

# Datenstrukturen und Algorithmen

## Vorlesung 16: Minimale Spannbäume

Joost-Pieter Katoen

Lehrstuhl für Informatik 2  
Software Modeling and Verification Group

<http://www-i2.informatik.rwth-aachen.de/i2/dsal12/>

15. Juni 2012

# Übersicht

## 1 Minimale Spannbäume

- Greedy Algorithmen
- Minimaler Spannbaum
- Algorithmus von Prim

# Übersicht

## 1 Minimale Spannbäume

- Greedy Algorithmen
- Minimaler Spannbaum
- Algorithmus von Prim

# Probleme auf kanten-gewichteten Graphen

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

# Probleme auf kanten-gewichteten Graphen

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

## Beispiel (Optimierungsprobleme auf Graphen)

# Probleme auf kanten-gewichteten 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 kanten-gewichteten 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 kanten-gewichteten 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 kanten-gewichteten 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 kanten-gewichteten 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.

# Greedy Algorithmen

Eine Lösungstechnik:

**Greedy-Algorithmen („gierig“)**

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- ▶ Treffe in jedem Schritt eine Entscheidung, die bezüglich eines „kurzfristigen“ Kriteriums optimal ist.

# Greedy Algorithmen

Eine Lösungstechnik:

## Greedy-Algorithmen („gierig“)

- ▶ Treffe in jedem Schritt eine Entscheidung, die bezüglich eines „kurzfristigen“ 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 „kurzfristigen“ 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 „kurzfristigen“ 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 „kurzfristigen“ 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 „kurzfristigen“ 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.

# Greedy?

## Beispiel

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- ▶ Traveling Salesman Problem (TSP)

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- ▶ Traveling Salesman Problem (TSP)

Greedy kann gut sein:

- ▶ Bin Packing ( $\leq 2x$  Optimum)

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- ▶ Traveling Salesman Problem (TSP)

Greedy kann gut sein:

- ▶ Bin Packing ( $\leq 2x$  Optimum)

Greedy kann optimal sein:

- ▶ Minimaler Spannbaum, Kürzester-Weg-Problem.

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- ▶ Traveling Salesman Problem (TSP)

Greedy kann gut sein:

- ▶ Bin Packing ( $\leq 2x$  Optimum)

Greedy kann optimal sein:

- ▶ Minimaler Spannbaum, Kürzester-Weg-Problem.

# Greedy?

## Beispiel

Greedy kann beliebig schlecht werden:

- ▶ Traveling Salesman Problem (TSP)

Greedy kann gut sein:

- ▶ Bin Packing ( $\leq 2x$  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 den kostengünstigsten Weg, um eine Menge von Flughafenterminals, Städten, ... zu verbinden.

# Anwendungen

## Problem

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

## Beispiel

- ▶ Finde den 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 den 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

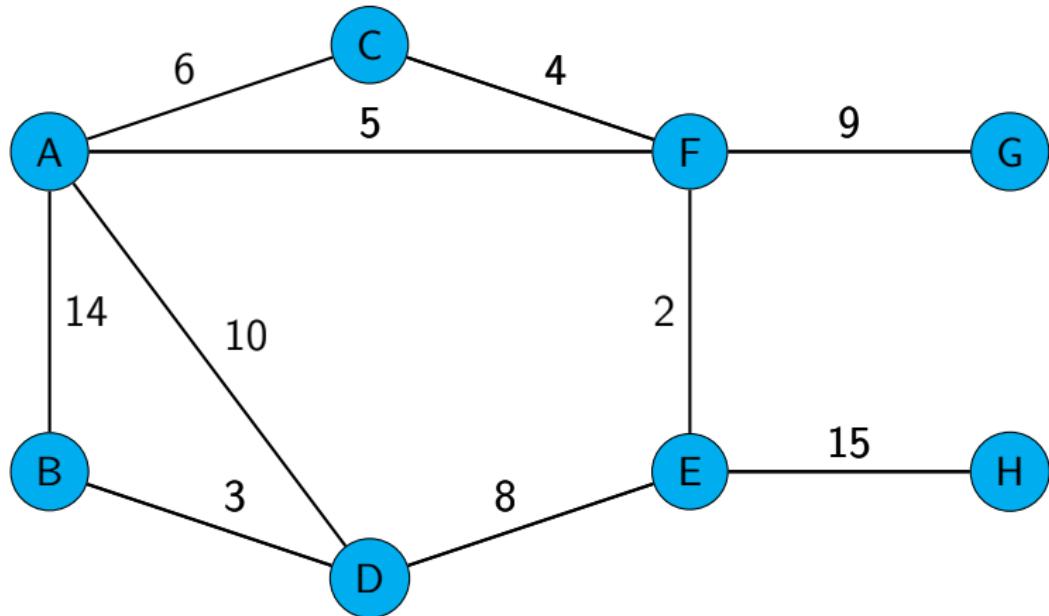
## Problem

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

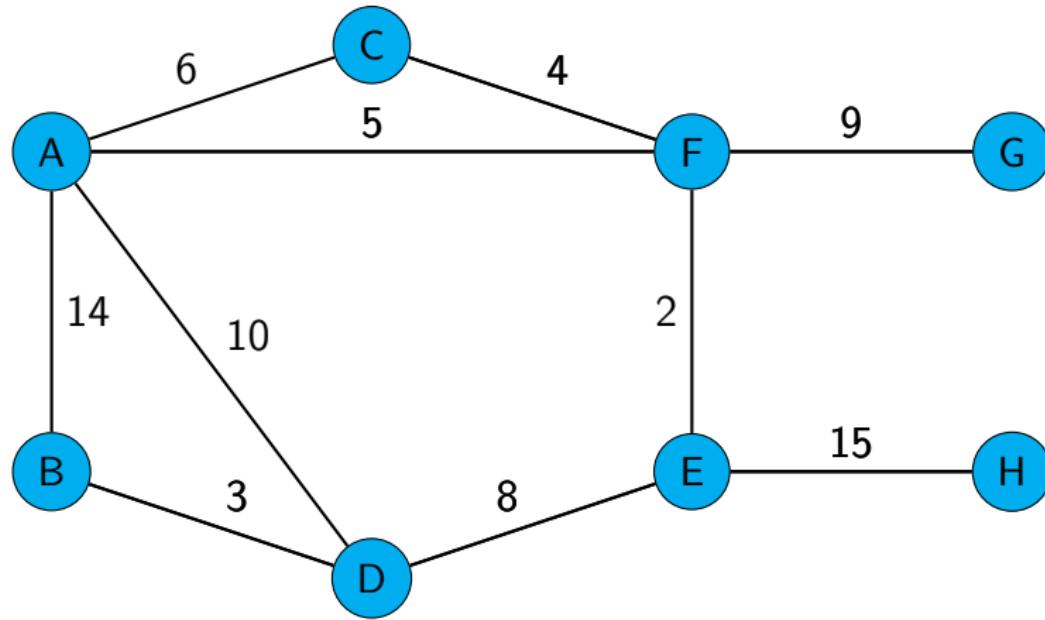
## Beispiel

- ▶ Finde den 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

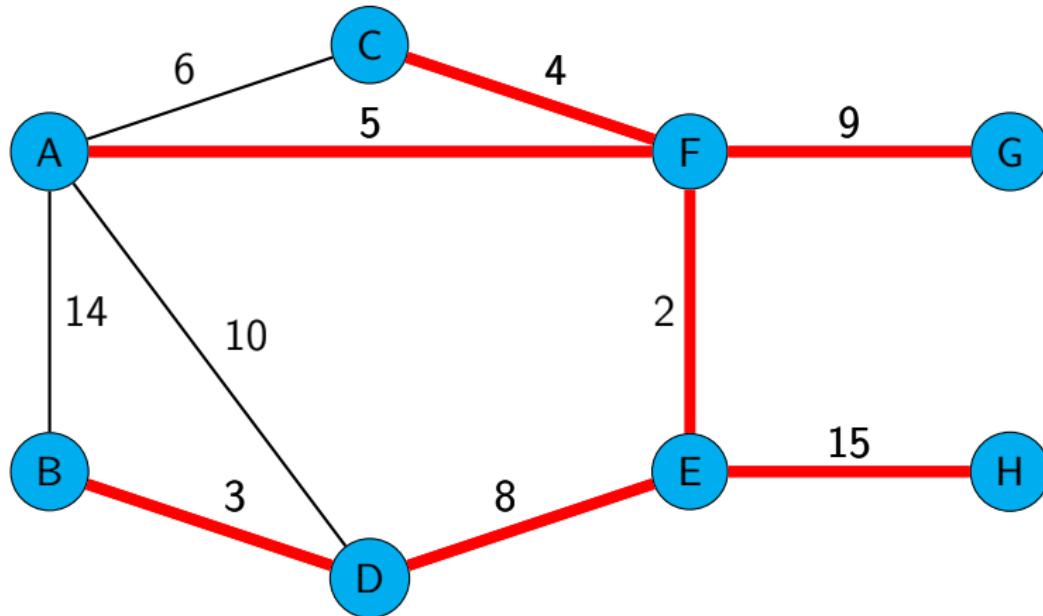


# Minimaler Spannbaum – Beispiel



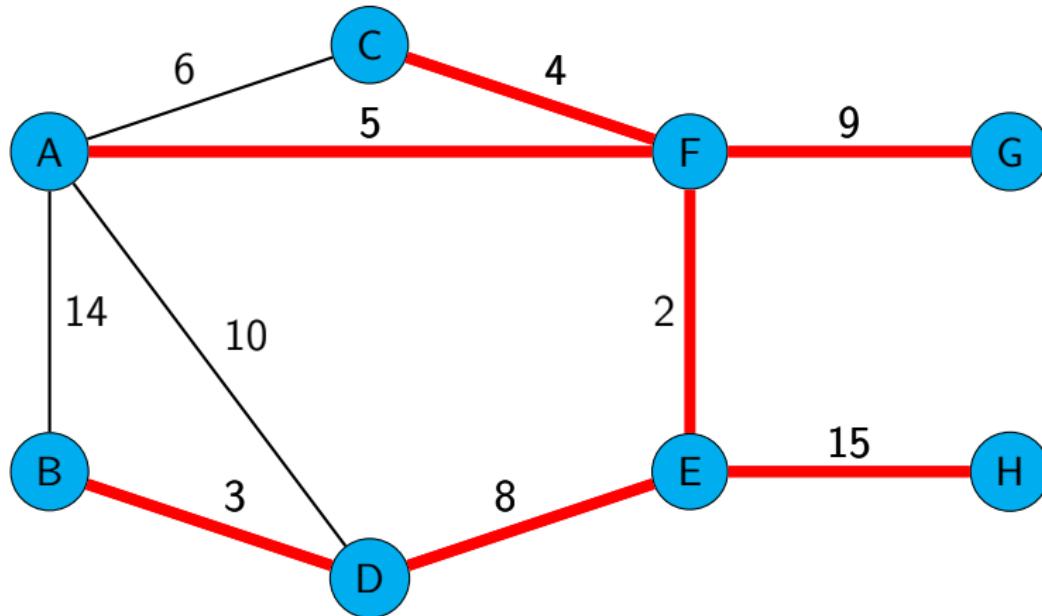
Was ist ein minimaler Spannbaum?

# Minimaler Spannbaum – Beispiel



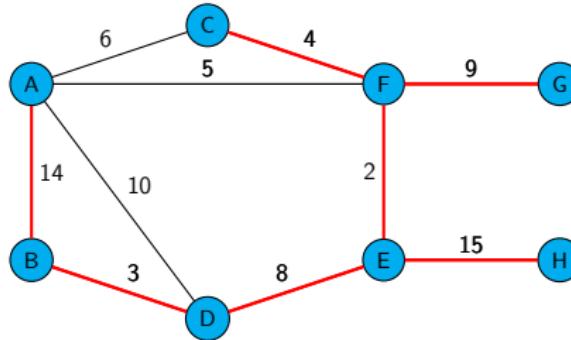
Das ist *ein* minimaler Spannbaum (mit Gesamtgewicht 46).

# Minimaler Spannbaum – Beispiel

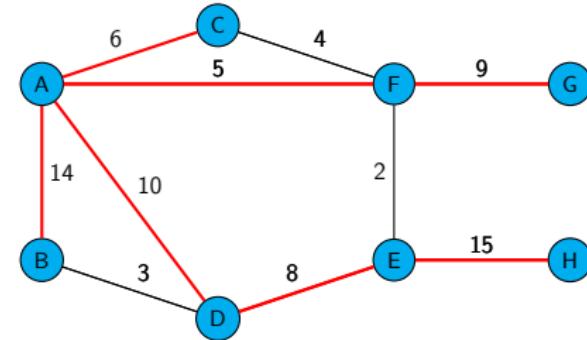


Das ist *ein* minimaler Spannbaum (mit Gesamtgewicht 46).  
In diesem Fall ist es auch der einzige.

# 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 (BLACK, GRAY, WHITE) 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 (BLACK, GRAY, WHITE) 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 (BLACK, GRAY, WHITE) 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 einem Baum aus nur einem Knoten an, indem ein beliebiger Knoten des Graphens ausgewählt wird.

# Der Algorithmus von Prim – Übersicht

Wir ordnen die Knoten in drei Kategorien (BLACK, GRAY, WHITE) 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 einem Baum aus 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 (BLACK, GRAY, WHITE) 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 einem Baum aus 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 (BLACK, GRAY, WHITE) 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 einem Baum aus 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 (BLACK, GRAY, WHITE) 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 einem Baum aus 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 (BLACK, GRAY, WHITE) 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 einem Baum aus 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?

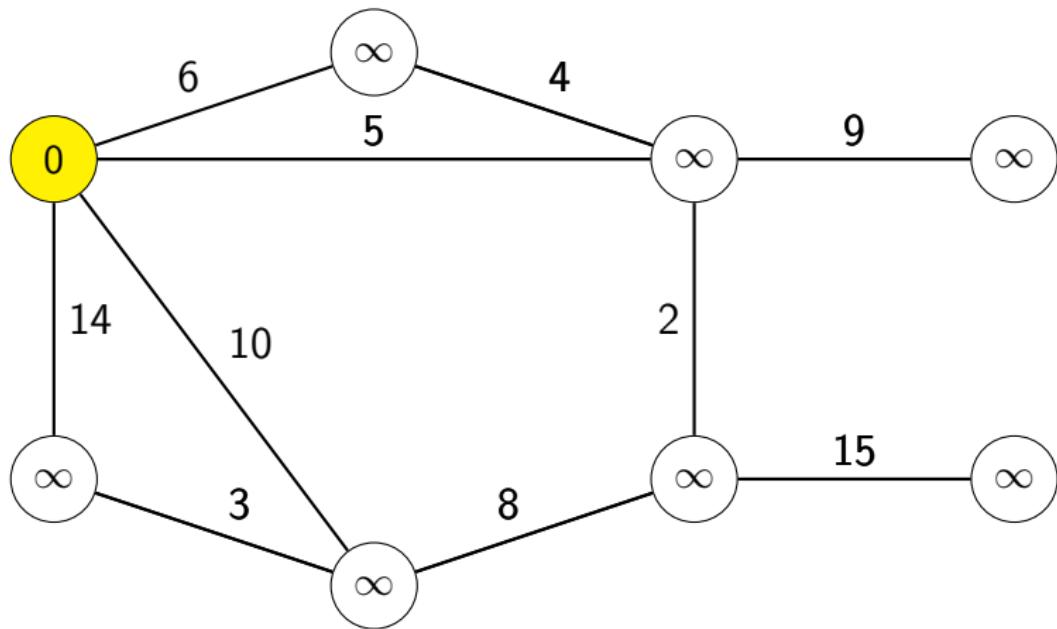
# Prim's Algorithmus – Grundgerüst

---

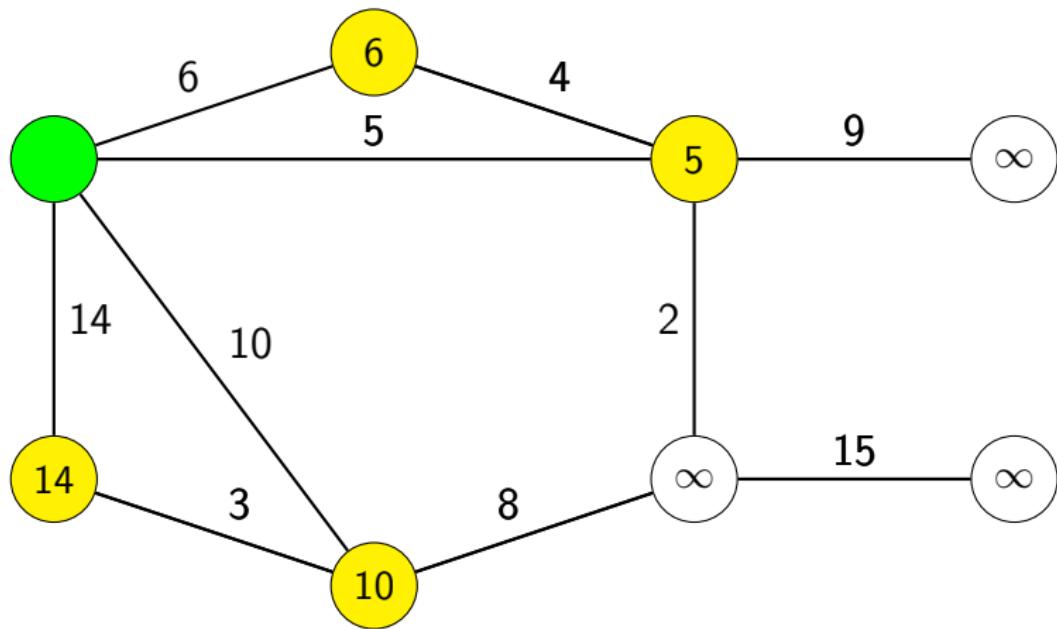
```
1 // ungerichteter Graph G mit n Knoten
2 void primMST(Graph G, int n) {
3     initialisiere alle Knoten als ungesehen (WHITE);
4     wähle irgendeinen Knoten s und markiere ihn mit Baum (BLACK);
5     reklassifiziere alle zu s adjazenten Knoten als Rand (GRAY);
6     while (es gibt Randknoten) {
7         wähle von allen Kanten zwischen einem Baumknoten t und
8             einem Randknoten v die billigste;
9         reklassifiziere v als Baum (BLACK);
10        füge Kante tv zum Baum hinzu;
11        reklassifiziere alle zu v adjazenten ungesesehenen Knoten
12            als Rand (GRAY);
13    }
14 }
```

---

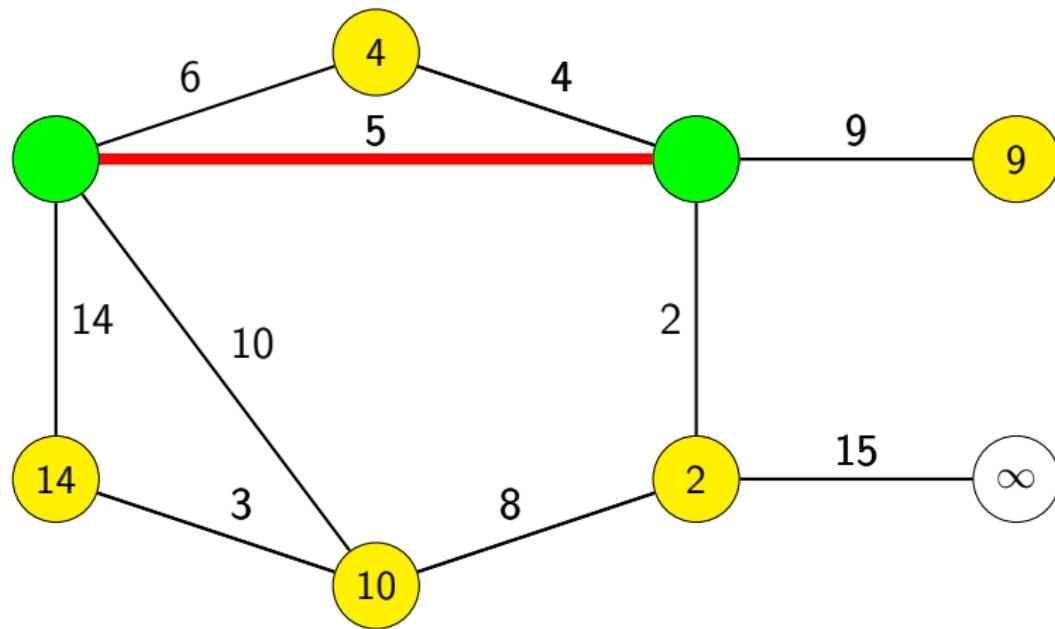
# Prim's Algorithmus – Beispiel



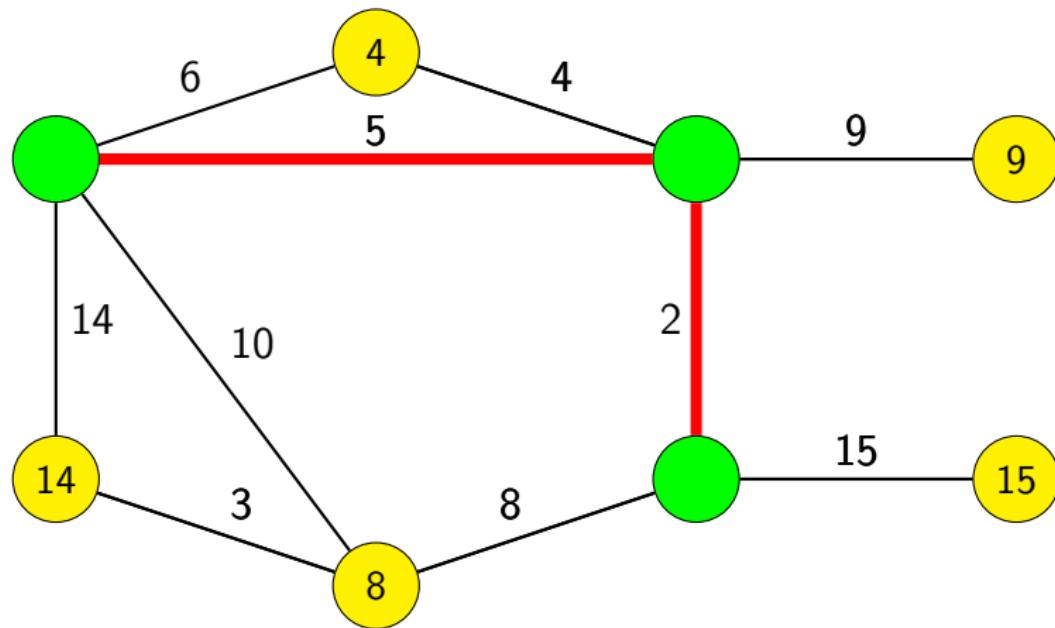
# Prim's Algorithmus – Beispiel



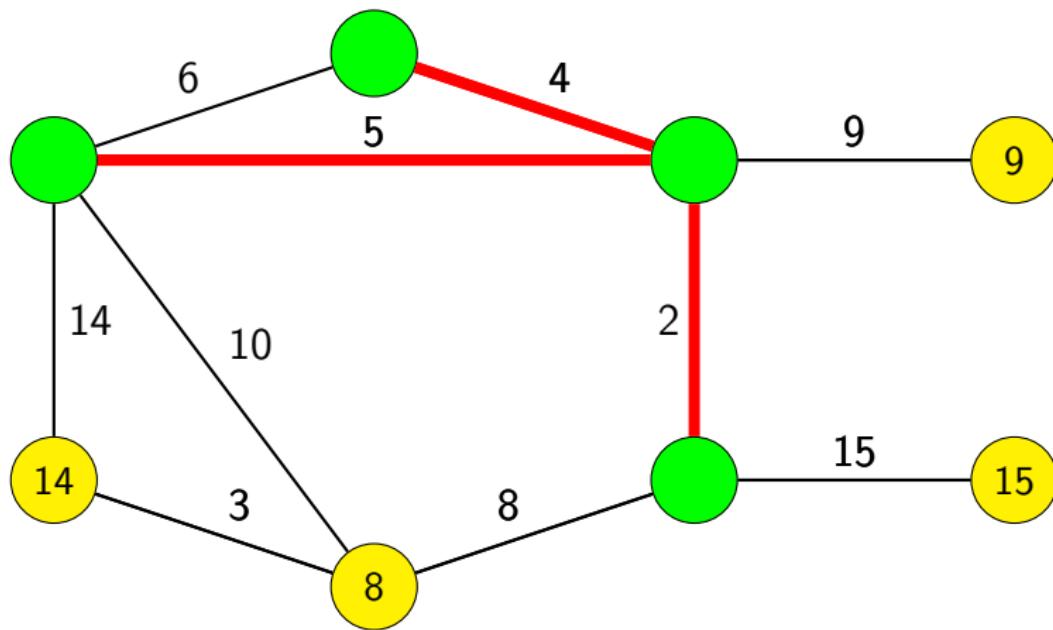
# Prim's Algorithmus – Beispiel



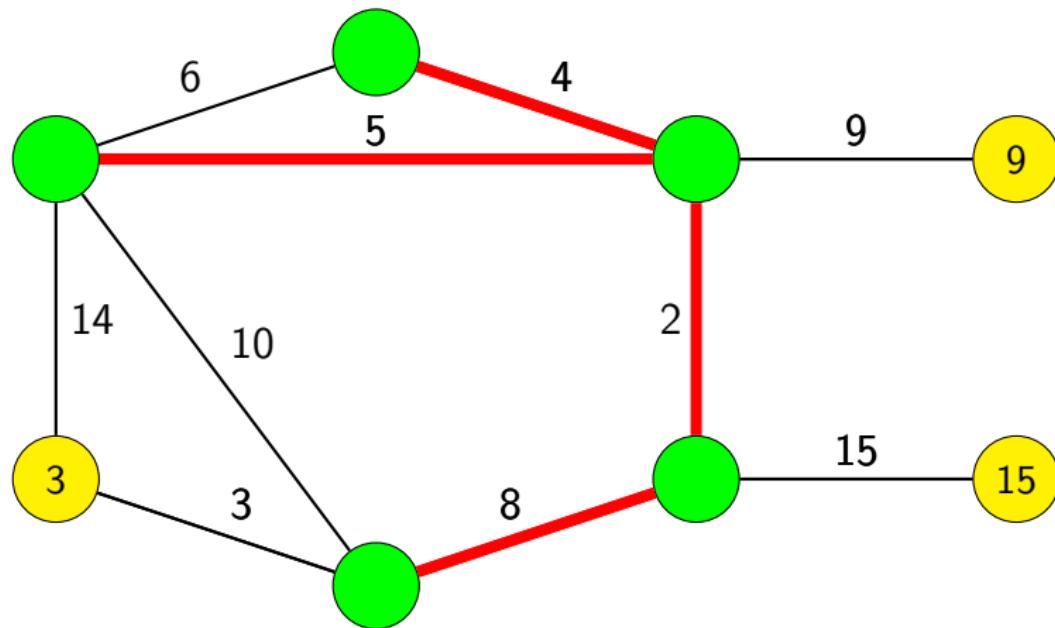
# Prim's Algorithmus – Beispiel



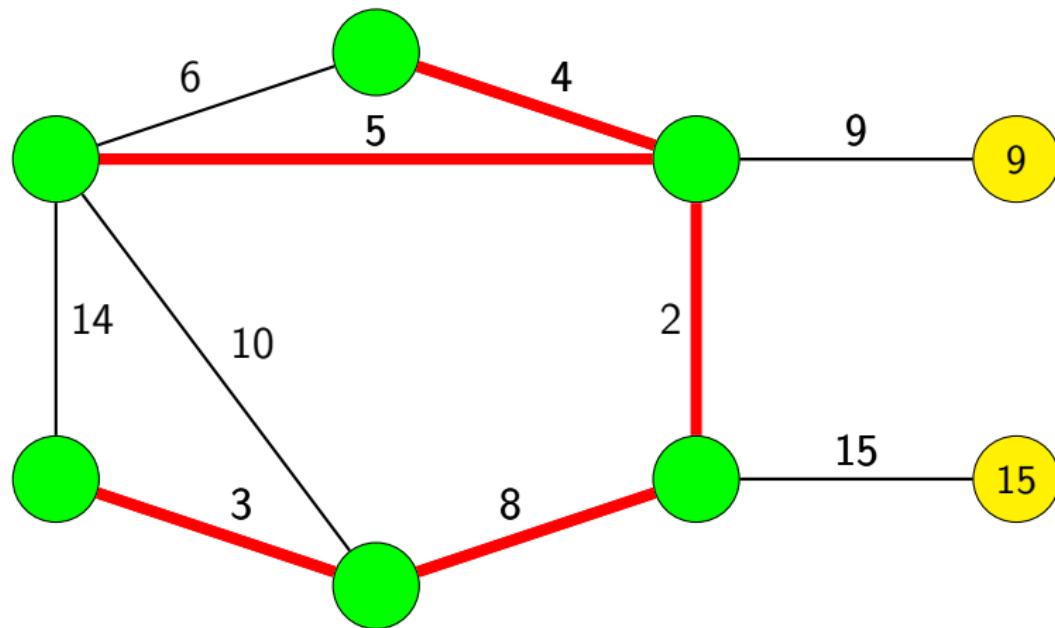
# Prim's Algorithmus – Beispiel



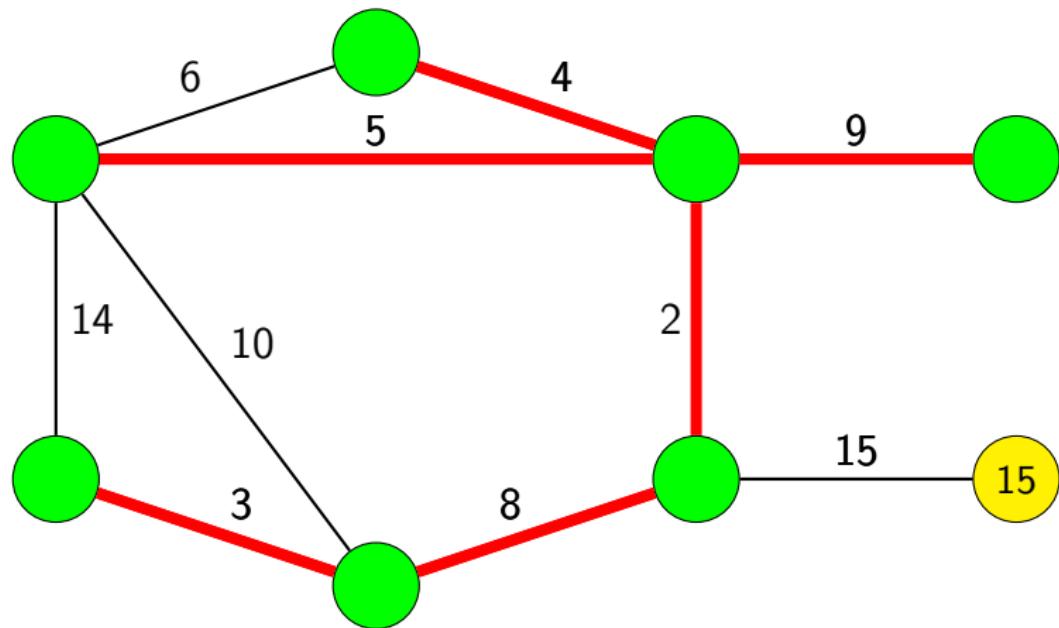
# Prim's Algorithmus – Beispiel



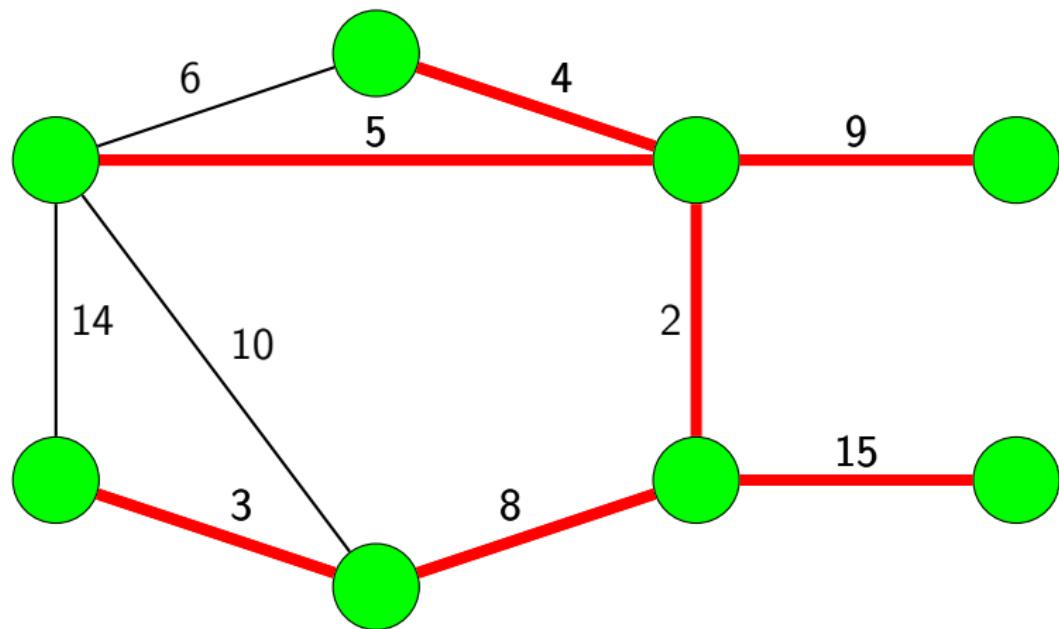
# Prim's Algorithmus – Beispiel



# Prim's Algorithmus – Beispiel



# Prim's Algorithmus – Beispiel



# Die MST-Eigenschaft

## MST-Eigenschaft auf $G$

Ein Spannbaum  $T$  hat die **minimale-Spannbaum-Eigenschaft auf  $G$** , wenn

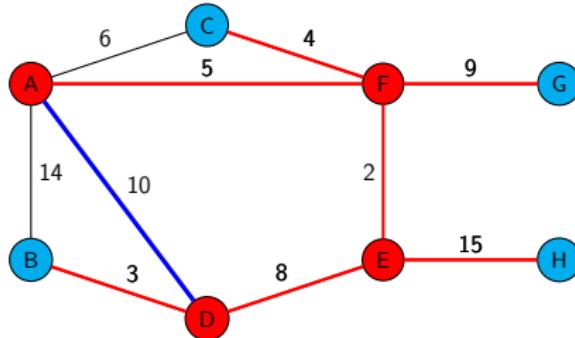
1. jede Kante  $(u, v) \in G - T$  einen Zyklus in  $T$  erzeugen würde, und
2. in diesem Zyklus die Kante mit maximalem Gewicht ist.

# Die MST-Eigenschaft

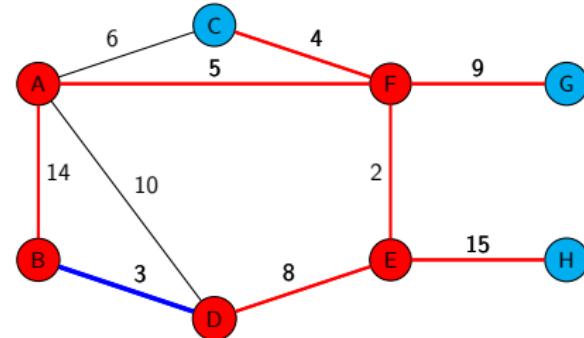
## MST-Eigenschaft auf $G$

Ein Spannbaum  $T$  hat die **minimale-Spannbaum-Eigenschaft auf  $G$** , wenn

1. jede Kante  $(u, v) \in G - T$  einen Zyklus in  $T$  erzeugen würde, und
2. in diesem Zyklus die Kante mit maximalem Gewicht ist.



Spannbaum, der die MST-Eigenschaft hat



Spannbaum, der die MST-Eigenschaft verletzt

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

## Beweis.

Induktion über die Anzahl  $k$  an Kanten in  $T_1 - T_2$ .

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

## Beweis.

Induktion über die Anzahl  $k$  an Kanten in  $T_1 - T_2$ .

Induktionsanfang:

$k = 0$ ,  $T_1$  und  $T_2$  sind gleich, daher ist  $W(T_1) = W(T_2)$ .

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

## Beweis.

Induktion über die Anzahl  $k$  an Kanten in  $T_1 - T_2$ .

### Induktionsanfang:

$k = 0$ ,  $T_1$  und  $T_2$  sind gleich, daher ist  $W(T_1) = W(T_2)$ .

### Induktionsschritt:

$k > 0$ , Angenommen das Lemma gilt für Spannbäume, die sich nur in  $j < k$  Kanten unterscheiden.

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

## Beweis.

Induktion über die Anzahl  $k$  an Kanten in  $T_1 - T_2$ .

### Induktionsanfang:

$k = 0$ ,  $T_1$  und  $T_2$  sind gleich, daher ist  $W(T_1) = W(T_2)$ .

### Induktionsschritt:

$k > 0$ , Angenommen das Lemma gilt für Spannbäume, die sich nur in  $j < k$  Kanten unterscheiden.

- ▶  $T_1$  und  $T_2$  unterscheiden sich nun um  $k$  Kanten.

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

## Beweis.

Induktion über die Anzahl  $k$  an Kanten in  $T_1 - T_2$ .

### Induktionsanfang:

$k = 0$ ,  $T_1$  und  $T_2$  sind gleich, daher ist  $W(T_1) = W(T_2)$ .

### Induktionsschritt:

$k > 0$ , Angenommen das Lemma gilt für Spannbäume, die sich nur in  $j < k$  Kanten unterscheiden.

- ▶  $T_1$  und  $T_2$  unterscheiden sich nun um  $k$  Kanten.
- ▶ Betrachte die günstigste Kante  $(u, v)$  aus  $T_2 - T_1$ .

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

## Beweis.

Induktion über die Anzahl  $k$  an Kanten in  $T_1 - T_2$ .

### Induktionsanfang:

$k = 0$ ,  $T_1$  und  $T_2$  sind gleich, daher ist  $W(T_1) = W(T_2)$ .

### Induktionsschritt:

$k > 0$ , Angenommen das Lemma gilt für Spannbäume, die sich nur in  $j < k$  Kanten unterscheiden.

- ▶  $T_1$  und  $T_2$  unterscheiden sich nun um  $k$  Kanten.
- ▶ Betrachte die günstigste Kante  $(u, v)$  aus  $T_2 - T_1$ .
- ▶ Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten.

# Spannbäume mit gleichem Gewicht

## Lemma

Wenn zwei Spannbäume  $T_1$  und  $T_2$  die MST-Eigenschaft auf  $G$  haben, dann ist  $W(T_1) = W(T_2)$ .

## Beweis.

Induktion über die Anzahl  $k$  an Kanten in  $T_1 - T_2$ .

### Induktionsanfang:

$k = 0$ ,  $T_1$  und  $T_2$  sind gleich, daher ist  $W(T_1) = W(T_2)$ .

### Induktionsschritt:

$k > 0$ , Angenommen das Lemma gilt für Spannbäume, die sich nur in  $j < k$  Kanten unterscheiden.

- ▶  $T_1$  und  $T_2$  unterscheiden sich nun um  $k$  Kanten.
- ▶ Betrachte die günstigste Kante  $(u, v)$  aus  $T_2 - T_1$ .
- ▶ Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:



# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .
  - Wegen der MST-Eigenschaft von  $T_2$  gilt  $W(u, v) \leq W(w, x)$ .

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .
  - Wegen der MST-Eigenschaft von  $T_2$  gilt  $W(u, v) \leq W(w, x)$ . $\Rightarrow W(w, x) = W(u, v)$ .

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .
  - Wegen der MST-Eigenschaft von  $T_2$  gilt  $W(u, v) \leq W(w, x)$ . $\Rightarrow W(w, x) = W(u, v)$ .
- Füge  $(u, v)$  zu  $T_1$  hinzu und entferne  $(w, x)$ ;

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .
  - Wegen der MST-Eigenschaft von  $T_2$  gilt  $W(u, v) \leq W(w, x)$ .
  - ⇒  $W(w, x) = W(u, v)$ .
- Füge  $(u, v)$  zu  $T_1$  hinzu und entferne  $(w, x)$ ;  
Wir erhalten  $T'_1$  mit  $W(T_1) = W(T'_1)$ .

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .
  - Wegen der MST-Eigenschaft von  $T_2$  gilt  $W(u, v) \leq W(w, x)$ .
  - ⇒  $W(w, x) = W(u, v)$ .
- Füge  $(u, v)$  zu  $T_1$  hinzu und entferne  $(w, x)$ ;  
Wir erhalten  $T'_1$  mit  $W(T_1) = W(T'_1)$ .
- Bemerke, daß  $T'_1$  die MST-Eigenschaft hat, da  $T_1$  die hatte.

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .
  - Wegen der MST-Eigenschaft von  $T_2$  gilt  $W(u, v) \leq W(w, x)$ . $\Rightarrow W(w, x) = W(u, v)$ .
- Füge  $(u, v)$  zu  $T_1$  hinzu und entferne  $(w, x)$ ;  
Wir erhalten  $T'_1$  mit  $W(T_1) = W(T'_1)$ .
- Bemerke, daß  $T'_1$  die MST-Eigenschaft hat, da  $T_1$  die hatte.
- Da  $T'_1$  die MST-Eigenschaft hat und  $T'_1$  und  $T_2$  sich nur noch um  $k-1$  Kanten unterscheiden:

# Spannbäume mit gleichem Gewicht – Beweis

## Beweis. (Forts.)

- ▶ Der Pfad von  $u$  nach  $v$  in  $T_1$  (mit Länge  $\geq 2$ ) muss eine Kante  $(w, x) \notin T_2$  enthalten. Es gilt  $W(w, x) = W(u, v)$ , denn:
  - Wegen der MST-Eigenschaft von  $T_1$  gilt  $W(w, x) \leq W(u, v)$ .
  - Wegen der MST-Eigenschaft von  $T_2$  gilt  $W(u, v) \leq W(w, x)$ .
  - ⇒  $W(w, x) = W(u, v)$ .
- ▶ Füge  $(u, v)$  zu  $T_1$  hinzu und entferne  $(w, x)$ ;  
Wir erhalten  $T'_1$  mit  $W(T_1) = W(T'_1)$ .
- ▶ Bemerke, daß  $T'_1$  die MST-Eigenschaft hat, da  $T_1$  die hatte.
- ▶ Da  $T'_1$  die MST-Eigenschaft hat und  $T'_1$  und  $T_2$  sich nur noch um  $k-1$  Kanten unterscheiden:
  - ⇒ mit Induktionsannahme folgt, dass  $W(T'_1) = W(T_2)$  und damit  $W(T_1) = W(T_2)$ .



# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch.

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ .

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt,

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus,

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ .

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ . Das Ersetzen von  $(x, y)$  durch  $(u, v)$  in  $T$  liefert den Spannbaum  $T'$  mit  $W(T') < W(T)$ .

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ . Das Ersetzen von  $(x, y)$  durch  $(u, v)$  in  $T$  liefert den Spannbaum  $T'$  mit  $W(T') < W(T)$ . Also kann  $T$  kein MST gewesen sein. Widerspruch.

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ . Das Ersetzen von  $(x, y)$  durch  $(u, v)$  in  $T$  liefert den Spannbaum  $T'$  mit  $W(T') < W(T)$ . Also kann  $T$  kein MST gewesen sein. Widerspruch.

( $\Leftarrow$ ) Angenommen  $T$  hat die MST-Eigenschaft.

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ . Das Ersetzen von  $(x, y)$  durch  $(u, v)$  in  $T$  liefert den Spannbaum  $T'$  mit  $W(T') < W(T)$ . Also kann  $T$  kein MST gewesen sein. Widerspruch.

( $\Leftarrow$ ) Angenommen  $T$  hat die MST-Eigenschaft. Sei  $T'$  ein MST von  $G$ .

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ . Das Ersetzen von  $(x, y)$  durch  $(u, v)$  in  $T$  liefert den Spannbaum  $T'$  mit  $W(T') < W(T)$ . Also kann  $T$  kein MST gewesen sein. Widerspruch.

( $\Leftarrow$ ) Angenommen  $T$  hat die MST-Eigenschaft. Sei  $T'$  ein MST von  $G$ . Wegen  $\Rightarrow$  hat dann  $T'$  die MST-Eigenschaft.

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ . Das Ersetzen von  $(x, y)$  durch  $(u, v)$  in  $T$  liefert den Spannbaum  $T'$  mit  $W(T') < W(T)$ . Also kann  $T$  kein MST gewesen sein. Widerspruch.

( $\Leftarrow$ ) Angenommen  $T$  hat die MST-Eigenschaft. Sei  $T'$  ein MST von  $G$ . Wegen  $\Rightarrow$  hat dann  $T'$  die MST-Eigenschaft. Mit dem vorigen Lemma haben Spannbäume mit MST-Eigenschaft das selbe Gewicht, also:  $W(T) = W(T')$ .

# Theorem

## Theorem

*Ein Baum ist ein minimaler Spannbaum gdw. er die MST-Eigenschaft hat.*

## Beweis.

( $\Rightarrow$ ) Durch Widerspruch. Sei  $T$  ein MST von  $G$ . Nehme an, daß  $T$  die MST-Eigenschaft verletzt, d. h., das Hinzufügen von der Kante  $(u, v) \notin T$  zu  $T$  erzeugt einen Zyklus, so dass für  $(x, y) \in T$  aus dem Zyklus  $W(u, v) < W(x, y)$ . Das Ersetzen von  $(x, y)$  durch  $(u, v)$  in  $T$  liefert den Spannbaum  $T'$  mit  $W(T') < W(T)$ . Also kann  $T$  kein MST gewesen sein. Widerspruch.

( $\Leftarrow$ ) Angenommen  $T$  hat die MST-Eigenschaft. Sei  $T'$  ein MST von  $G$ . Wegen  $\Rightarrow$  hat dann  $T'$  die MST-Eigenschaft. Mit dem vorigen Lemma haben Spannbäume mit MST-Eigenschaft das selbe Gewicht, also:  $W(T) = W(T')$ . Also ist auch  $T$  ein MST. □

# Korrektheit von Prim's Algorithmus

## Theorem

Der vom Prim's Algorithmus erzeugte Spannbaum  $T_k$  mit  $k > 0$  Knoten ( $k = 1, \dots, n$ ) hat die MST-Eigenschaft auf dem durch die Knoten in  $T_k$  induzierten Teilgraph  $G_k$  (d.h.  $(u, v)$  ist eine Kante in  $G_k$  wenn  $(u, v)$  eine Kante in  $G$  ist, und  $u$  und  $v$  sind in  $T_k$ ).

# Korrektheit von Prim's Algorithmus

## Theorem

Der vom Prim's Algorithmus erzeugte Spannbaum  $T_k$  mit  $k > 0$  Knoten ( $k = 1, \dots, n$ ) hat die MST-Eigenschaft auf dem durch die Knoten in  $T_k$  induzierten Teilgraph  $G_k$  (d.h.  $(u, v)$  ist eine Kante in  $G_k$  wenn  $(u, v)$  eine Kante in  $G$  ist, und  $u$  und  $v$  sind in  $T_k$ ).

## Beweis.

Induktion nach  $k$ .

# Korrektheit von Prim's Algorithmus

## Theorem

Der vom Prim's Algorithmus erzeugte Spannbaum  $T_k$  mit  $k > 0$  Knoten ( $k = 1, \dots, n$ ) hat die MST-Eigenschaft auf dem durch die Knoten in  $T_k$  induzierten Teilgraph  $G_k$  (d.h.  $(u, v)$  ist eine Kante in  $G_k$  wenn  $(u, v)$  eine Kante in  $G$  ist, und  $u$  und  $v$  sind in  $T_k$ ).

## Beweis.

Induktion nach  $k$ .

### Induktionsanfang:

$k = 1$ ,  $T_1$  und  $G_1$  enthalten nur Knoten und keine Kanten.  $T_1$  hat damit die MST-Eigenschaft in  $G_1$ .



# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

$k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

- $k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .
- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

- $k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .
- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
  - ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

- $k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .
- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
  - ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .
  - ▶ Sei  $(u_1, v)$  die günstigste dieser Kanten, die in  $T_k$  gewählt wurde.

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

$k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .

- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
- ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .
- ▶ Sei  $(u_1, v)$  die günstigste dieser Kanten, die in  $T_k$  gewählt wurde.
- ▶ Betrachte die Kante  $(x, y) \in G_k - T_k$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

- $k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .
- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
  - ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .
  - ▶ Sei  $(u_1, v)$  die günstigste dieser Kanten, die in  $T_k$  gewählt wurde.
  - ▶ Betrachte die Kante  $(x, y) \in G_k - T_k$ .
    1. Sei  $x \neq v$  und  $y \neq v$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

- $k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .
- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
  - ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .
  - ▶ Sei  $(u_1, v)$  die günstigste dieser Kanten, die in  $T_k$  gewählt wurde.
  - ▶ Betrachte die Kante  $(x, y) \in G_k - T_k$ .
    1. Sei  $x \neq v$  und  $y \neq v$ . Dann  $(x, y) \in G_{k-1} - T_{k-1}$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

- $k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .
- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
  - ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .
  - ▶ Sei  $(u_1, v)$  die günstigste dieser Kanten, die in  $T_k$  gewählt wurde.
  - ▶ Betrachte die Kante  $(x, y) \in G_k - T_k$ .
    1. Sei  $x \neq v$  und  $y \neq v$ . Dann  $(x, y) \in G_{k-1} - T_{k-1}$ . Hinzufügen von  $(x, y)$  zu  $T_{k-1}$  liefert einen Zyklus, mit (nach Ind. Annahme)  $(x, y)$  maximalem Gewicht auf dem Zyklus.

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

- $k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .
- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
  - ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .
  - ▶ Sei  $(u_1, v)$  die günstigste dieser Kanten, die in  $T_k$  gewählt wurde.
  - ▶ Betrachte die Kante  $(x, y) \in G_k - T_k$ .
    1. Sei  $x \neq v$  und  $y \neq v$ . Dann  $(x, y) \in G_{k-1} - T_{k-1}$ . Hinzufügen von  $(x, y)$  zu  $T_{k-1}$  liefert einen Zyklus, mit (nach Ind. Annahme)  $(x, y)$  maximalem Gewicht auf dem Zyklus. Dies ist jedoch der Zykel den es auch in  $T_k$  gibt.

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

### Induktionsschritt:

$k > 1$ . Angenommen  $T_j$  hat die MST-Eigenschaft auf  $G_j$  für  $j < k$ .

- ▶ Sei  $v \in T_k - T_{k-1}$  der  $k$ -te Knoten, der hinzugefügt wurde und
- ▶  $(u_1, v), \dots, (u_m, v)$  die Kanten zwischen Knoten in  $T_{k-1}$  und  $v$ .
- ▶ Sei  $(u_1, v)$  die günstigste dieser Kanten, die in  $T_k$  gewählt wurde.
- ▶ Betrachte die Kante  $(x, y) \in G_k - T_k$ .
  1. Sei  $x \neq v$  und  $y \neq v$ . Dann  $(x, y) \in G_{k-1} - T_{k-1}$ . Hinzufügen von  $(x, y)$  zu  $T_{k-1}$  liefert einen Zyklus, mit (nach Ind. Annahme)  $(x, y)$  maximalem Gewicht auf dem Zyklus. Dies ist jedoch der Zyklus den es auch in  $T_k$  gibt. Deswegen hat  $T_k$  die MST-Eigenschaft auf  $G_k$ .
  2. siehe nächste Folie.



# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow{w_1} w_2 \dots \xrightarrow{w_\ell} u_i$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow{w_1} w_2 \dots \xrightarrow{w_\ell} u_i$ . Nimm an, dass  $(w_j, w_{j+1})$  die erste Kante auf diesem Pfad ist mit  $W(w_j, w_{j+1}) > W(u_i, v)$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow[w_1]{=} u_1 w_2 \dots \xrightarrow[w_\ell]{=} u_i$ . Nimm an, dass  $(w_j, w_{j+1})$  die erste Kante auf diesem Pfad ist mit  $W(w_j, w_{j+1}) > W(u_i, v)$ . Sei  $(w_{p-1}, w_p)$  die letzte Kante auf dem Pfad mit  $W(w_{p-1}, w_p) > W(u_i, v)$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow[w_1]{=} u_1 w_2 \dots \xrightarrow[w_\ell]{=} u_i$ . Nimm an, dass

$(w_j, w_{j+1})$  die erste Kante auf diesem Pfad ist mit  $W(w_j, w_{j+1}) > W(u_i, v)$ . Sei  $(w_{p-1}, w_p)$  die letzte Kante auf dem Pfad mit  $W(w_{p-1}, w_p) > W(u_i, v)$ . (Möglichlicherweise gilt  $p = j+1$ .)

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow[w_1]{=} u_1 w_2 \dots \xrightarrow[w_\ell]{=} u_i$ . Nimm an, dass

$(w_j, w_{j+1})$  die erste Kante auf diesem Pfad ist mit  $W(w_j, w_{j+1}) > W(u_i, v)$ . Sei  $(w_{p-1}, w_p)$  die letzte Kante auf dem Pfad mit  $W(w_{p-1}, w_p) > W(u_i, v)$ . (Möglichlicherweise gilt  $p = j+1$ .) Wir zeigen, dass  $w_j$  und  $w_p$  nicht in  $T_{k-1}$  existieren können. (Siehe Vorlesung.)

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow[w_1]{=} u_1 w_2 \dots \xrightarrow[w_\ell]{=} u_i$ . Nimm an, dass

$(w_j, w_{j+1})$  die erste Kante auf diesem Pfad ist mit  $W(w_j, w_{j+1}) > W(u_i, v)$ . Sei  $(w_{p-1}, w_p)$  die letzte Kante auf dem Pfad mit  $W(w_{p-1}, w_p) > W(u_i, v)$ . (Möglichlicherweise gilt  $p = j+1$ .) Wir zeigen, dass  $w_j$  und  $w_p$  nicht in  $T_{k-1}$  existieren können. (Siehe Vorlesung.) Damit hat keine Kante auf dem Pfad  $w_1 \dots w_\ell$  ein größeres Gewicht als  $W(u_i, v)$ ,

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow{w_1} w_2 \dots \xrightarrow{w_\ell} u_i$ . Nimm an, dass  $(w_j, w_{j+1})$  die erste Kante auf diesem Pfad ist mit  $W(w_j, w_{j+1}) > W(u_i, v)$ . Sei  $(w_{p-1}, w_p)$  die letzte Kante auf dem Pfad mit  $W(w_{p-1}, w_p) > W(u_i, v)$ . (Möglichlicherweise gilt  $p = j+1$ .) Wir zeigen, dass  $w_j$  und  $w_p$  nicht in  $T_{k-1}$  existieren können. (Siehe Vorlesung.) Damit hat keine Kante auf dem Pfad  $w_1 \dots w_\ell$  ein größeres Gewicht als  $W(u_i, v)$ , und damit auch nicht größer als  $W(u_1, v)$ .

# Korrektheit von Prim's Algorithmus – Beweis

## Beweis. (Forts.)

Induktionsschritt:

2.  $(x, y) \in \{(u_2, v), \dots, (u_m, v)\}$  mit  $m > 2$ . Betrachte den Pfad von  $v$  nach  $u_i$  ( $1 < i \leq m$ ) in  $T_k$ :  $v \xrightarrow[w_1]{=} u_1 w_2 \dots \xrightarrow[w_\ell]{=} u_i$ . Nimm an, dass  $(w_j, w_{j+1})$  die erste Kante auf diesem Pfad ist mit  $W(w_j, w_{j+1}) > W(u_i, v)$ . Sei  $(w_{p-1}, w_p)$  die letzte Kante auf dem Pfad mit  $W(w_{p-1}, w_p) > W(u_i, v)$ . (Möglichlicherweise gilt  $p = j+1$ .) Wir zeigen, dass  $w_j$  und  $w_p$  nicht in  $T_{k-1}$  existieren können. (Siehe Vorlesung.) Damit hat keine Kante auf dem Pfad  $w_1 \dots w_\ell$  ein größeres Gewicht als  $W(u_i, v)$ , und damit auch nicht größer als  $W(u_1, v)$ . Also, hat  $T_k$  die MST-Eigenschaft.



# Korrektheit

## Korrektheit

Der Algorithmus von Prim bestimmt einen minimalen Spannbaum.

# Korrektheit

## Korrektheit

Der Algorithmus von Prim bestimmt einen minimalen Spannbaum.

## Beweis.

Sei  $G = (V, E)$  und  $|V| = n$  Anzahl der Knoten.

# Korrektheit

## Korrektheit

Der Algorithmus von Prim bestimmt einen minimalen Spannbaum.

## Beweis.

Sei  $G = (V, E)$  und  $|V| = n$  Anzahl der Knoten.

- Sei  $T_n$  der durch Prim berechnete Baum. Dann ist  $G_n = G$ , und es folgt – siehe letztes Theorem – dass  $T_n$  die MST-Eigenschaft hat.

# Korrektheit

## Korrektheit

Der Algorithmus von Prim bestimmt einen minimalen Spannbaum.

## Beweis.

Sei  $G = (V, E)$  und  $|V| = n$  Anzahl der Knoten.

- ▶ Sei  $T_n$  der durch Prim berechnete Baum. Dann ist  $G_n = G$ , und es folgt – siehe letztes Theorem – dass  $T_n$  die MST-Eigenschaft hat.
- ▶ Da  $T_n$  die MST-Eigenschaft hat gdw. es ein MST ist, folgt:  $T_n$  ist ein MST.



# ADT zum Vorhalten der Randknoten (I)

Die benötigten Operationen für den Algorithmus von Prim sind:

# ADT zum Vorhalten der Randknoten (I)

Die benötigten Operationen für den Algorithmus von Prim sind:

- ▶ Wähle eine billigste Kante zu einem Randknoten (Kantenkandidat).

# ADT zum Vorhalten der Randknoten (I)

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).

# ADT zum Vorhalten der Randknoten (I)

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 (I)

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 (I)

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).*

## Prioritätswarteschlange (priority queue)

- ▶ `PriorityQueue` `pq`;
- ▶ `pq.insert(int e, int k)`, `int pq.getMin()`, `pq.delMin()`
- ▶ `void pq.decrKey(int e, int k)` setzt den Schlüssel von Element `e` auf `k`; `k` muss kleiner als der bisherige Schlüssel von `e` sein.

# ADT zum Vorhalten der Randknoten (I)

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).*

## Prioritätswarteschlange (priority queue)

- ▶ `PriorityQueue pq;`
  - ▶ `pq.insert(int e, int k), int pq.getMin(), pq.delMin()`
  - ▶ `void pq.decrKey(int e, int k)` setzt den Schlüssel von Element e auf k; k muss kleiner als der bisherige Schlüssel von e sein.
- ⇒ Wir entscheiden uns für die Prioritätswarteschlange als Datenstruktur für die Randknoten.

# Vorläufige Komplexitätsanalyse

Im Worst-Case:

# Vorläufige Komplexitätsanalyse

Im Worst-Case:

- ▶ **Jeder Knoten** muss zur Prioritätswarteschlange hinzugefügt 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.

# 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:

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

# 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:

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Welche Implementierung der Prioritätswarteschlange ist dafür gut geeignet?

# Drei Prioritätswarteschlangenimplementierungen

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

Operation	Implementierung		
	unsortiertes Array	sortiertes Array	Heap
isEmpty(pq)	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
insert(pq, e, k)	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$
getMin(pq)	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
delMin(pq)	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$
getElt(pq, k)	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$
decrKey(pq, e, k)	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$
Prim	$O(n^2 + m)$	$O(n^2 + m \cdot n)$	$O(n \log n + m \log n)$

# Drei Prioritätswarteschlangenimplementierungen

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

Operation	Implementierung		
	unsortiertes Array	sortiertes Array	Heap
isEmpty(pq)	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
insert(pq, e, k)	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$
getMin(pq)	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
delMin(pq)	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$
getElt(pq, k)	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$
decrKey(pq, e, k)	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$
Prim	$\Theta(n^2 + m)$	$\Theta(n^2 + m \cdot n)$	$\Theta(n \log n + m \log n)$

# Drei Prioritätswarteschlangenimplementierungen

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

Operation	Implementierung		
	unsortiertes Array	sortiertes Array	Heap
isEmpty(pq)	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
insert(pq, e, k)	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$
getMin(pq)	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
delMin(pq)	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$
getElt(pq, k)	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$
decrKey(pq, e, k)	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$
Prim	$\Theta(n^2 + m)$	$\Theta(n^2 + m \cdot n)$	$\Theta(n \log n + m \log n)$

- Wir ergänzen außerdem noch zwei Operationen:

# Prioritätswarteschlange, die Vierte (III)

## Prioritätswarteschlange

- `bool pq.isEmpty()`  $\Theta(n)$

# Prioritätswarteschlange, die Vierte (III)

## Prioritätswarteschlange

- ▶ `bool pq.isEmpty()`  $\Theta(n)$
- ▶ `void pq.insert(int elem, VertexState &key)`  $\Theta(1)$

# Prioritätswarteschlange, die Vierte (III)

## Prioritätswarteschlange

- ▶ `bool pq.isEmpty()`  $\Theta(n)$
- ▶ `void pq.insert(int elem, VertexState &key)`  $\Theta(1)$
- ▶ `float pq.getMin()`  $\Theta(n)$

# Prioritätswarteschlange, die Vierte (III)

## Prioritätswarteschlange

- ▶ `bool pq.isEmpty()`  $\Theta(n)$
- ▶ `void pq.insert(int elem, VertexState &key)`  $\Theta(1)$
- ▶ `float pq.getMin()`  $\Theta(n)$
- ▶ `void pq.delMin()`  $\Theta(n)$

# Prioritätswarteschlange, die Vierte (III)

## Prioritätswarteschlange

- ▶ `bool pq.isEmpty()`  $\Theta(n)$
- ▶ `void pq.insert(int elem, VertexState &key)`  $\Theta(1)$
- ▶ `float pq.getMin()`  $\Theta(n)$
- ▶ `void pq.delMin()`  $\Theta(n)$
- ▶ `void pq.decrKey(int elem, VertexState &newkey)` setzt den Schlüssel von `elem` auf `newkey`; `newkey.curWeight` muss kleiner als beim bisherigen Schlüssel von `elem` sein.  $\Theta(1)$

# Prioritätswarteschlange, die Vierte (III)

## Prioritätswarteschlange

- ▶ `bool pq.isEmpty()`  $\Theta(n)$
- ▶ `void pq.insert(int elem, VertexState &key)`  $\Theta(1)$
- ▶ `float pq.getMin()`  $\Theta(n)$
- ▶ `void pq.delMin()`  $\Theta(n)$
- ▶ `void pq.decrKey(int elem, VertexState &newkey)` setzt den Schlüssel von `elem` auf `newkey`; `newkey.curWeight` muss kleiner als beim bisherigen Schlüssel von `elem` sein.  $\Theta(1)$
- ▶ `int pq.getColor(int elem)` gibt `color` von `elem` zurück. `elem` muss dazu *nicht* in der Warteschlange sein.  $\Theta(1)$

# Prioritätswarteschlange, die Vierte (III)

## Prioritätswarteschlange

- ▶ `bool pq.isEmpty()`  $\Theta(n)$
- ▶ `void pq.insert(int elem, VertexState &key)`  $\Theta(1)$
- ▶ `float pq.getMin()`  $\Theta(n)$
- ▶ `void pq.delMin()`  $\Theta(n)$
- ▶ `void pq.decrKey(int elem, VertexState &newkey)` setzt den Schlüssel von `elem` auf `newkey`; `newkey.curWeight` muss kleiner als beim bisherigen Schlüssel von `elem` sein.  $\Theta(1)$
- ▶ `int pq.getColor(int elem)` gibt `color` von `elem` zurück.  
`elem` muss dazu *nicht* in der Warteschlange sein.  $\Theta(1)$
- ▶ `float pq.getWeight(int elem)` gibt `curWeight` von `elem` zurück.  
`elem` muss dazu *nicht* in der Warteschlange sein.  $\Theta(1)$

# Der Algorithmus von Prim – Implementierung (I)

---

```
1 // Ergebnis als Vorgängerbaum in .parent:
2 //  $\forall v \in V : (x, v) \in MST(V, E)$  gdw.  $x = state[v].parent$ ,  $x \neq -1$ 
3 VertexState[n] primMST(List adjLst[n], int n, int start) {
4     VertexState state[n] = // (eigentlich im Konstruktor von pq)
5         { color: WHITE, parent: -1, curWeight: +inf };
6     PriorityQueue pq = VS_PriorityQueue<&VS.curWeight>(&state);
7
8     pq.insert(start, {parent: -1, curWeight: 0});
9     while (!pq.isEmpty()) { // solange es Randknoten gibt
10         int v = pq.getMin(); // günstigste Kante, bzw. Randknoten
11         pq.delMin(); // setzt auch Farbe auf BLACK
12         updateFringe(pq, adjList, v); // update den Rand
13     }
14     return state;
15 }
```

---

# Der Algorithmus von Prim – Implementierung (II)

---

```
1 void updateFringe(PriorityQueue &pq, List adjLst[], int v) {
2     foreach (edge in adjLst[v]) {
3         // berechnet MST.
4         float newWeight = edge.weight;
5
6         if (pq.getColor(edge.w) == WHITE) { // -> GRAY
7             pq.insert(edge.w, {parent: v, curWeight: newWeight});
8         } else if (pq.getColor(edge.w) == GRAY) {
9             if (newWeight < pq.getWeight(edge.w)) {
10                 // Randknoten-update: Kante von v aus ist besser
11                 pq.decrKey(edge.w, {parent: v, curWeight: newWeight});
12             }
13         }
14     }
15 }
```

---

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Die Schleife in primMST wird  $n$  mal ausgeführt.

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Die Schleife in `primMST` wird  $n$  mal ausgeführt.
  - ⇒ `isEmpty`, `getMin`, `delMin` und `updateFringe` wird  $n$  mal ausgeführt.
  - ▶ Beachte, dass `getMin` eine Komplexität von  $\Theta(n)$  hat.

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Die Schleife in `primMST` wird  $n$  mal ausgeführt.
  - ⇒ `isEmpty`, `getMin`, `delMin` und `updateFringe` wird  $n$  mal ausgeführt.
    - ▶ Beachte, dass `getMin` eine Komplexität von  $\Theta(n)$  hat.
- ▶ Die Schleife in `updateFringe` wird insgesamt etwa  $2m$  mal durchlaufen.

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Die Schleife in `primMST` wird  $n$  mal ausgeführt.
  - ⇒ `isEmpty`, `getMin`, `delMin` und `updateFringe` wird  $n$  mal ausgeführt.
    - ▶ Beachte, dass `getMin` eine Komplexität von  $\Theta(n)$  hat.
- ▶ Die Schleife in `updateFringe` wird insgesamt etwa  $2m$  mal durchlaufen.
  - ⇒ `insert`, `getColor`, `getWeight` und `decrKey` werden  $m$  mal ausgeführt und sind  $\Theta(1)$ .

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Die Schleife in `primMST` wird  $n$  mal ausgeführt.
  - ⇒ `isEmpty`, `getMin`, `delMin` und `updateFringe` wird  $n$  mal ausgeführt.
    - ▶ Beachte, dass `getMin` eine Komplexität von  $\Theta(n)$  hat.
- ▶ Die Schleife in `updateFringe` wird insgesamt etwa  $2m$  mal durchlaufen.
  - ⇒ `insert`, `getColor`, `getWeight` und `decrKey` werden  $m$  mal ausgeführt und sind  $\Theta(1)$ .
- ▶ Der zusätzliche Speicherbedarf ist  $\Theta(n)$ .

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Die Schleife in `primMST` wird  $n$  mal ausgeführt.
  - ⇒ `isEmpty`, `getMin`, `delMin` und `updateFringe` wird  $n$  mal ausgeführt.
    - ▶ Beachte, dass `getMin` eine Komplexität von  $\Theta(n)$  hat.
- ▶ Die Schleife in `updateFringe` wird insgesamt etwa  $2m$  mal durchlaufen.
  - ⇒ `insert`, `getColor`, `getWeight` und `decrKey` werden  $m$  mal ausgeführt und sind  $\Theta(1)$ .
- ▶ Der zusätzliche Speicherbedarf ist  $\Theta(n)$ .
- ▶ Die untere Schranke der Zeitkomplexität ist  $\Omega(m)$ , da jede Kante des Graphen untersucht werden muss, um einen MST zu konstruieren.

# Komplexitätsanalyse

$$T(n, m) \in O(n \cdot T(\text{insert}) + n \cdot T(\text{getMin}) + n \cdot T(\text{delMin}) + m \cdot T(\text{decrKey}))$$

- ▶ Die Schleife in `primMST` wird  $n$  mal ausgeführt.
  - ⇒ `isEmpty`, `getMin`, `delMin` und `updateFringe` wird  $n$  mal ausgeführt.
    - ▶ Beachte, dass `getMin` eine Komplexität von  $\Theta(n)$  hat.
- ▶ Die Schleife in `updateFringe` wird insgesamt etwa  $2m$  mal durchlaufen.
  - ⇒ `insert`, `getColor`, `getWeight` und `decrKey` werden  $m$  mal ausgeführt und sind  $\Theta(1)$ .
- ▶ Der zusätzliche Speicherbedarf ist  $\Theta(n)$ .
- ▶ Die untere Schranke der Zeitkomplexität ist  $\Omega(m)$ , da jede Kante des Graphen untersucht werden muss, um einen MST zu konstruieren.
- ⇒ Insgesamt: Worst-Case-Komplexität  $O(n^2 + m) = O(n^2)$ .