Hochschule Zittau/Görlitz
UNIVERSITY OF APPLIED SCIENCES

Fakultät Elektrotechnik und Informatik

Greedy-Algorithmen

El Jabri Großmann

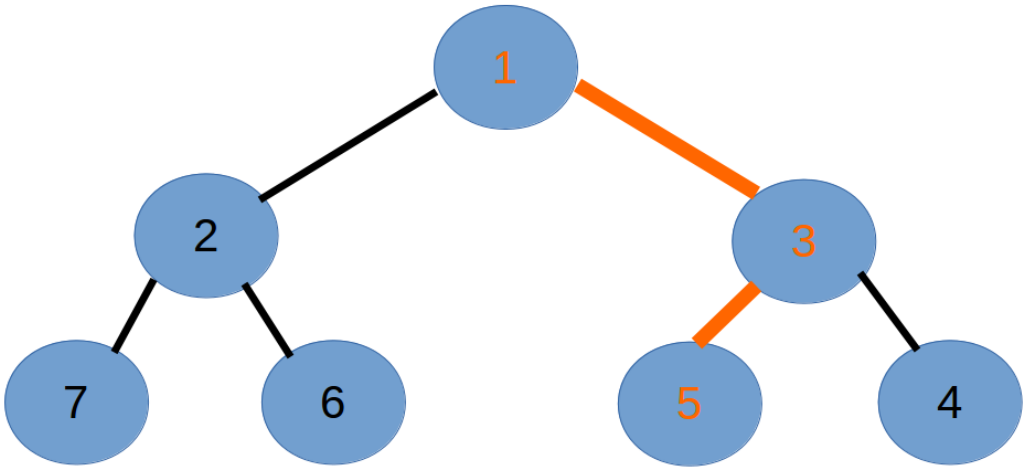
Hochschule Zittau/Görlitz
Fakultät Elektrotechnik und Informatik
eljabri.m.a.informatik@gmail.com
s3fagros@stud.hszg.de

11. Dezember 2014

Inhaltsverzeichnis

Inhaltsverzeichnis

Greedy Algorithmen



Vergleichsweise effizient bei Lösung von Optimierungsproblemen

Greedy Algorithmen

Definition

- „Gierige/Gefräßige“ Algorithmen.

Greedy Algorithmen

Definition

- „Gierige/Gefräßige“ Algorithmen.
- Lösung auf Grundlage lokaler Optima.

Greedy Algorithmen

Definition

- „Gierige/Gefräßige“ Algorithmen.
- Lösung auf Grundlage lokaler Optima.
  - Vorteil: sehr schnell, gute (Näherungs-)Lösung.

Greedy Algorithmen

Definition

- „Gierige/Gefräßige“ Algorithmen.
- Lösung auf Grundlage lokaler Optima.
  - Vorteil: sehr schnell, gute (Näherungs-)Lösung.
  - Nachteil :

Greedy Algorithmen

Definition

- „Gierige/Gefräßige“ Algorithmen.
- Lösung auf Grundlage lokaler Optima.
  - Vorteil: sehr schnell, gute (Näherungs-)Lösung.
  - Nachteil :
    1. keine Berücksichtigung früherer Lösungen; keine Revision einmal getroffener Entscheidungen.

Greedy Algorithmen

Definition

- „Gierige/Gefräßige“ Algorithmen.
- Lösung auf Grundlage lokaler Optima.
  - Vorteil: sehr schnell, gute (Näherungs-)Lösung.
  - Nachteil :
    1. keine Berücksichtigung früherer Lösungen; keine Revision einmal getroffener Entscheidungen.
    2. Endlösung möglicherweise nicht optimale Lösung.

Greedy Algorithmen

Definition

- „Gierige/Gefräßige“ Algorithmen.
- Lösung auf Grundlage lokaler Optima.
  - Vorteil: sehr schnell, gute (Näherungs-)Lösung.
  - Nachteil :
    1. keine Berücksichtigung früherer Lösungen; keine Revision einmal getroffener Entscheidungen.
    2. Endlösung möglicherweise nicht optimale Lösung.
- Voraussetzungen: Optimierungsproblem, Lösungen aus Einzelstücken bestehend, Teillösungen qualitativ unterscheidbar.

Implementierung

- Vorsortierte liste zur Kanditatenauswahl – priority queue.

Implementierung

- Vorsortierte liste zur Kanditatenauswahl – priority queue.

```
(define greedy
        (lambda (vorgabewert)
                (let
        ((kandidatenliste ´(…))
         (loesung? (lambda (ls) … ))
         (okay? (lambda (ls) … ))
         (naechster (lambda (ls) … ))
         (modify+ (lambda (ls) … ))
         (modify- (lambda (ls) … ))
         (ziel (lambda (ls) … )))
        (letrec
                ((helfer
                 (lambda (kandidaten resultat)
                  (cond
                   ((loesung? resultat)(ziel resultat)
                   ((and (null? kandidaten)(not (loesung? resultat)
                    (error ´greed „es gibt keine lösung“))
                  (else
                    (let* ((neuer-kandidat (naechster kandidaten))
                           (neues-resultat (cons neuer-kandidat resultat))
                       (if (okay? neues-resultat)
                           (helfer
                              (modify+ neuer-kandidat kandidaten) neues-resultat)
                           (helfer
                              (modify- neuer-kandidat kandidaten)
                               resultat)))))))))
          (helfer kandidatenliste ´())))))
```

Inhaltsverzeichnis

Dijkstra-Algorithmus

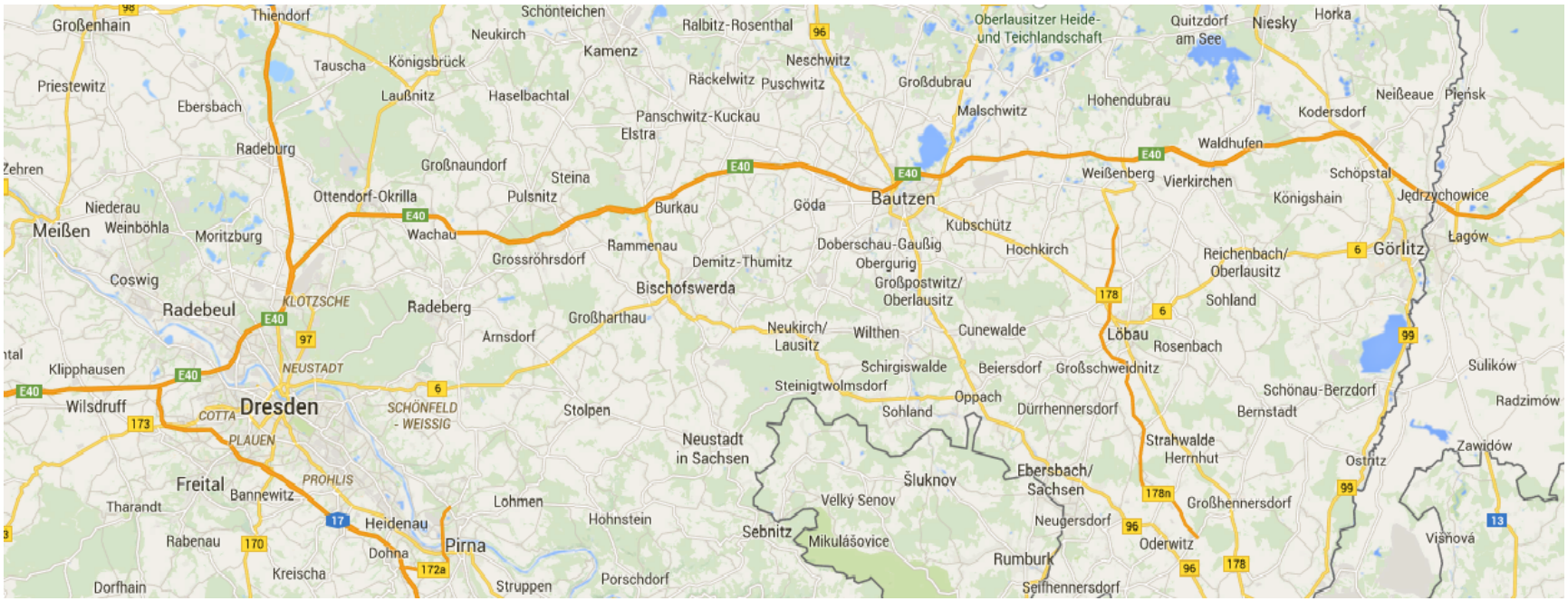
- DIJKSTRA Algorithmus: berechnet kürzesten Weg in Graphen von einem Startknoten aus für nichtnegative Kantengewichte.

Dijkstra-Algorithmus

- DIJKSTRA Algorithmus: berechnet kürzesten Weg in Graphen von einem Startknoten aus für nichtnegative Kantengewichte.
- kürzeste Strecke zw. A und B, z.B. kürzeste Route Görlitz-Dresden.

Dijkstra-Algorithmus

- DIJKSTRA Algorithmus: berechnet kürzesten Weg in Graphen von einem Startknoten aus für nichtnegative Kantengewichte.
- kürzeste Strecke zw. A und B, z.B. kürzeste Route Görlitz-Dresden.



Floyd-Warshall-Algorithmus und die transitive Hülle

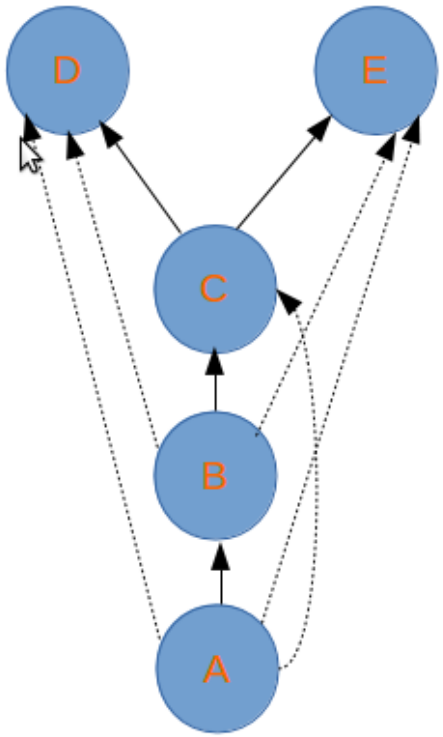
- FLOYD und WARSHALL: Floyd-algorithmus findet kürzeste Pfade zwischen allen Knotenpaaren; Warshall-algorithmus findet transitive Hülle.

Floyd-Warshall-Algorithmus und die transitive Hülle

- FLOYD und WARSHALL: Floyd-algorithmus findet kürzeste Pfade zwischen allen Knotenpaaren; Warshall-algorithmus findet transitive Hülle.
- Transitive Hülle: enthält zusätzlich zu direkten Relationen (erreichbare Punkte) alle indirekten Relationen

Floyd-Warshall-Algorithmus und die transitive Hülle

- FLOYD und WARSHALL: Floyd-algorithmus findet kürzeste Pfade zwischen allen Knotenpaaren; Warshall-algorithmus findet transitive Hülle.
- Transitive Hülle: enthält zusätzlich zu direkten Relationen (erreichbare Punkte) alle indirekten Relationen



Inhaltsverzeichnis

Gerichtete Graphen

Definition

- Ein gerichteter Graph umfaßt eine endliche Menge von Knoten V und eine endliche Menge von gerichteten Kanten E.

Gerichtete Graphen

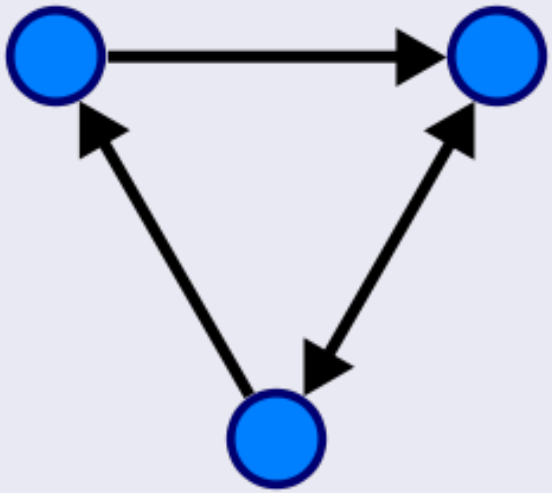
Definition

- Ein gerichteter Graph umfaßt eine endliche Menge von Knoten V und eine endliche Menge von gerichteten Kanten E.
- Dabei verbindet jede Kante $e \in E$ genau einen Anfangsknoten $v \in V$ mit genau einem Endknoten $u \in V$. Man sagt auch, Kante e führt von Knoten v nach Knoten u.

Gerichtete Graphen

Definition

- Ein gerichteter Graph umfaßt eine endliche Menge von Knoten V und eine endliche Menge von gerichteten Kanten E.
- Dabei verbindet jede Kante $e \in E$ genau einen Anfangsknoten $v \in V$ mit genau einem Endknoten $u \in V$. Man sagt auch, Kante e führt von Knoten v nach Knoten u.



Ungerichtete Graphen

Definition

- Ein ungerichteter Graph umfaßt eine endliche Menge von Knoten V und eine endliche Menge von ungerichteten Kanten E.

Ungerichtete Graphen

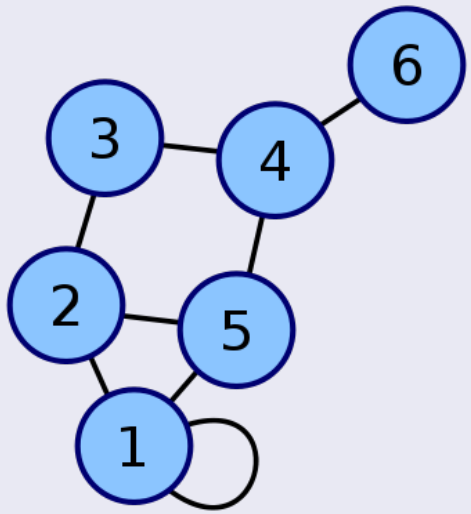
Definition

- Ein ungerichteter Graph umfaßt eine endliche Menge von Knoten V und eine endliche Menge von ungerichteten Kanten E.
- Jede ungerichtete Kante $e \in E$ verbindet entweder zwei verschiedene Knoten $(u, v) \in V$ miteinander oder im Falle einer ungerichteten Schlinge einen Knoten $v \in V$ mit sich selbst.

Ungerichtete Graphen

Definition

- Ein ungerichteter Graph umfaßt eine endliche Menge von Knoten V und eine endliche Menge von ungerichteten Kanten E.
- Jede ungerichtete Kante $e \in E$ verbindet entweder zwei verschiedene Knoten $(u, v) \in V$ miteinander oder im Falle einer ungerichteten Schlinge einen Knoten $v \in V$ mit sich selbst.



Die Adjazenzmatrix

Die Adjazenzmatrix Repräsentation

- Die Adjazenzmatrix von einem Graph $G = (V, E)$, wo $V = 1, 2, ..., n$, ist die Matrix $A[1..n, 1..n]$.
  Gegeben durch :

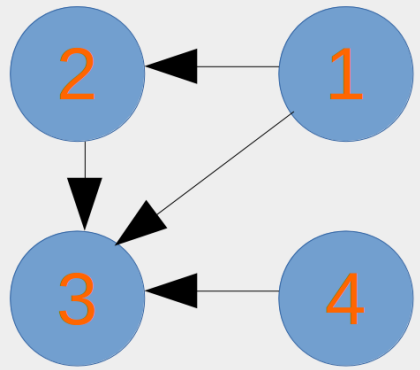
$$A[i,j] = \begin{cases} 1 & \text{wenn } (i,j) \in E \\ 0 & \text{wenn } (i,j) \notin E \end{cases}$$

Die Adjazenzmatrix

Die Adjazenzmatrix Repräsentation

- Die Adjazenzmatrix von einem Graph $G = (V, E)$, wo $V = 1, 2, ..., n$, ist die Matrix $A[1..n, 1..n]$.
  Gegeben durch :

$$A[i,j] = \begin{cases} 1 & \text{wenn } (i,j) \in E \\ 0 & \text{wenn } (i,j) \notin E \end{cases}$$

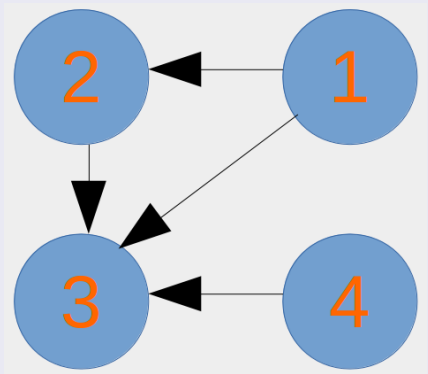


Die Adjazenzmatrix

Die Adjazenzmatrix Repräsentation

- Die Adjazenzmatrix von einem Graph $G = (V, E)$, wo $V = 1, 2, ..., n$, ist die Matrix $A[1..n, 1..n]$.
  Gegeben durch :

$$A[i,j] = \begin{cases} 1 & \text{wenn } (i,j) \in E \\ 0 & \text{wenn } (i,j) \notin E \end{cases}$$



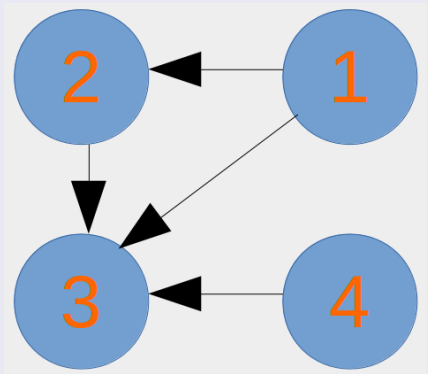
A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

Die Adjazenzmatrix

Die Adjazenzmatrix Repräsentation

- Die Adjazenzmatrix von einem Graph $G = (V, E)$, wo $V = 1, 2, ..., n$, ist die Matrix $A[1..n, 1..n]$.
  Gegeben durch :

$$A[i,j] = \begin{cases} 1 & \text{wenn } (i,j) \in E \\ 0 & \text{wenn } (i,j) \notin E \end{cases}$$



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$$Storage = \Theta(V^2)$$

Die Adjazenzliste

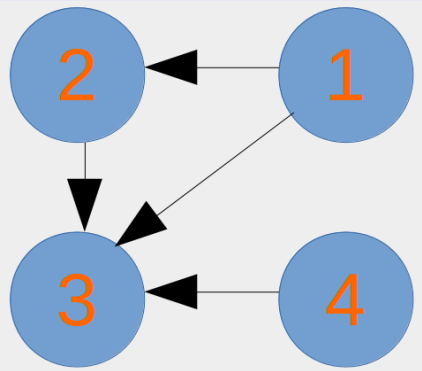
Die Adjazenzliste Repräsentation

- Die Adjazenzliste von einem Knoten $v \in V$ ist die Liste Adj[v] von Knoten, die zu v adjazent sind.

Die Adjazenzliste

Die Adjazenzliste Repräsentation

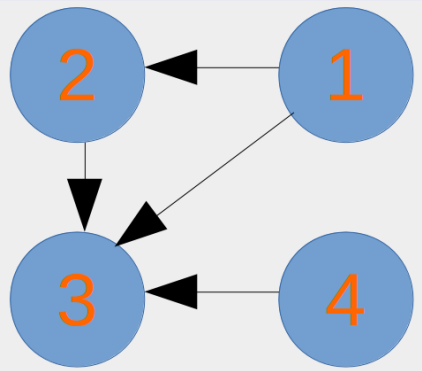
- Die Adjazenzliste von einem Knoten $v \in V$ ist die Liste Adj[v] von Knoten, die zu v adjazent sind.
- Die Adjazenzliste von diesem Graph ist :



Die Adjazenzliste

Die Adjazenzliste Repräsentation

- Die Adjazenzliste von einem Knoten $v \in V$ ist die Liste Adj[v] von Knoten, die zu v adjazent sind.
- Die Adjazenzliste von diesem Graph ist :

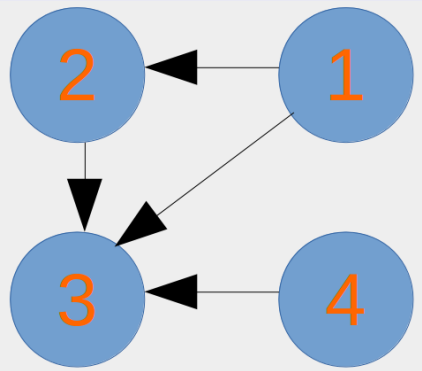


$$Adj[1] = (2, 3)$$
$$Adj[2] = (3)$$
$$Adj[3] = ()$$
$$Adj[4] = (3)$$

Die Adjazenzliste

Die Adjazenzliste Repräsentation

- Die Adjazenzliste von einem Knoten $v \in V$ ist die Liste Adj[v] von Knoten, die zu v adjazent sind.
- Die Adjazenzliste von diesem Graph ist :

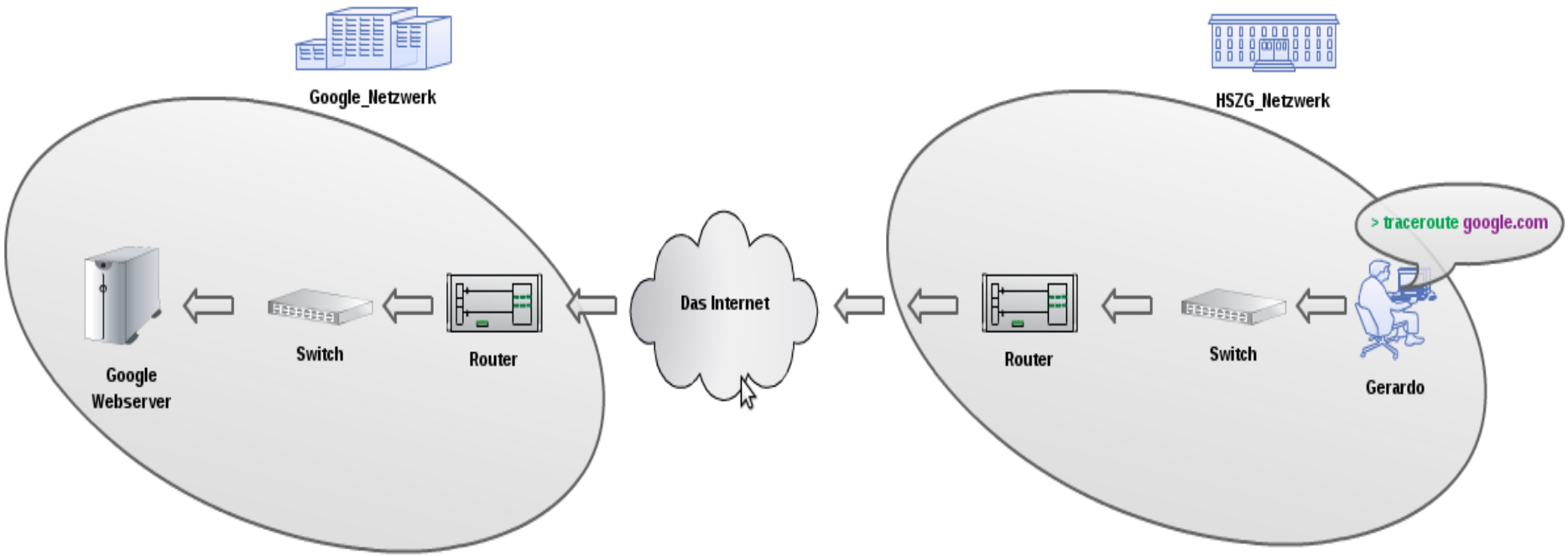


$$Adj[1] = (2, 3)$$
$$Adj[2] = (3)$$
$$Adj[3] = ()$$
$$Adj[4] = (3)$$

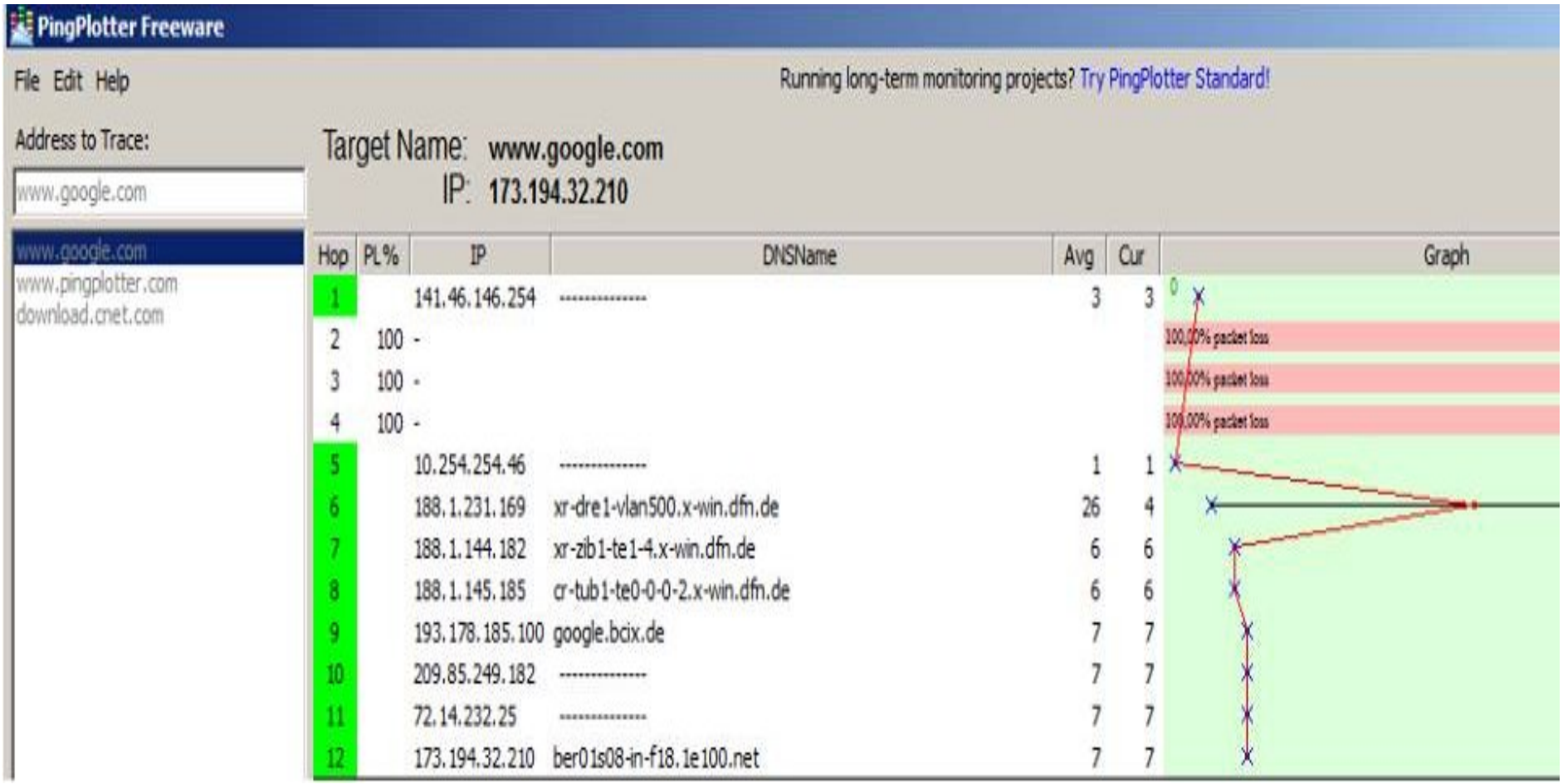
$$Storage = \Theta(V + E)$$

Inhaltsverzeichnis

Minimaler Spannbaum



Minimaler Spannbaum



Minimaler Spannbaum

Verwendungen von Minimalen Spannbäumen

Verwendung	Knoten	Kanten
Schaltung	Bauelement	Draht
Fluggesellschaft	Flughafen	Flugroute
Stromverteiler	Kraftwerk	Leitung
Fluggesellschaft	Flughafen	Flugroute

Minimaler Spannbaum

Definition

1. Ein gewichteter ungerichteter Graph (G, w) ist ein ungerichteter Graph $G = (V, E)$ zusammen mit einer Gewichtsfunktion

$$w : E \longrightarrow \mathbb{R}$$

Minimaler Spannbaum

Definition

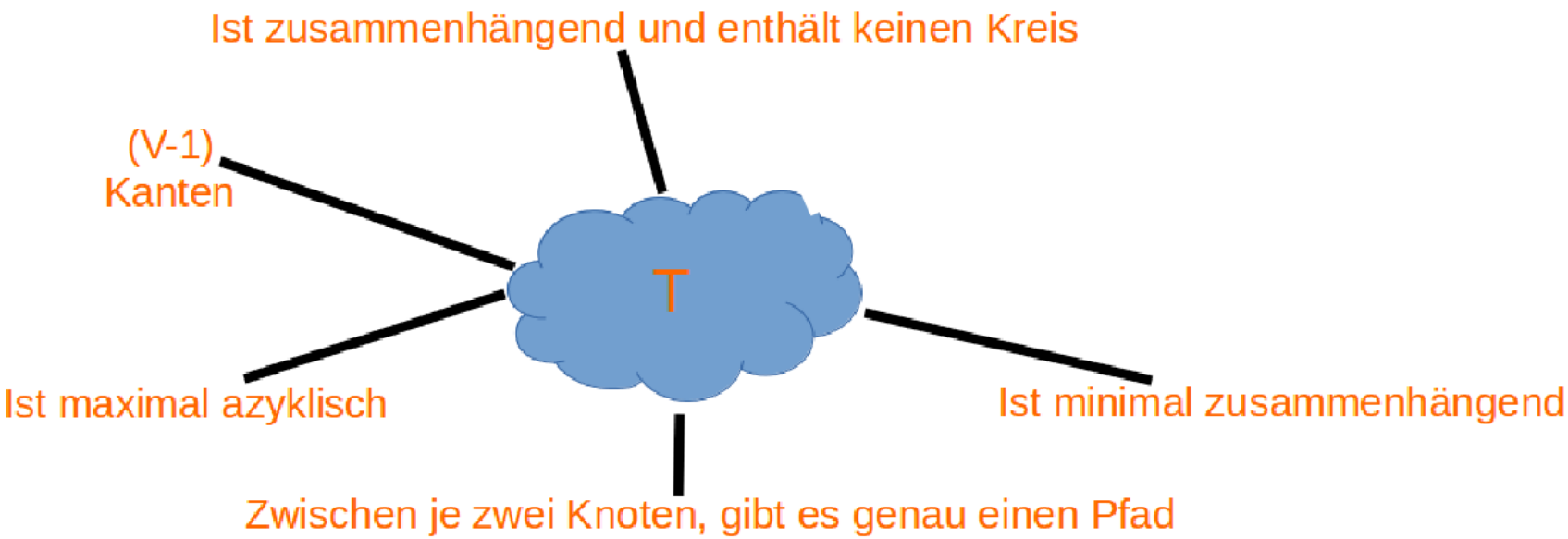
1. Ein gewichteter ungerichteter Graph (G, w) ist ein ungerichteter Graph $G = (V, E)$ zusammen mit einer Gewichtsfunktion
$$w : E \Longrightarrow \mathbb{R}$$
2. Ein Teilgraph H eines ungerichteten Graphen G heisst Spannbaum von G, wenn H ein Baum auf den Knoten von G ist.

Minimaler Spannbaum

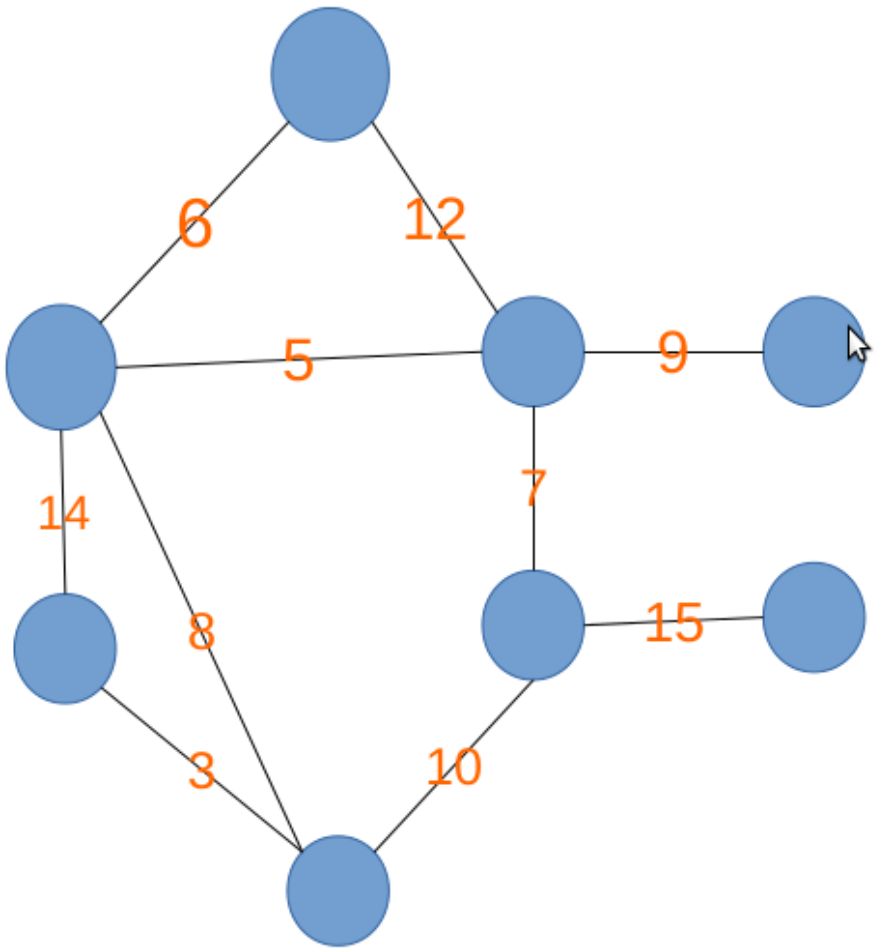
Definition

1. Ein gewichteter ungerichteter Graph (G, w) ist ein ungerichteter Graph $G = (V, E)$ zusammen mit einer Gewichtsfunktion
$$w : E \Longrightarrow \mathbb{R}$$
2. Ein Teilgraph H eines ungerichteten Graphen G heisst Spannbaum von G, wenn H ein Baum auf den Knoten von G ist.
3. Ein Spannbaum T eines gewichteten ungerichteten Graphen G heisst minimaler Spannbaum von G, wenn T minimales Gewicht unter allen Spannbäumen von G besitzt.
$$w(t) = \sum_{(u,v) \in T} w(u, v)$$

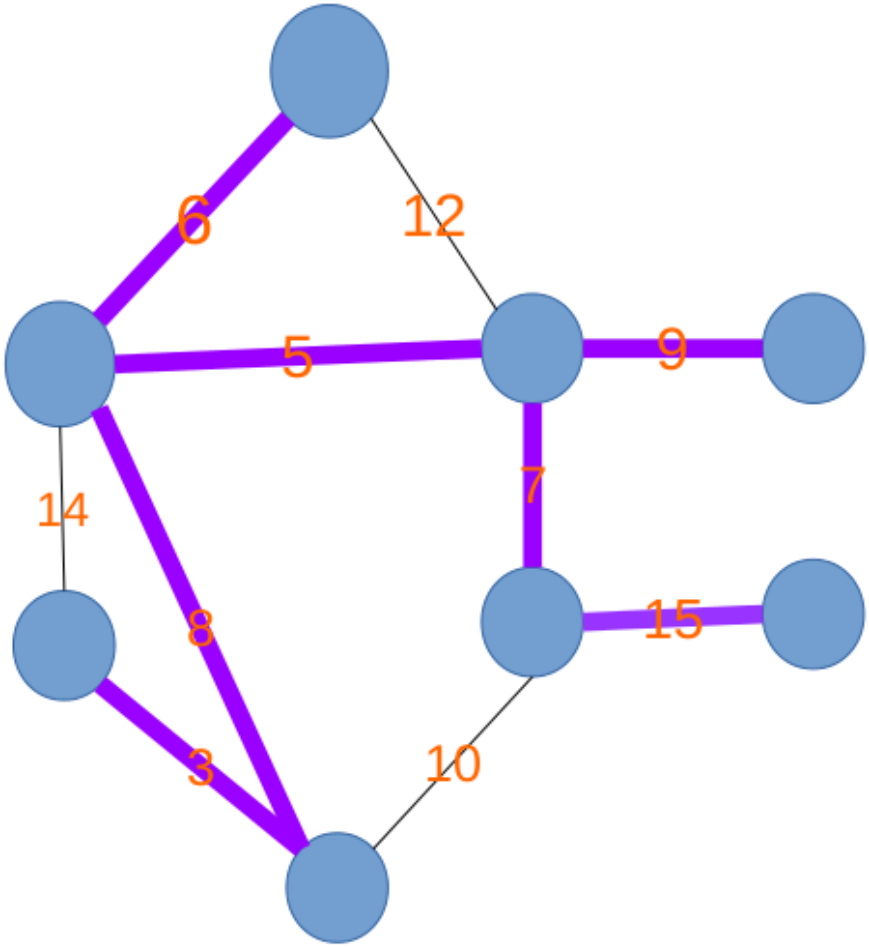
Äquivalente Charakterisierungen von Bäumen



Beispiel



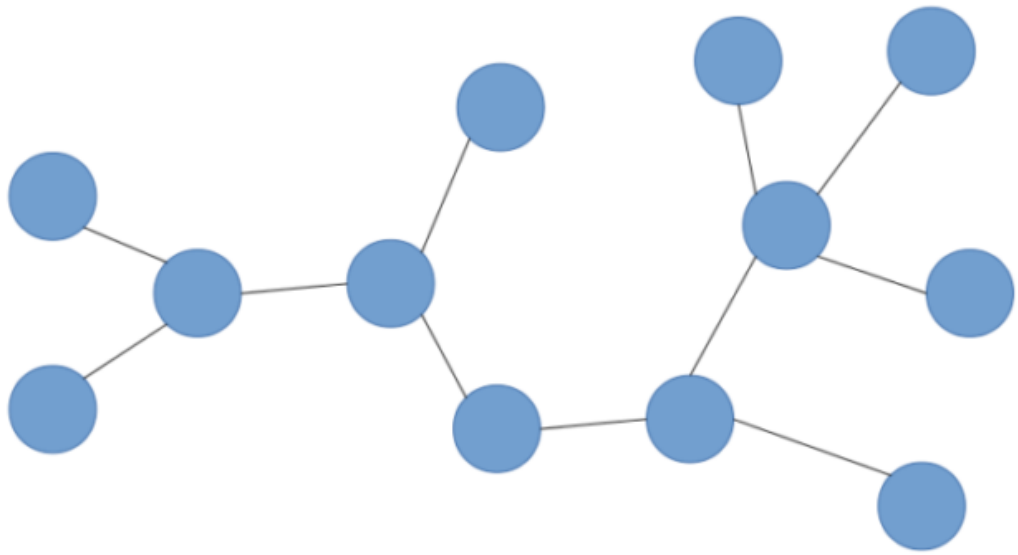
Beispiel



Inhaltsverzeichnis

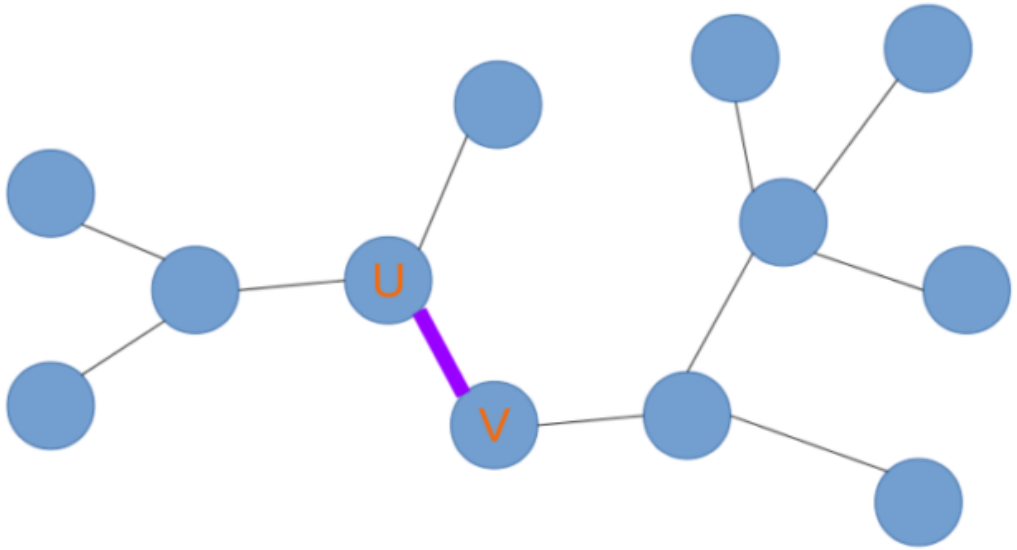
Optimale Substruktur - Dynamic Programming

- Minimaler Spannbaum T : (Die anderen Kanten von G sind nicht gezeigt)



Optimale Substruktur - Dynamic Programming

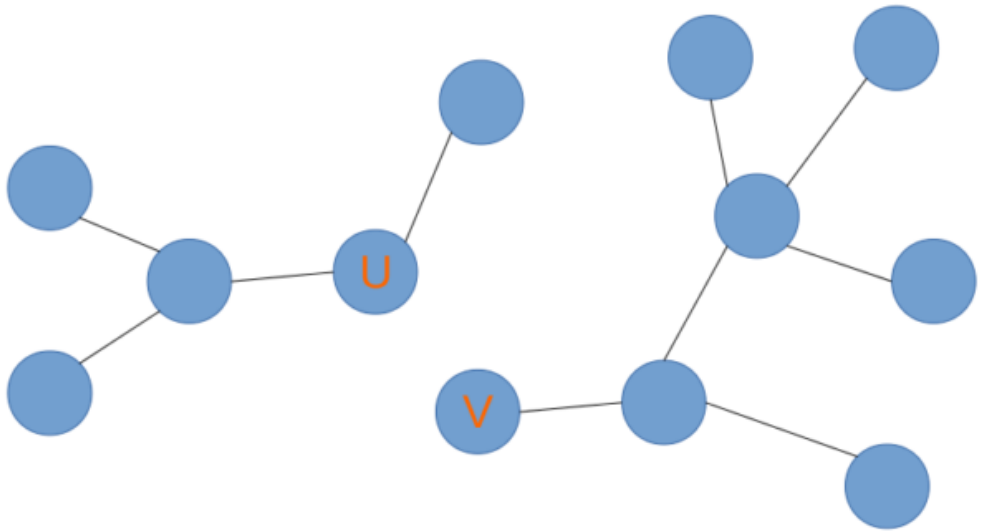
- Minimaler Spannbaum T : (Die anderen Kanten von G sind nicht gezeigt)



- Wir löschen irgendeine Kante $(u, v) \in T$.

Optimale Substruktur - Dynamic Programming

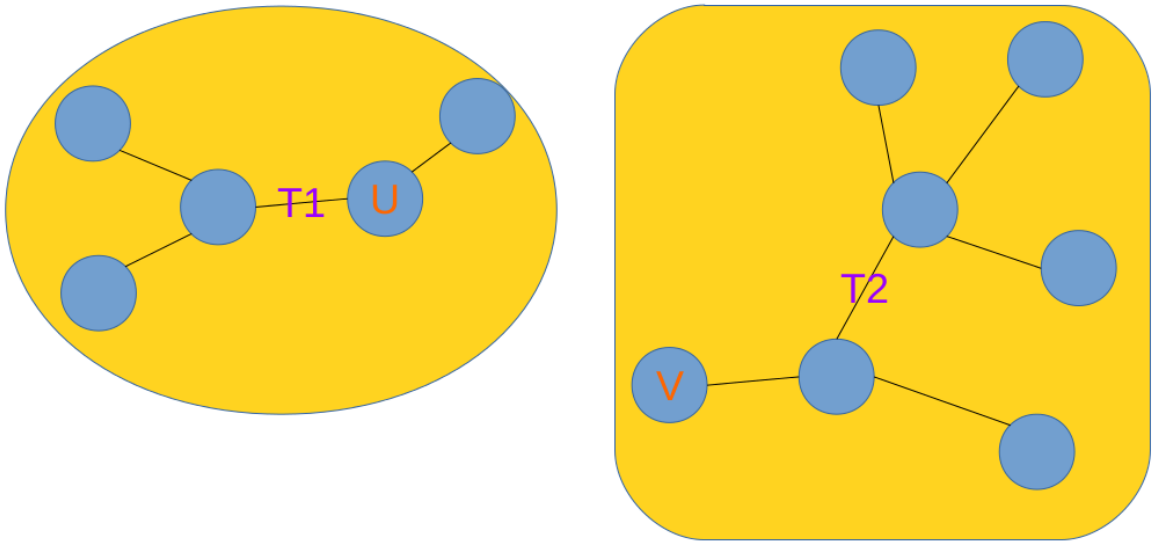
- Minimaler Spannbaum T : (Die anderen Kanten von G sind nicht gezeigt)



- Wir löschen irgendeine Kante $(u, v) \in T$.

Optimale Substruktur - Dynamic Programming

- Minimaler Spannbaum T : (Die anderen Kanten von G sind nicht gezeigt)



- Wir löschen irgendeine Kante $(u, v) \in T$.
- T ist unterteilt in zwei Teilbäume T_1 und T_2

Optimale Substruktur - Dynamic Programming

Theorem

Der Teilbaum T_1 ist ein minimaler Spannbaum von $G_1 = (V_1, E_1)$, der den Teilgraph von G durch die Knoten von T_1 induziert.

Optimale Substruktur - Dynamic Programming

Theorem

Der Teilbaum T_1 ist ein minimaler Spannbaum von $G_1 = (V_1, E_1)$, der den Teilgraph von G durch die Knoten von T_1 induziert.

$$V_1 = \text{die Knoten von } T_1$$
$$E_1 = (x,y) \in E : (x,y) \in V_1$$

Optimale Substruktur - Dynamic Programming

Beweis (durch Wiederspruch)

$$w(T) = w(u, v) + w(T_1) + w(T_2)$$

Beweis (durch Wiederspruch)

$$w(T) = w(u, v) + w(T_1) + w(T_2)$$

Widerspruch ausnahme :

Beweis (durch Wiederspruch)

$$w(T) = w(u, v) + w(T_1) + w(T_2)$$

Widerspruch ausnahme :

Es gibt einen Teilbaum T_1', der ein niedrigeres Gewicht als T_1 hat

Beweis (durch Wiederspruch)

$$w(T) = w(u, v) + w(T_1) + w(T_2)$$

Widerspruch ausnahme :

Es gibt einen Teilbaum T_1', der ein niedrigeres Gewicht als T_1 hat

Beweis (durch Wiederspruch)

$$w(T) = w(u, v) + w(T_1) + w(T_2)$$

Widerspruch ausnahme :

Es gibt einen Teilbaum T_1', der ein niedrigeres Gewicht als T_1 hat

$$\Downarrow$$

$$w(T') = w(u, v) + w(T_1') + w(T_2)$$

Beweis (durch Wiederspruch)

$$w(T) = w(u,v) + w(T_1) + w(T_2)$$

Widerspruch ausnahme :

Es gibt einen Teilbaum T_1', der ein niedrigeres Gewicht als T_1 hat

$$\Downarrow$$

$$w(T') = w(u,v) + w(T_1') + w(T_2)$$

$$\Downarrow$$

T' ist optimaler als T: Wiederspruch !

Beweis (durch Wiederspruch)

$$w(T) = w(u,v) + w(T_1) + w(T_2)$$

Widerspruch ausnahme :

Es gibt einen Teilbaum T_1', der ein niedrigeres Gewicht als T_1 hat

$$\Downarrow$$

$$w(T') = w(u,v) + w(T_1') + w(T_2)$$

$$\Downarrow$$

T' ist optimaler als T: Wiederspruch !

$\Rightarrow$ Eigenschaft Dynamischer Programmierung : Overlapping subproblems

Inhaltsverzeichnis

Minimaler Spannbaum Algorithmen

Der Prim-Algorithmus

Vorgehen

1. Ausgangspunkt für das Verfahren ist ein beliebiger Startknoten.

Der Prim-Algorithmus

Vorgehen

1. Ausgangspunkt für das Verfahren ist ein beliebiger Startknoten.
2. Alle Kanten zu Nachbarknoten werden in eine Nachbarliste eingefügt, man wählt eine Kante minimaler Länge aus der Nachbarliste und fügt diese Kante dem bereits initialisierten Spannbaum zu (Greedy-Prinzip!) .

Der Prim-Algorithmus

Vorgehen

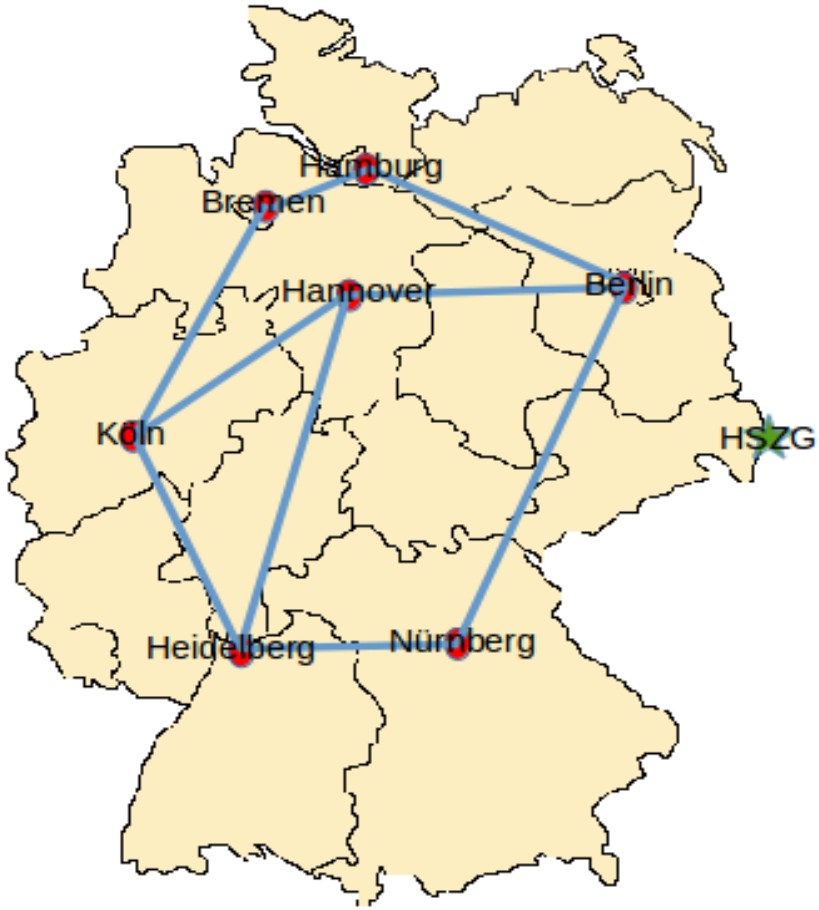
1. Ausgangspunkt für das Verfahren ist ein beliebiger Startknoten.
2. Alle Kanten zu Nachbarknoten werden in eine Nachbarliste eingefügt, man wählt eine Kante minimaler Länge aus der Nachbarliste und fügt diese Kante dem bereits initialisierten Spannbaum zu (Greedy-Prinzip!) .
3. Von dort wird wieder der minimale Weg, basierend auf der ausgewählten Kante, zum nächsten Knoten gewählt, ist dieser Knoten bereits besucht worden, wird er nicht berücksichtigt.

Der Prim-Algorithmus

Vorgehen

1. Ausgangspunkt für das Verfahren ist ein beliebiger Startknoten.
2. Alle Kanten zu Nachbarknoten werden in eine Nachbarliste eingefügt, man wählt eine Kante minimaler Länge aus der Nachbarliste und fügt diese Kante dem bereits initialisierten Spannbaum zu (Greedy-Prinzip!) .
3. Von dort wird wieder der minimale Weg, basierend auf der ausgewählten Kante, zum nächsten Knoten gewählt, ist dieser Knoten bereits besucht worden, wird er nicht berücksichtigt.
4. Dieses Verfahren führt man durch, bis alle Knoten besucht wurden.

Illustration

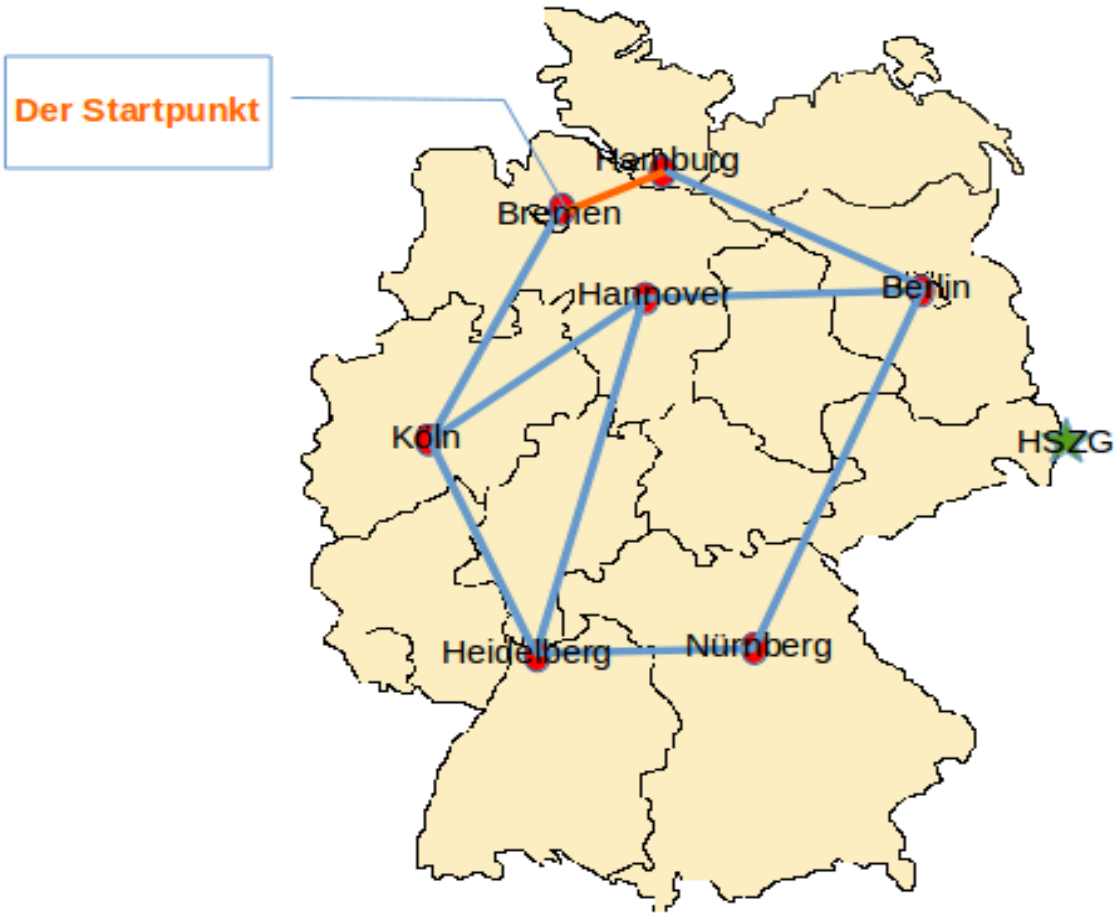


121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

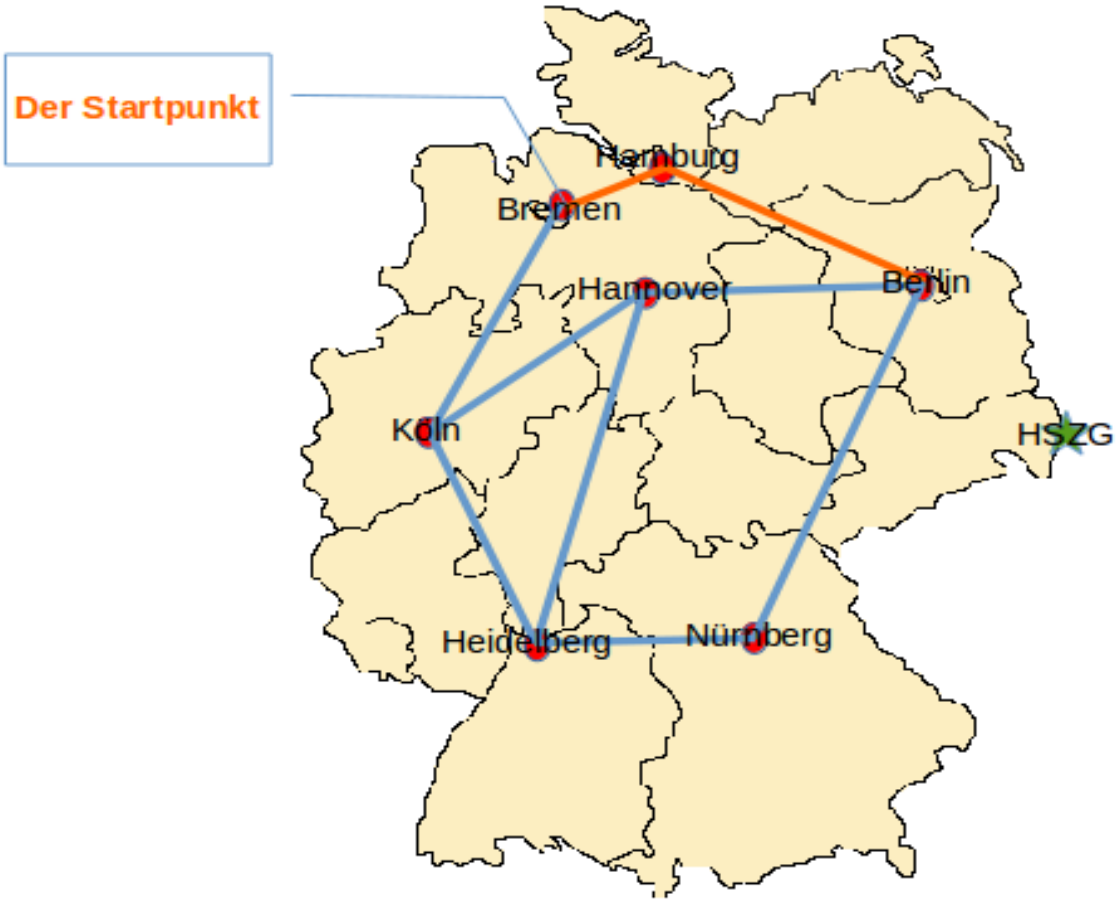


121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

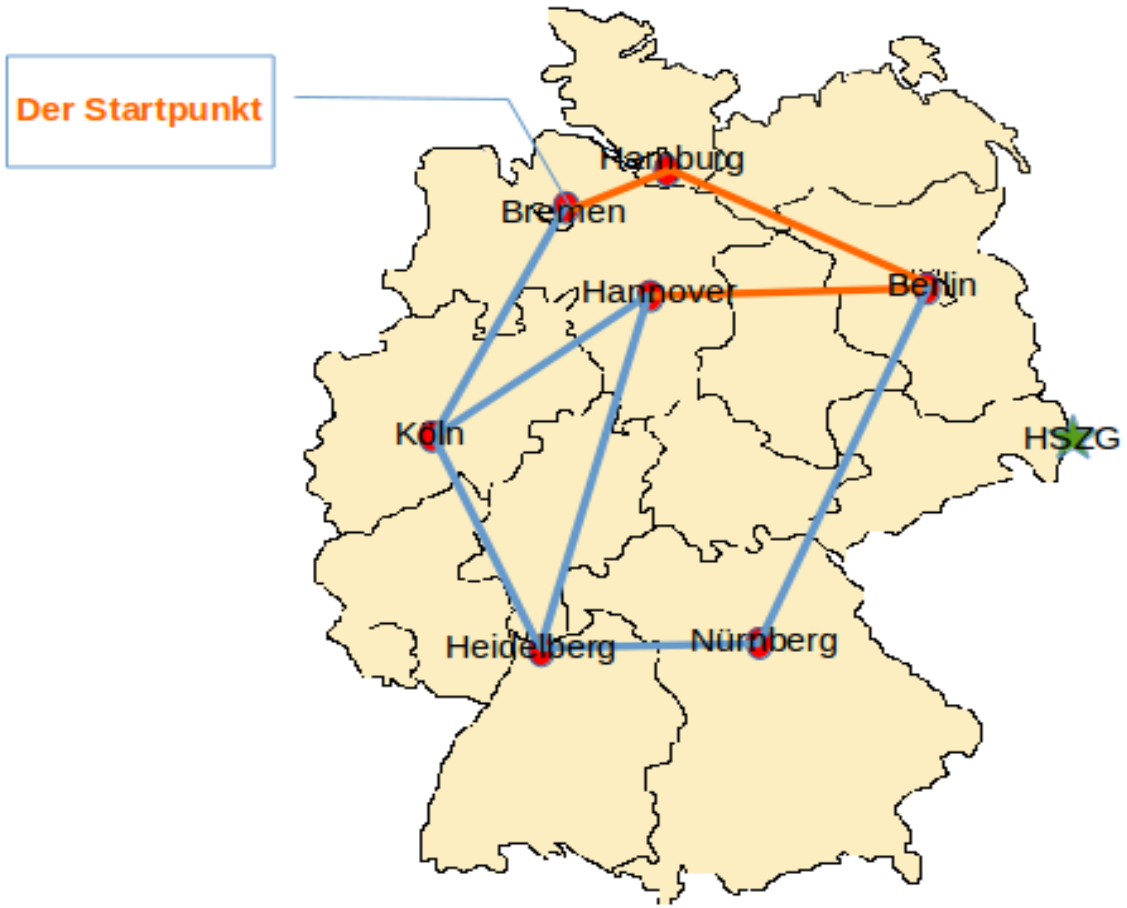
286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration



Illustration

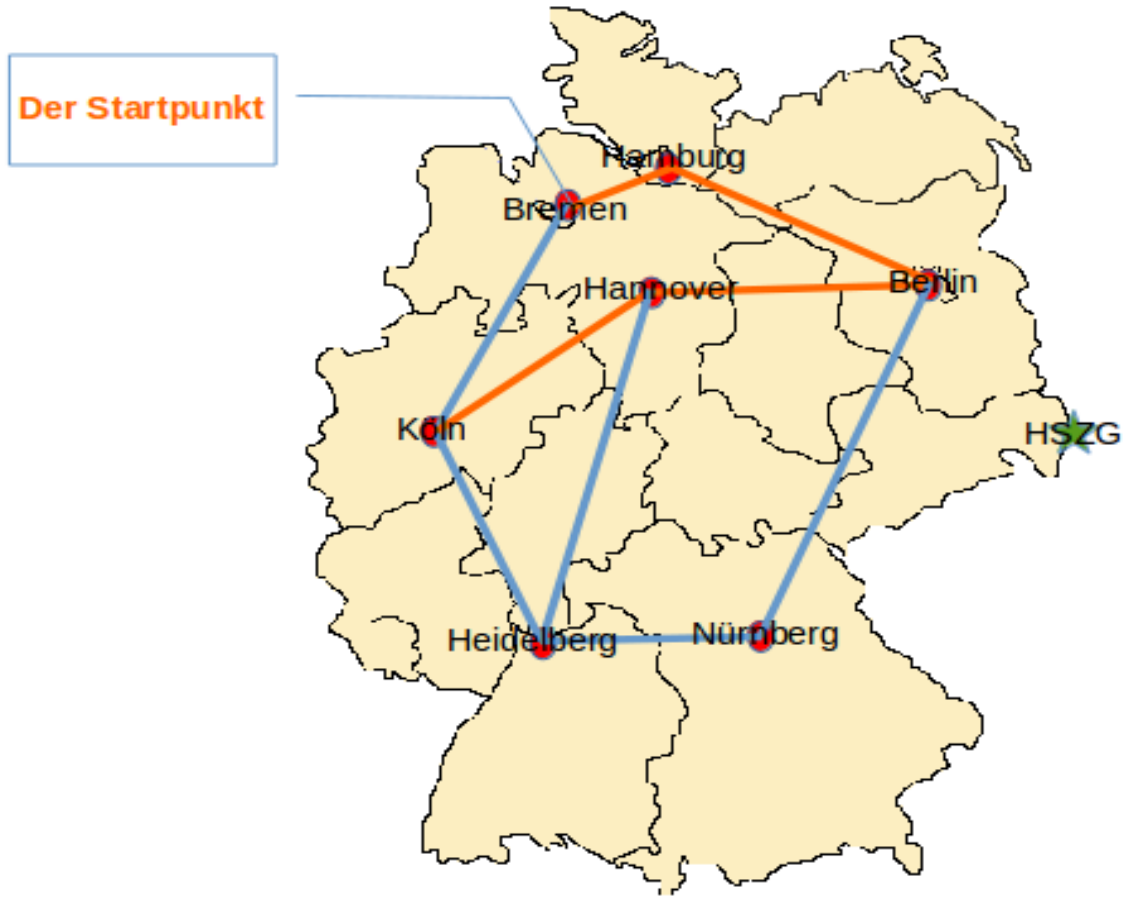


121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

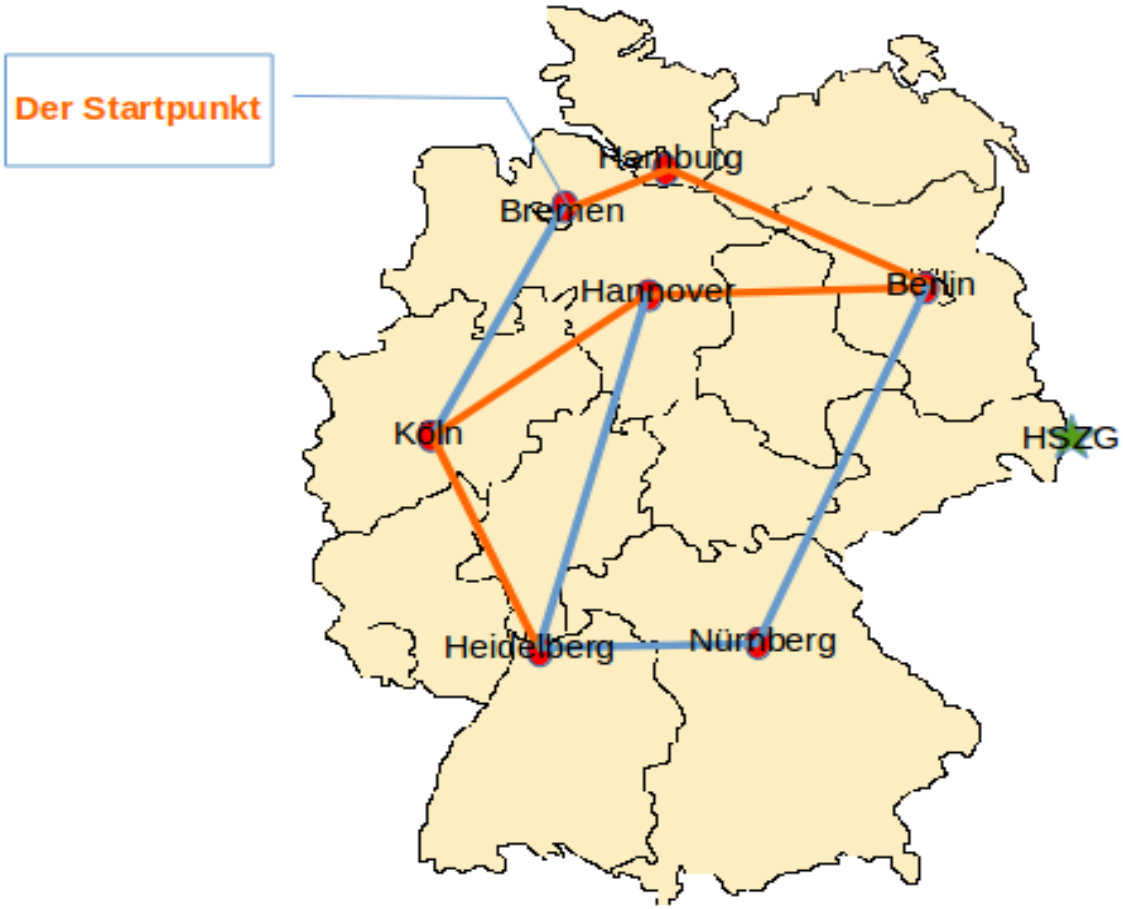
286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

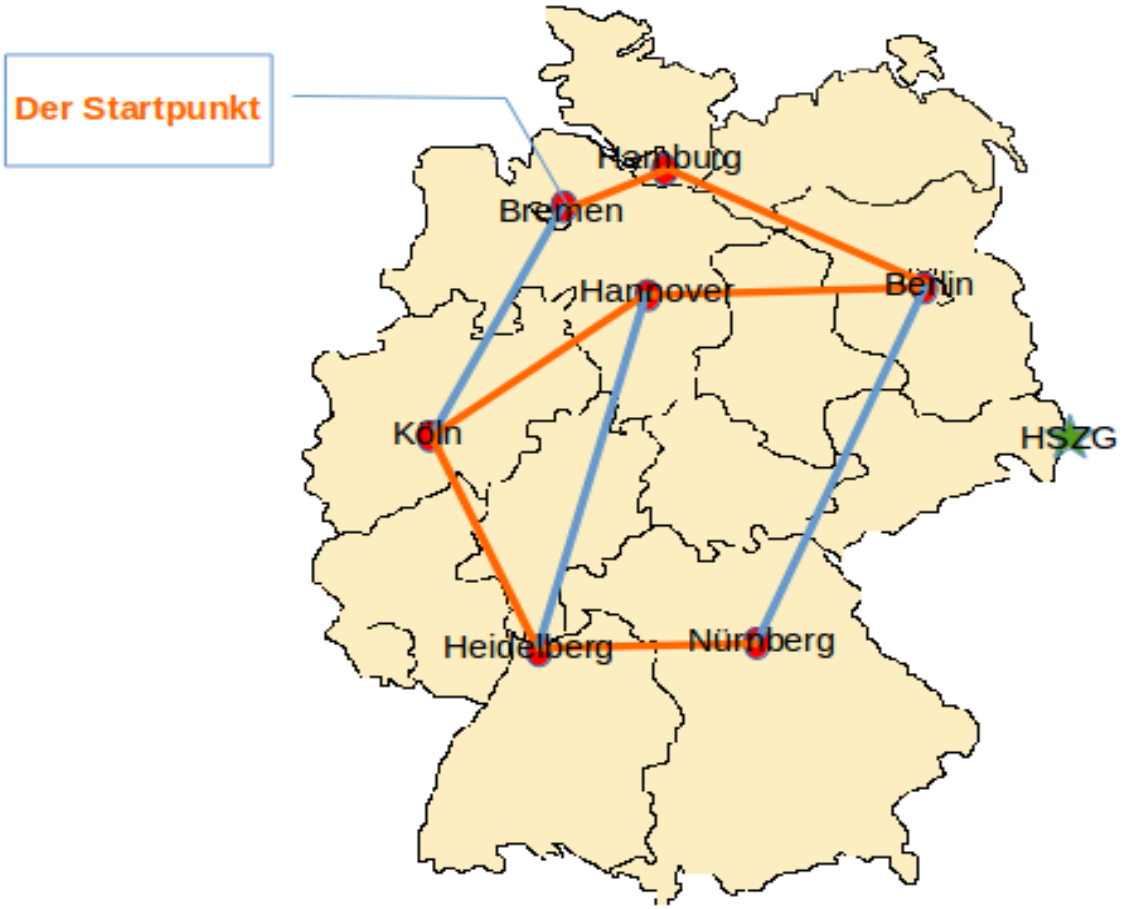
Illustration



Illustration



Illustration



121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

Minimaler Spannbaum = 1477km

Prim-Pseudocode

Algorithmus von prim(G,w,r)
Q $\leftarrow$ VG //Initialisierung
for all $u \in Q$ **do**
    $wert[u] \leftarrow \infty$
    $\pi[u] \leftarrow 0$
    $wert[r] \leftarrow 0$
while $Q \neq \varnothing$ **do**
    $u \leftarrow extractmin(Q)$
    for all $v \in Adj[u]$ **do**
        wenn $v \in Q$ und $w(u, v) < wert[v]$
        dann $\pi[v] \leftarrow u$
        $wert[v] \leftarrow w(u, v)$

Der Kruskal-Algorithmus

Vorgehen

1. Alle Kanten des Graphen werden nach ihrem Gewicht in einer Kantenliste sortiert.

Der Kruskal-Algorithmus

Vorgehen

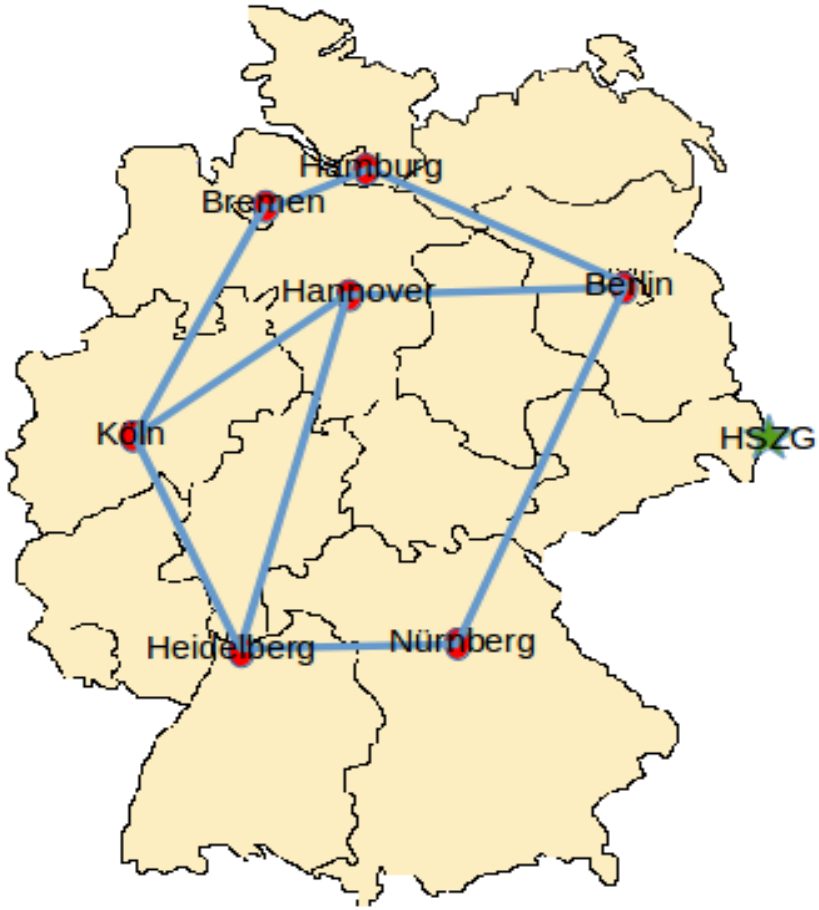
1. Alle Kanten des Graphen werden nach ihrem Gewicht in einer Kantenliste sortiert.
2. Jeder Knoten wird als Baum aufgefasst, dabei ist jeder Knoten Sohn und Vater zugleich.

Der Kruskal-Algorithmus

Vorgehen

1. Alle Kanten des Graphen werden nach ihrem Gewicht in einer Kantenliste sortiert.
2. Jeder Knoten wird als Baum aufgefasst, dabei ist jeder Knoten Sohn und Vater zugleich.
3. Die Kante mit dem geringsten Gewicht wird genommen (Greedy-Prinzip) und überprüft, ob die Knoten am Kantenende in unterschiedlichen Bäumen sind (kein Zyklus), ist dies der Fall, werden die Kanten vereinigt und die Kante aus der Kantenliste gelöscht, im anderen Fall wird die Kante nicht aufgenommen .

Illustration

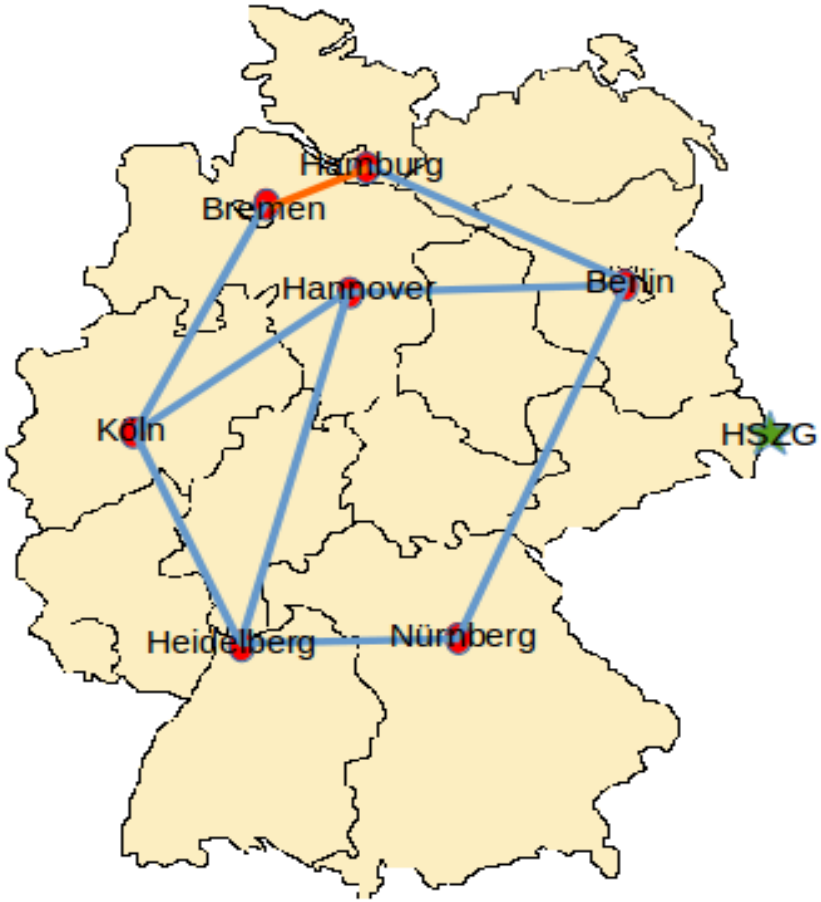


121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration



121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

121km : Hamburg - Bremen
238km : Nürnberg – Heidelberg
254km : Köln - Heidelberg

286km : Hannover - Berlin
288km : Berlin – Hamburg
290km : Köln - Hannover

313km : Köln - Bremen
431km : Nürnberg – Berlin
432km : Hannover - Heidelberg

Illustration

Minimaler Spannbaum = 1477km

Kruskal-Pseudocode

G = (V,E,w) : ungerichteter, kantengewichteter Graph.
Kruskal(G)
$E' \leftarrow \emptyset$
$L \leftarrow E$
Sortiere die Kanten in L aufsteigend nach ihrem Kantengewicht.
for all $L \neq \emptyset$ **do**
    wähle eine Kante $e \in L$ mit kleinstem Kantengewicht
    entferne die Kante e aus L
    wenn der Graph $(V, E' \cup \{e\})$ keinen Kreis enthält
    dann $E' \leftarrow E' \cup \{e\}$
M = (V,E') ist ein minimaler Spannbaum von G.

Effizienz der MST-Algorithmen

Algorithmus	Speicherbedarf	Laufzeit
Der Prim-Algorithmus	V	$E \log(V)$
Der Kruskal-Algorithmus	E	$E \log(E)$

Vielen Dank für Ihre Aufmerksamkeit

Literaturhinweise

Algorithmen und Komplexität

Christian Wagenknecht (2003)

Algorithms

Robert Sedgewick, Kevin Wayne (2011)

Introduction to Algorithms - Third Edition

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2009)

The Algorithm Design Manual

Steven S Skiena (2008)

Data Structures and Algorithms

Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft (1983)