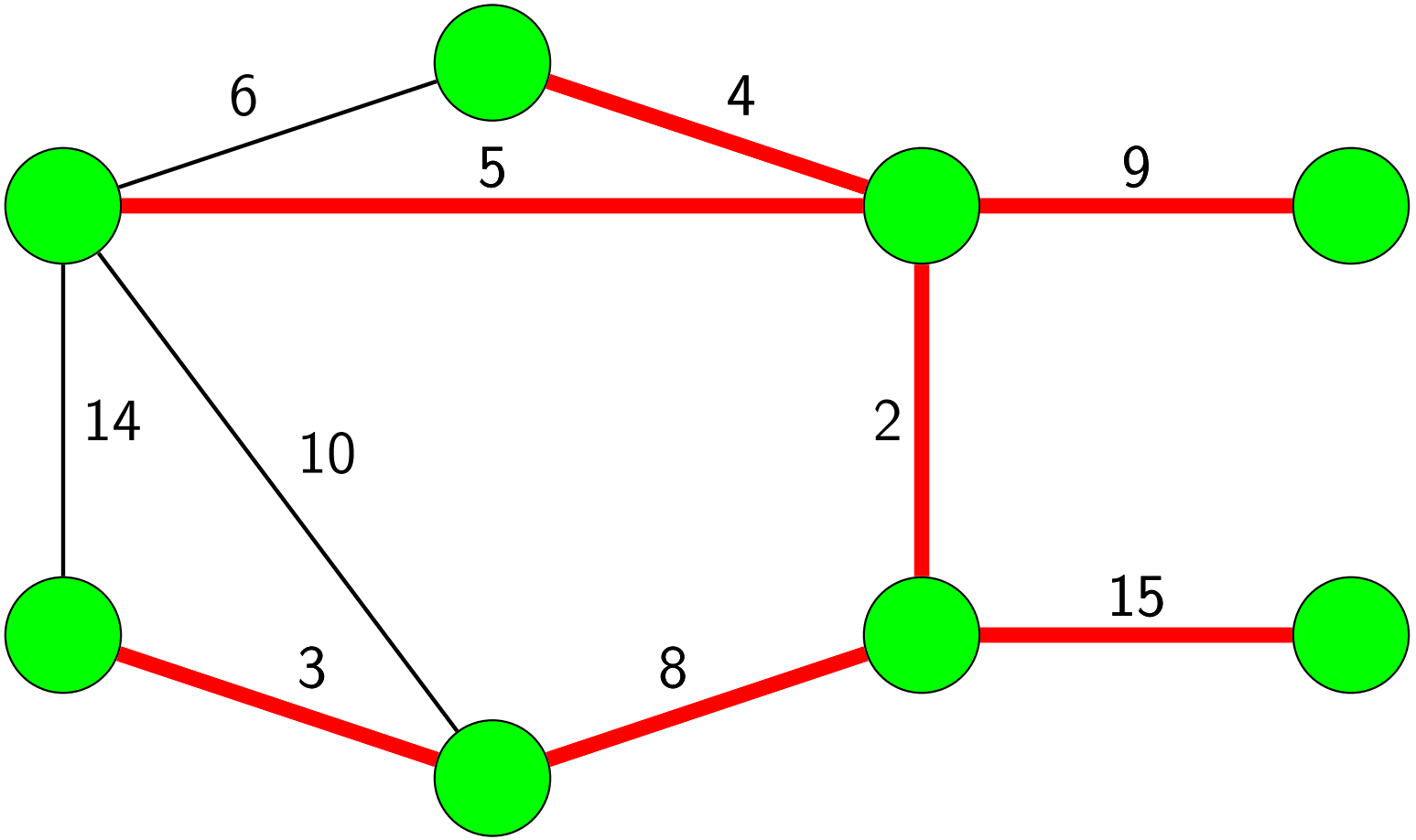
Minimale Spannbäume

# Prim's Algorithmus – Beispiel



Minimale Spannbäume

# Prim's Algorithmus – Beispiel



# Korrektheit von Prim

**Lemma**

*Unmittelbar vor jeder Iteration $i$ ist der aktuelle Baum $B_i = (V_i, E_i)$ eine Teilmenge eines minimalen Spannbaumes.*

Minimale Spannbäume

# Korrektheit von Prim

## Lemma

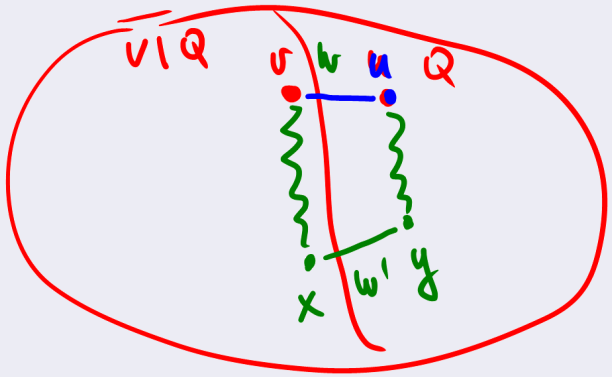
*Unmittelbar vor jeder Iteration $i$ ist der aktuelle Baum $B_i = (V_i, E_i)$ eine Teilmenge eines minimalen Spannbaumes.*

## Beweis.

Per Induktion über die Iterationstiefe.

- $k = 1$: Trivial, da $E_i = \varnothing$.

$1 \leq i < k \Rightarrow i+1 \leq k$




$w(v,u) \leq w(x,y)$

$w(T') = w(T) - w(x,y) + w(v,u)$

$\leq w(T)$

$w(T') \overset{!!}{=} w(T)$

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

# Korrektheit von Prim

## Lemma

*Unmittelbar vor jeder Iteration $i$ ist der aktuelle Baum $B_i = (V_i, E_i)$ eine Teilmenge eines minimalen Spannbaumes.*

## Beweis.

- $k > 1$: Sei die Eigenschaft gültig für alle $1 \leqslant i < k$; wir zeigen sie für $k$. Sei $T$ ein MST der $B_{k-1}$ enthält und $B_k = (V_{k-1} \cup \{v\}, E_{k-1} \cup \{(u, v)\})$. Wenn $T$ auch $B_k$ enthält, dann sind wir fertig. Falls $(u, v)$ nicht in $T$ enthalten ist, dann gibt es einen anderen Pfad von $u$ nach $v$ in $T$. Da $u \in V_{k-1}$ und $v \notin V_{k-1}$, gibt es eine Kante $(x, y)$ auf dem Pfad mit $x \in V_{k-1}$ und $y \notin V_{k-1}$. Es gilt $W(x, y) > W(u, v)$ wegen des Wahls von $(u, v)$. Der Baum $T'$, der aus $T$ durch entfernen von $(x, y)$ und Hinzufügen von $(u, v)$ entsteht, ist ein Spannbaum, der auch minimal ist (da $T$ minimal ist und $W(T') = W(T) - W(x, y) + W(u, v) \leqslant W(T)$). Da $B_k$ in $T'$ enthalten ist, gilt die Eigenschaft.

# ADT zum Vorhalten der Randknoten

Die benötigten Operationen für den Algorithmus von Prim sind:

- Wähle eine billigste Kante zu einem Randknoten (Kantenkandidat).
- Reklassifiziere einen Randknoten als Baumknoten (füge den Kantenkandidat zum Baum hinzu).
- Ändere die Kosten (Randgewicht) eines Randknotens, wenn ein günstigerer Kantenkandidat gefunden wird.

# ADT zum Vorhalten der Randknoten

Die benötigten Operationen für den Algorithmus von Prim sind:

- Wähle eine billigste Kante zu einem Randknoten (Kantenkandidat).
- Reklassifiziere einen Randknoten als Baumknoten (füge den Kantenkandidat zum Baum hinzu).
- Ändere die Kosten (Randgewicht) eines Randknotens, wenn ein günstigerer Kantenkandidat gefunden wird.

Idee: *Ordne die Randknoten nach ihrer Priorität (= Randgewicht).*

# ADT zum Vorhalten der Randknoten

Die benötigten Operationen für den Algorithmus von Prim sind:

- Wähle eine billigste Kante zu einem Randknoten (Kantenkandidat).
- Reklassifiziere einen Randknoten als Baumknoten (füge den Kantenkandidat zum Baum hinzu).
- Ändere die Kosten (Randgewicht) eines Randknotens, wenn ein günstigerer Kantenkandidat gefunden wird.

Idee: *Ordne die Randknoten nach ihrer Priorität (= Randgewicht).*

$\Rightarrow$ Wir entscheiden uns für die Prioritätswarteschlange als Datenstruktur für die Randknoten.

Minimale Spannbäume

# Vorläufige Komplexitätsanalyse

Im Worst-Case:

# Vorläufige Komplexitätsanalyse

Im Worst-Case:

- Jeder Knoten muss zur Prioritätswarteschlange hinzugefügt werden.
- Auf jeden Knoten muss auch wieder zugegriffen werden und er muss gelöscht werden.
- Die Priorität eines Randknotens muss nach jeder gefundenen Kante angepasst werden.

# Vorläufige Komplexitätsanalyse

Im Worst-Case:

- Jeder Knoten muss zur Prioritätswarteschlange hinzugefügt werden.
- Auf jeden Knoten muss auch wieder zugegriffen werden und er muss gelöscht werden.
- Die Priorität eines Randknotens muss nach jeder gefundenen Kante angepasst werden.

Bei einem Graph mit $n$ Knoten und $m$ Kanten ergibt sich:

$$T(n,m) \in \mathcal{O}\left(n \cdot T(\texttt{insert}) + n \cdot T(\texttt{getMin}) + n \cdot T(\texttt{delMin}) + m \cdot T(\texttt{decrKey})\right)$$

# Vorläufige Komplexitätsanalyse

Im Worst-Case:

- Jeder Knoten muss zur Prioritätswarteschlange hinzugefügt werden.
- Auf jeden Knoten muss auch wieder zugegriffen werden und er muss gelöscht werden.
- Die Priorität eines Randknotens muss nach jeder gefundenen Kante angepasst werden.

Bei einem Graph mit $n$ Knoten und $m$ Kanten ergibt sich:

$$T(n,m) \in \mathcal{O}\left(n \cdot T(\texttt{insert}) + n \cdot T(\texttt{getMin}) + n \cdot T(\texttt{delMin}) + m \cdot T(\texttt{decrKey})\right)$$

- Welche Implementierung der Prioritätswarteschlange ist dafür gut geeignet?

# Drei Prioritätswarteschlangenimplementierungen

$$T(n,m) \in \mathcal{O}\left(n \cdot T(\texttt{insert}) + n \cdot T(\texttt{getMin}) + n \cdot T(\texttt{delMin}) + m \cdot T(\texttt{decrKey})\right)$$

| | Implementierung | | |
|---|---|---|---|
| Operation | unsortiertes Array | sortiertes Array | Heap |
| `isEmpty()` | $\Theta(1)$ | $\Theta(1)$ | $\Theta(1)$ |
| `insert(e,k)` | $\Theta(1)$ | $\Theta(n)$ | $\Theta(\log n)$ |
| `getMin()` | $\Theta(n)$ | $\Theta(1)$ | $\Theta(1)$ |
| `delMin()` | $\Theta(n)$ | $\Theta(1)$ | $\Theta(\log n)$ |
| `decrKey(e,k)` | $\Theta(1)$ | $\Theta(n)$ | $\Theta(\log n)$ |
| Prim | $\mathcal{O}(n^2 + m))$ | $\mathcal{O}(n^2 + m \cdot n)$ | $\mathcal{O}(n \log n + m \log n)$ |