Algorithmen und Datenstrukturen

Erreichbarkeit, Transitive Hülle, Minimaler Spannbaum

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Felix Kuhr (Übungen)

sowie viele Tutoren

Danksagung

Die nachfolgenden Präsentationen wurden mit ausdrücklicher Erlaubnis des Autors übernommen aus:

- „Effiziente Algorithmen und Datenstrukturen“ (Kapitel 7,8,9) gehalten von Christian Scheideler an der TUM
  http://www14.in.tum.de/lehre/2008WS/ea/index.html.de

Es wurden Veränderungen vorgenommen,
etwaige Fehler sind nur uns anzulasten

Transitive Hülle / Erreichbarkeit

Problem: Konstruiere für einen gerichteten Graphen G=(V,E) eine Datenstruktur, die die folgende Operation (speicher- und zeit-)effizient unterstützt:

- Reachable(v,w): liefert 1, falls es einen gerichteten Weg von v nach w in G gibt und sonst 0

Naives Verfahren für Erreichbarkeit

Algorithmus von Warshall [Wikipedia]

```
(1) Für k = 1 bis n
(2)   Für i = 1 bis n
(3)     Falls d[i,k] = 1
(4)       Für j = 1 bis n
(5)         Falls d[k,j] = 1
(6)           d[i,j] := 1
```

Im Prinzip gleiche Idee wie APSP nach Floyd, daher auch Floyd-Warshall-Algorithmus genannt

Analyse: $O(n^3)$ Das sollten wir doch besser hinkriegen?

Stephen Warshall: A Theorem on Boolean Matrices.
In: Journal of the ACM 9, 1, **1962**

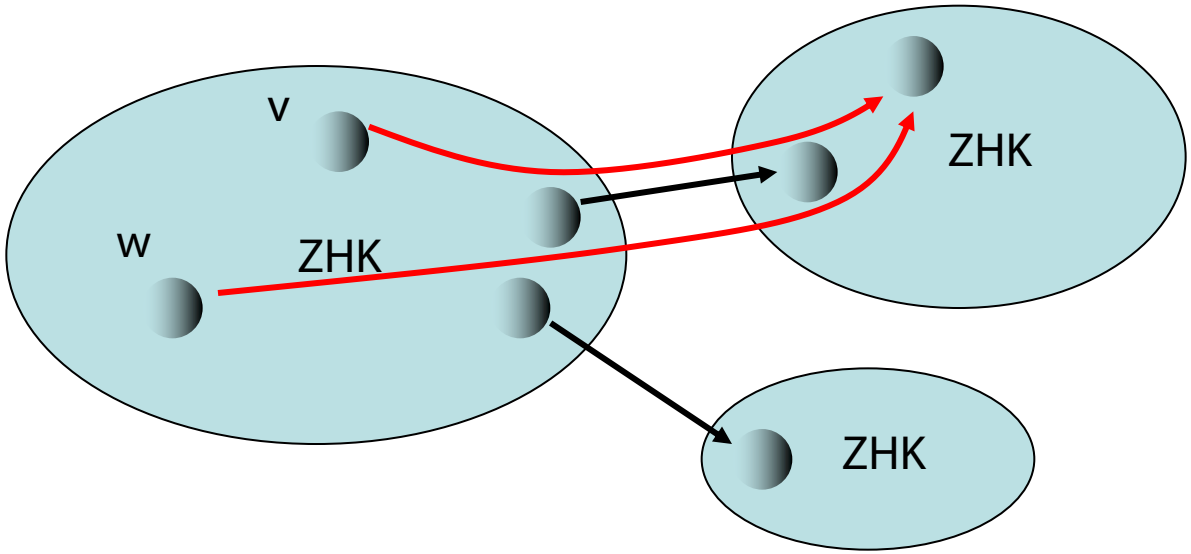
Transitive Hülle

Lösung 1: verwende APSP Algorithmus zur Erstellung einer Datenstruktur, aus der Ergebnis direkt ablesbar

- Laufzeit zur Erstellung der Datenstruktur: $O(n \cdot m + n^2 \log n)$
- Speicheraufwand: $O(n^2)$
- Laufzeit von Reachable(v,w): $O(1)$
  (Nachschauen in Tabelle, ob $\mu(v,w)<\infty$)

Transitive Hülle

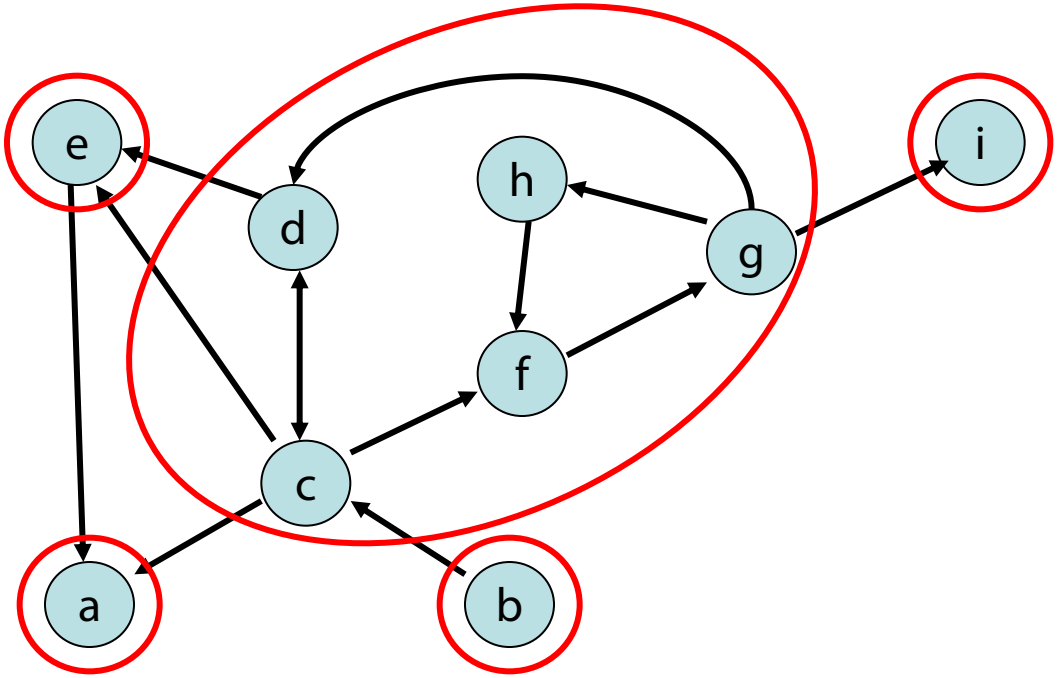
Einsicht: Alle Knoten in einer starken ZHK haben dieselbe Menge erreichbarer Knoten. Daher reicht es, sie durch Repräsentanten zu vertreten.



Transitive Hülle

Lösung 2: verwende ZHK-Algorithmus

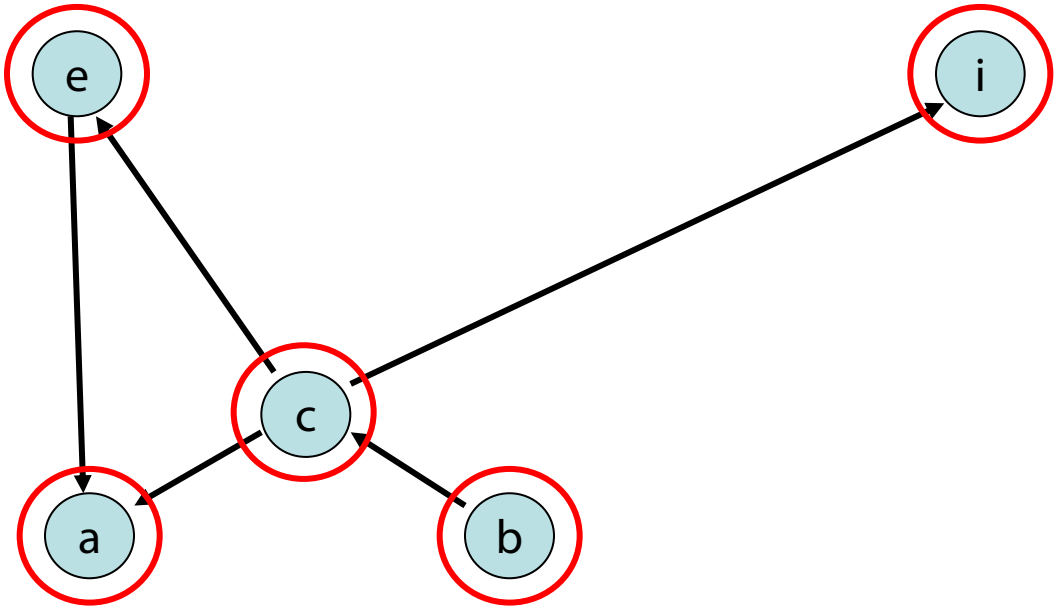
- Bestimme starke ZHKs



Transitive Hülle

Lösung 2: verwende ZHK-Algorithmus

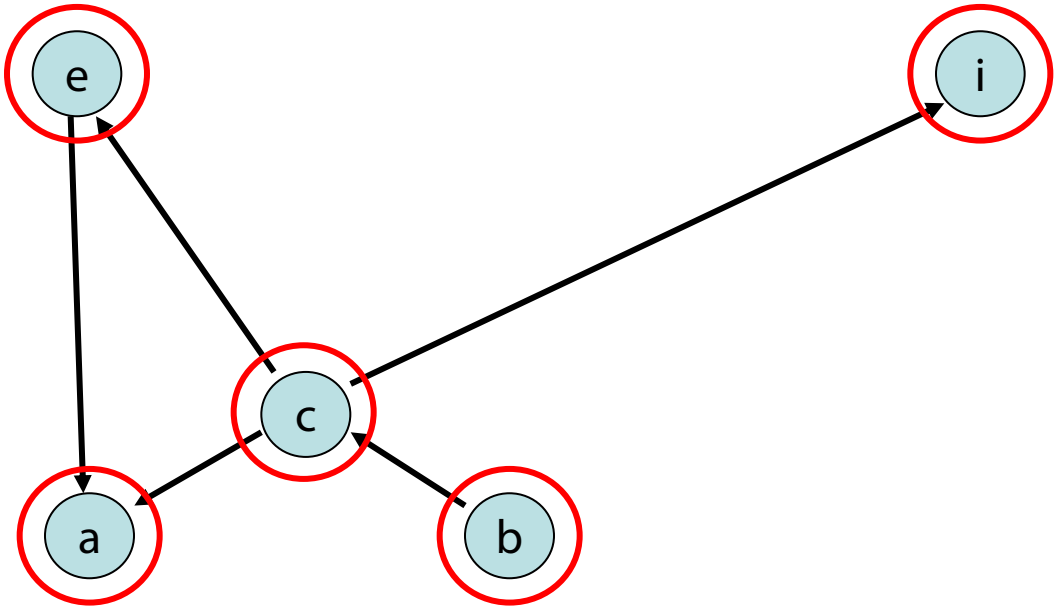
- Bestimme ZHK-Graph (Repräsentanten)



Transitive Hülle

Lösung 2: verwende ZHK-Algorithmus

- Wende APSP-Algo auf ZHK-Graph an



Transitive Hülle

Reachable(v,w):

- Bestimme Repräsentanten r_v und r_w von v und w
- $r_v = r_w$: gib 1 aus
- sonst gib Reachable(r_v,r_w) für ZHK-Graph zurück

Transitive Hülle

- Graph $G=(V,E)$: $n=|V|$, $m=|E|$
- ZHK-Graph $G'=(V',E')$: $n'=|V'|$, $m'=|E'|$

Datenstruktur:

- Berechnungszeit:
  $O(n + m + n' \cdot m' + (n')^2 \log n')$
- Speicher: $O(n + (n')^2)$

Reachable(v,w): Laufzeit $O(1)$

Transitive Hülle

Ist es auch möglich, mit ~ O(n+m) Speicher für die Datenstruktur die Operation Reachable(v,w) effizient abzuarbeiten?

Einsicht: Wenn für eine topologische Sortierung $(t_v)_{v \in V'}$ der Repräsentanten gilt $r_v > r_w$, dann gibt es keinen gerichteten Weg von r_v nach r_w

Was machen wir, falls $r_v < r_w$?

Transitive Hülle

Fall 1: Der ZHK-Graph ist eine gerichtete Liste

Reachable(v,w) ergibt 1 $\Leftrightarrow$ $t_v < t_w$

Transitive Hülle

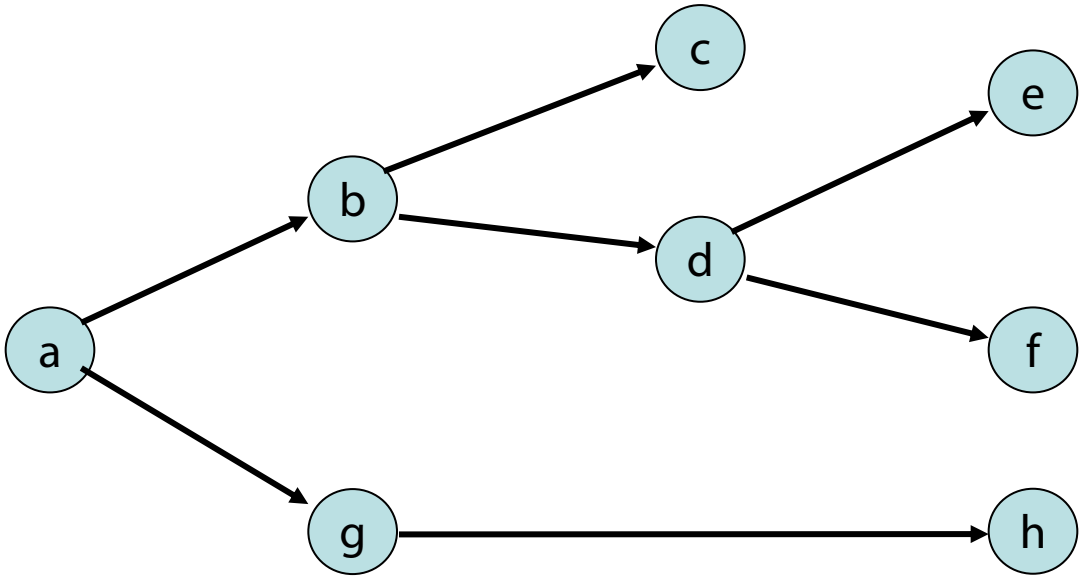
Fall 1: Der ZHK-Graph ist eine gerichtete Liste

Datenstruktur: O(n+m) Zeit, O(n) Speicher
(speichere Repräsentanten zu jedem Knoten und gib Repr. Ordnungsnummern)

Reachable(v,w): Laufzeit O(1)

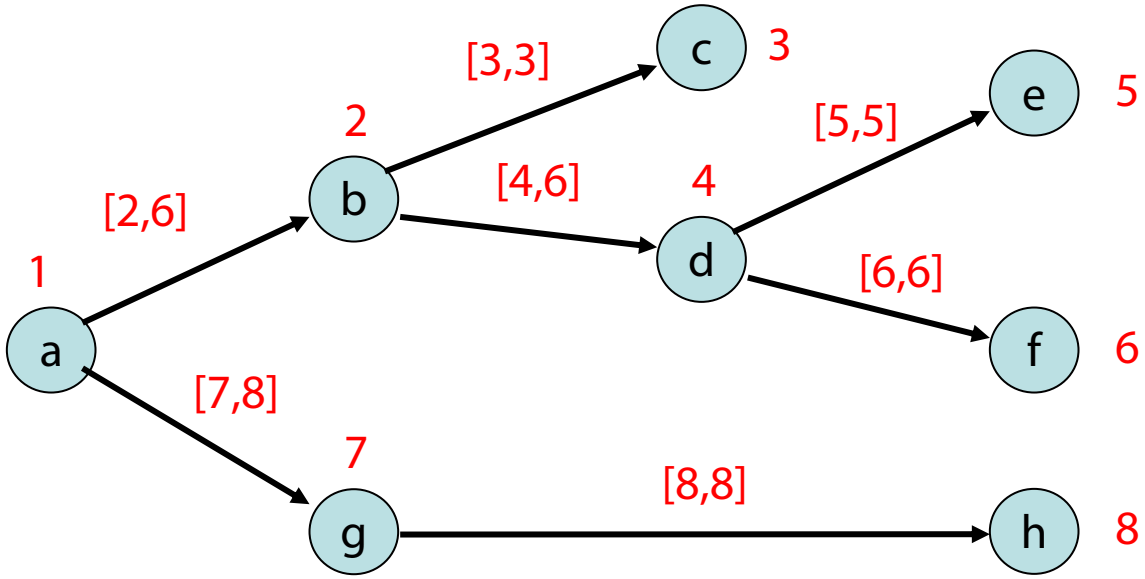
Transitive Hülle

Fall 2: Der ZHK-Graph ist ein gerichteter Baum



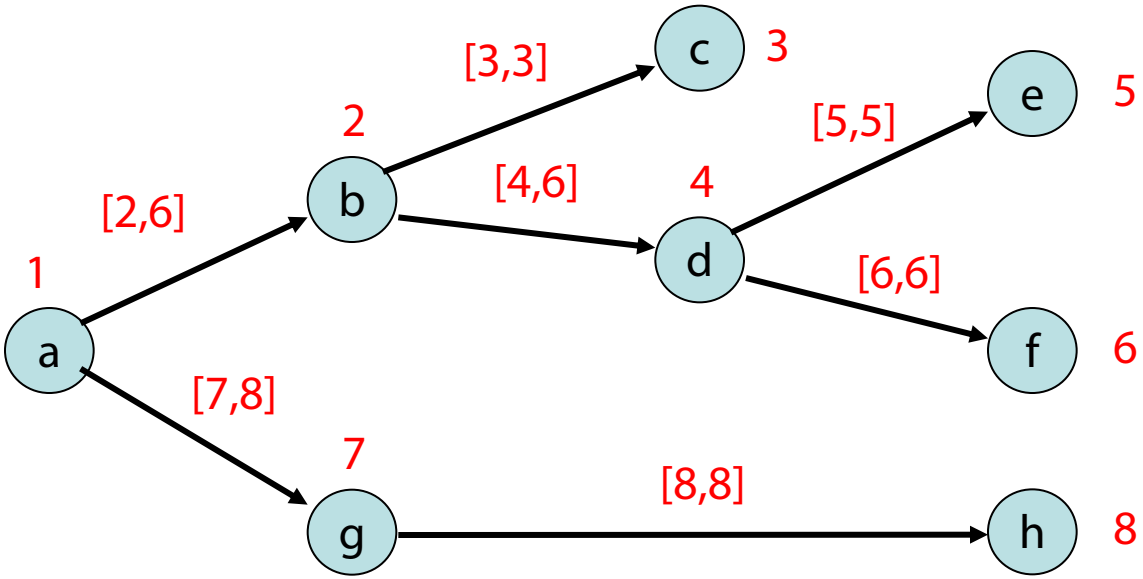
Transitive Hülle

Strategie: DFS-Durchlauf von Wurzel, Kanten mit dfsnum-Bereichen markieren



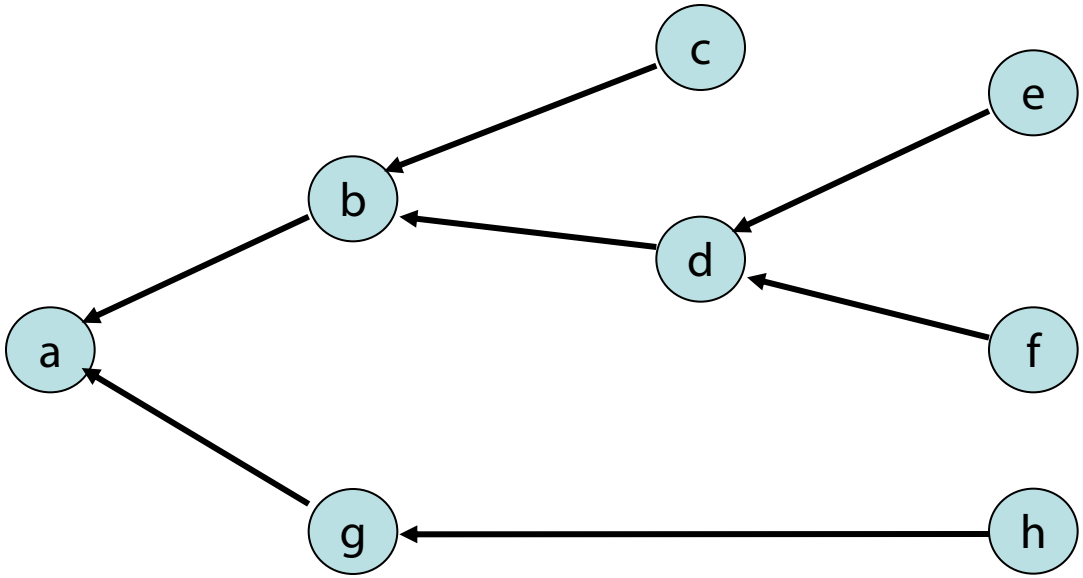
Transitive Hülle

Reachable(v,w): Bestimme Repräsentanten r_v und r_w, teste ob r_w in Intervall von ausgehender Kante von r_v



Transitive Hülle

Kantenrichtungen zur Wurzel:



Reachable(v,w) ist 1

⇔ Reachable(w,v) ist 1 für umgekehrte Richtungen

Transitive Hülle

Fall 2: Der ZHK-Graph ist ein gerichteter Baum

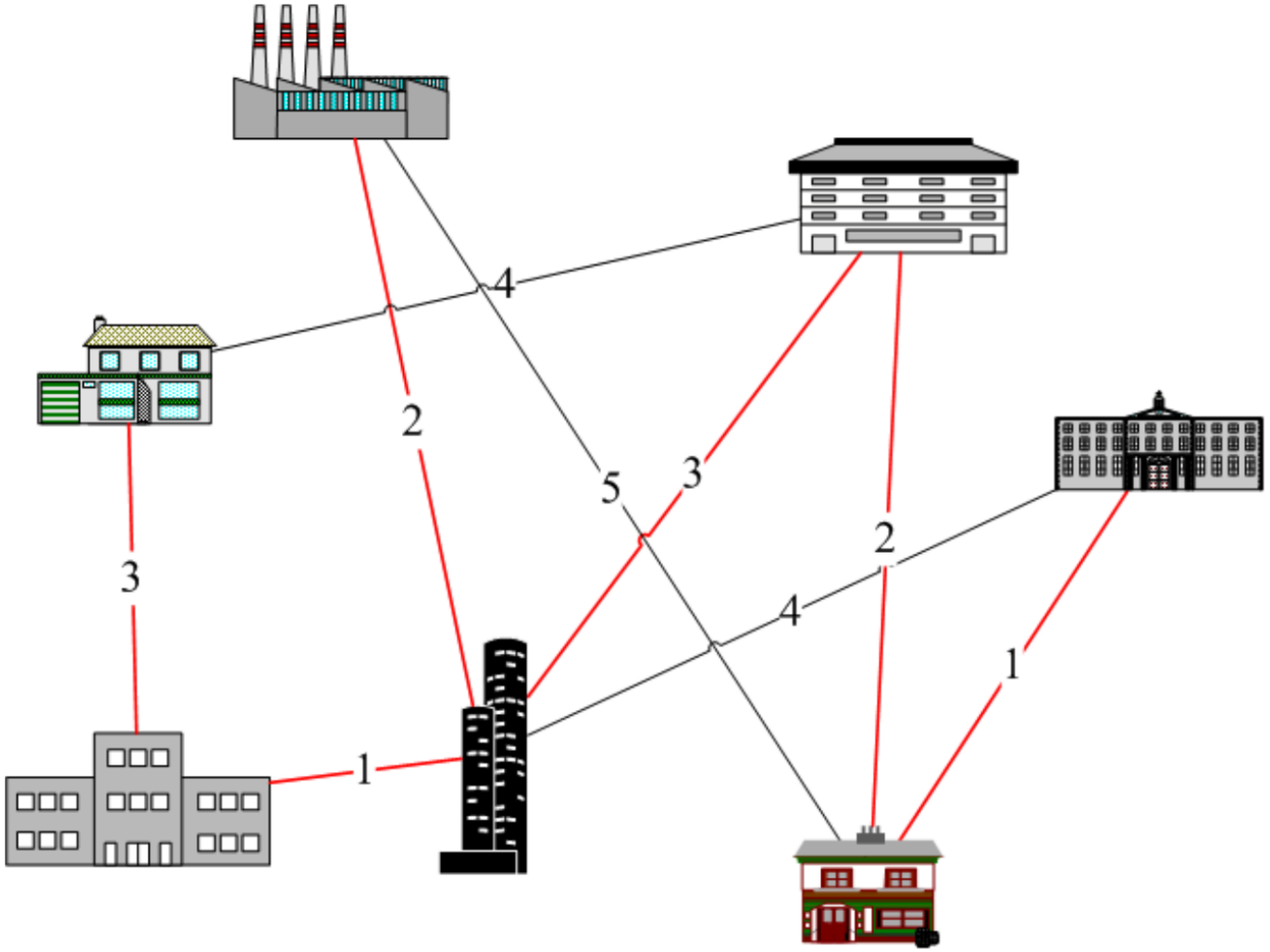
Datenstruktur: O(n+m) Zeit und Speicher
(speichere Repräsentanten zu jedem Knoten
Kantenintervalle zu jedem Repräsentanten)

Reachable(v,w): Laufzeit O(log d) (binäre Suche auf Intervallen), wobei d der maximale Grad im ZHK-Graph ist

Fall 3: Der ZHK-Graph ist ein beliebiger DAG

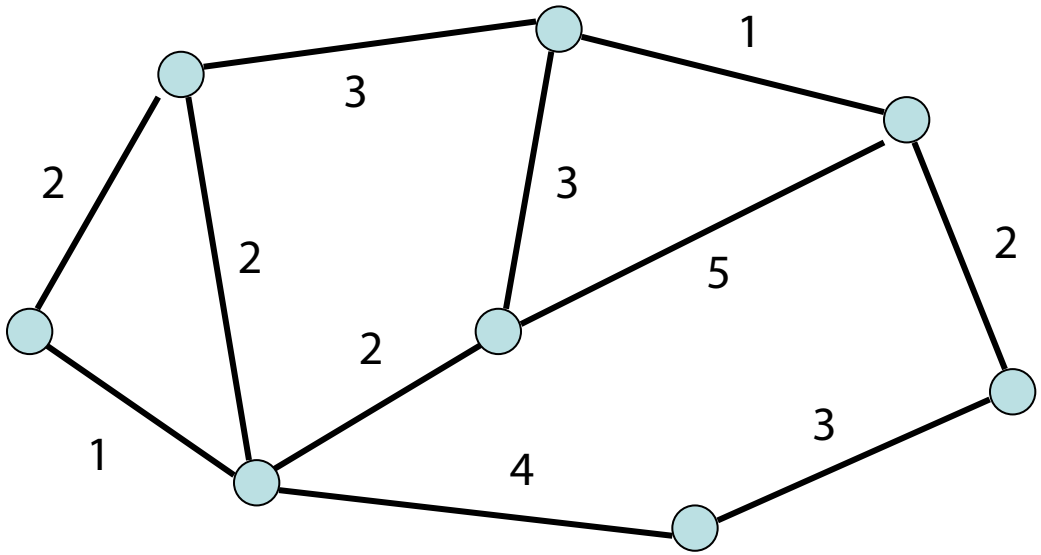
Geht in O(d log n) Zeit und $O(n^2)$ Speicher
(hier nicht vertieft)

Ein neues Anwendungsproblem



Minimaler Spannbaum

Zentrale Frage: Welche Kanten muss ich nehmen, um mit minimalen Kosten alle Knoten zu verbinden?



Anwendungen in der Praxis

- Erstellung von kostengünstigen zusammenhängenden Netzwerken
  - Beispielsweise Telefonnetze oder elektrische Netze
- Computernetzwerke mit redundanten Pfaden:
  - Spannbäume genutzt zum Routing und dabei zur Vermeidung von Paketverdopplungen

Minimaler Spannbaum

Eingabe:

- ungerichteter Graph $G=(V,E)$
- Kantenkosten $c : E \longrightarrow \mathbb{R}_+$

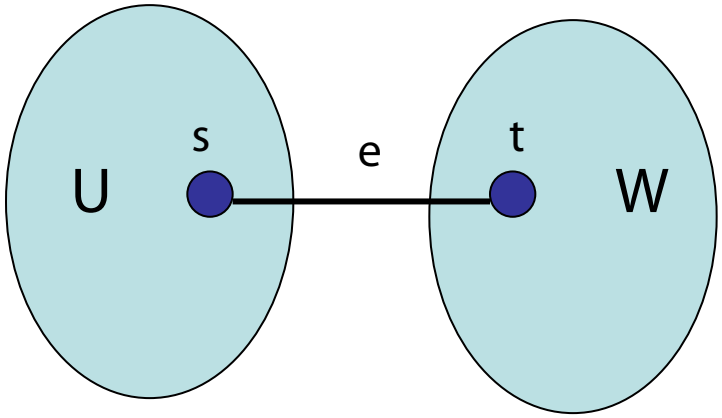
Ausgabe: $\text{argmin}_{T \subseteq E \wedge (V,T) \text{ verbunden}} \Sigma_{e \in T} c(e)$

- Teilmenge $T \subseteq E$, so dass Graph (V,T) verbunden und $c(T)=\Sigma_{e \in T} c(e)$ minimal

- T formt immer einen Baum (wenn c positiv).
- Baum über alle Knoten in V mit minimalen Kosten: minimaler Spannbaum (MSB)

Minimaler Spannbaum

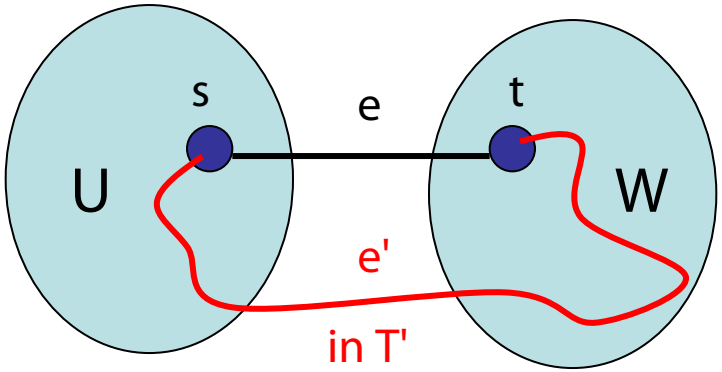
Beh 1: Sei (U,W) eine Partition von V (d.h. $U \cup W = V$ und $U \cap W = \emptyset$) und e={s,t} eine Kante mit minimalen Kosten mit $s \in U$ und $t \in W$. Dann gibt es einen minimalen Spannbaum (MSB) T, der e enthält.



Minimaler Spannbaum

Beweis von Beh 1:

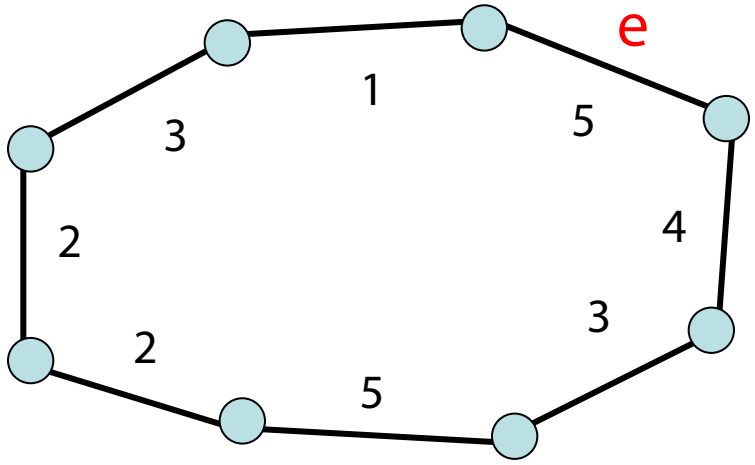
- Betrachte beliebigen MSB T'
- e={s,t}: (U,W)-Kante minimaler Kosten



- Ersetzung von e' durch e führt zu Baum T'', der höchstens Kosten von MSB T' hat, also MSB ist

Minimaler Spannbaum

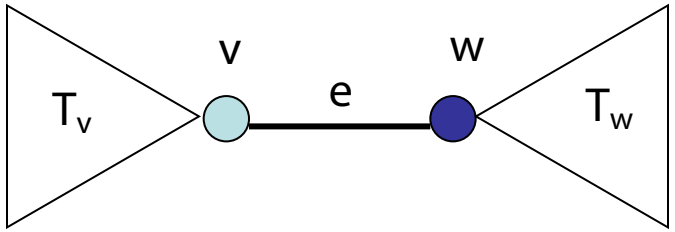
Beh 2: Betrachte beliebigen Kreis C in G und sei e Kante in C mit maximalen Kosten. Dann ist jeder MSB in G ohne e auch ein MSB in G.



Minimaler Spannbaum

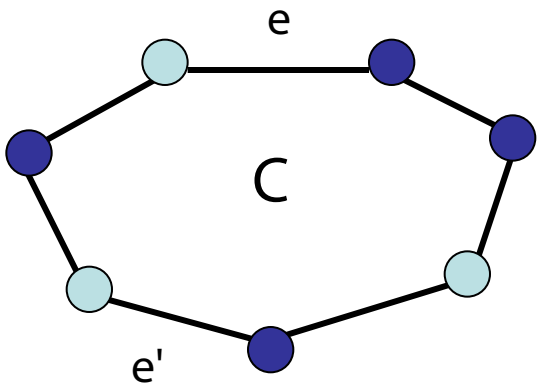
Beweis von Beh 2:

- Betrachte beliebigen MSB T in G
- Angenommen, T enthalte e



e maximal für C

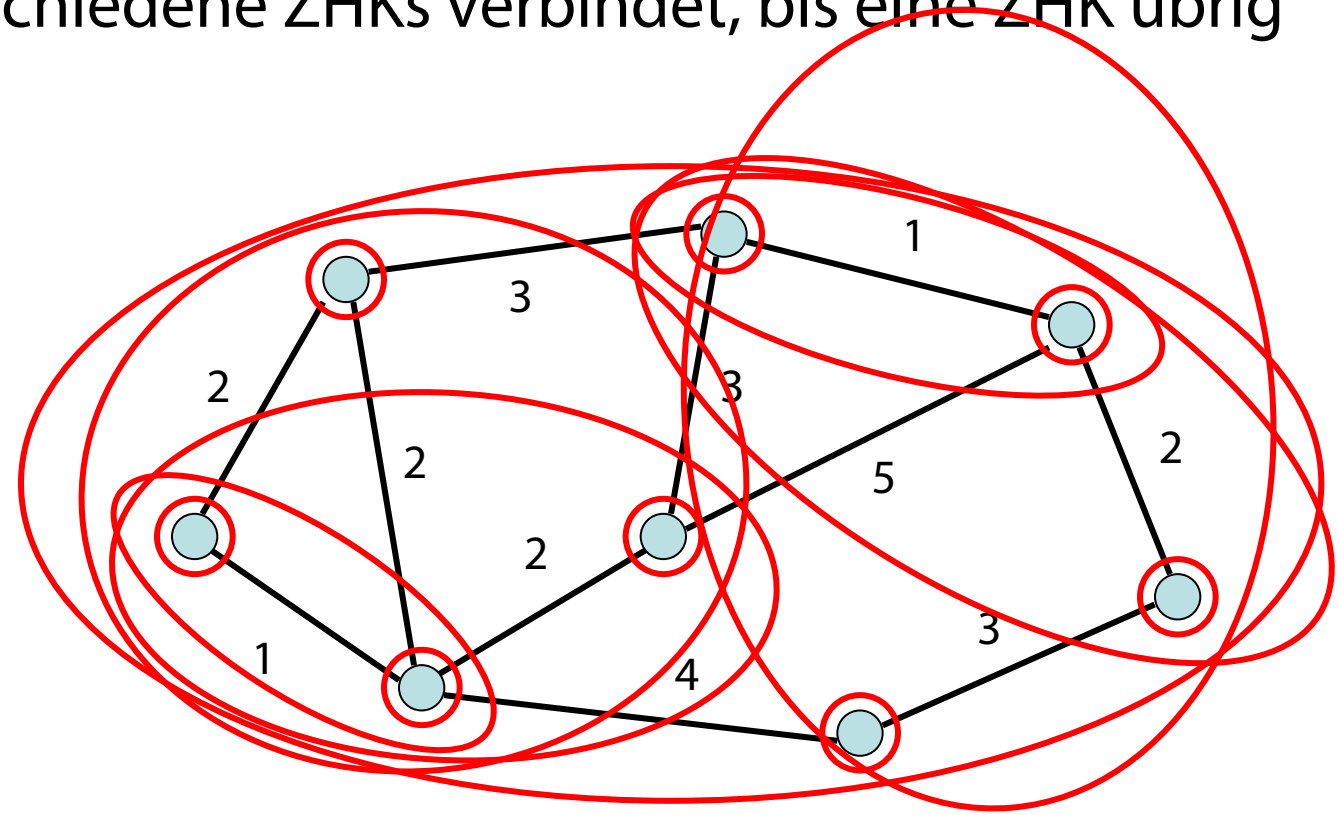
- ○ : zu T_v, ● : zu T_w
  - es gibt e' von T_v nach T_w
  - Ersetzung $e \longrightarrow e'$ ergibt MSB T' ohne e



Minimaler Spannbaum

Regel aus Beh 1:

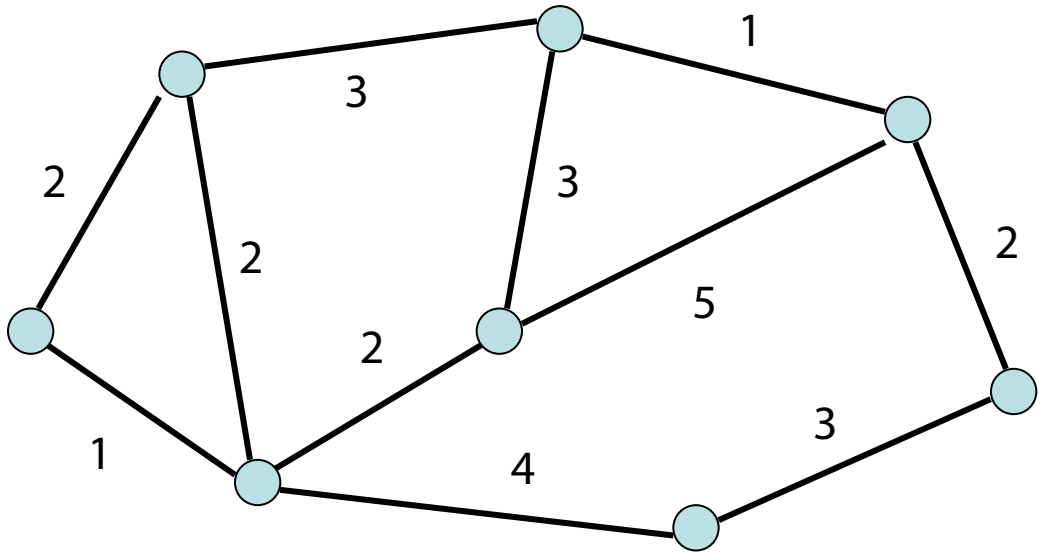
Wähle wiederholt Kante mit minimalen Kosten, die verschiedene ZHKs verbindet, bis eine ZHK übrig



Minimaler Spannbaum

Regel aus Beh 2:

Lösche wiederholt Kante mit maximalen Kosten, die Zusammenhang nicht gefährdet, bis ein Baum übrig



Minimaler Spannbaum

Problem: Wie implementiert man die Regeln effizient?

Strategie aus Beh 1:

- Setze T=Ø und sortiere die Kanten aufsteigend nach ihren Kosten
- Für jede Kante (u,v) in der sortierten Liste, teste, ob u und v bereits im selben Baum in T sind. Falls nicht, füge (u,v) zu T hinzu.

benötigt Union-Find DS

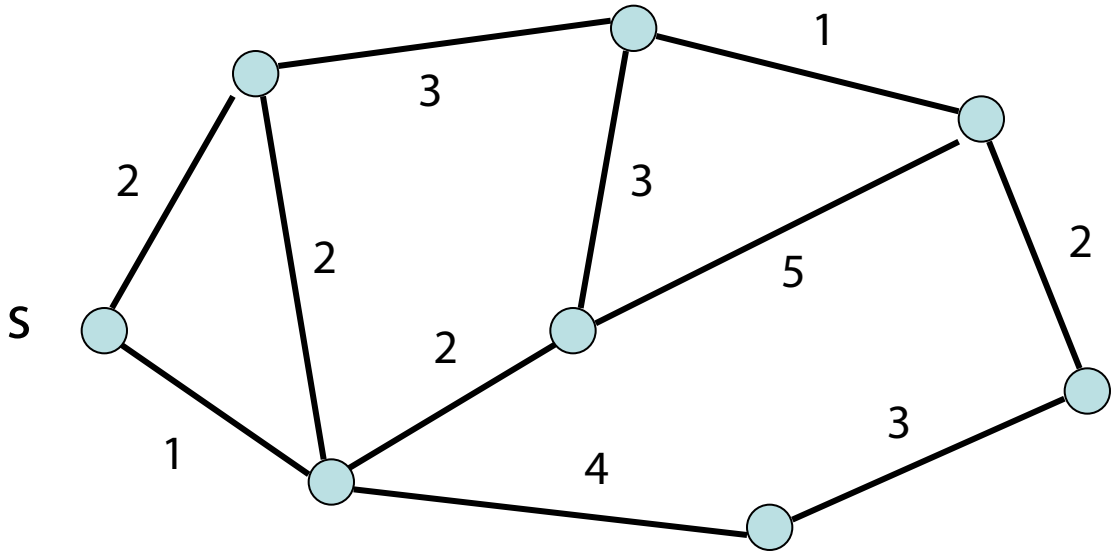
Erinnerung: Union-Find DS

Operationen:

- Union(x_1, x_2): vereinigt die Elemente in den Teilmengen T_1 und T_2, zu denen die Elemente x_1 und x_2 gehören, zu $T=T_1 \cup T_2$
- Find(x): gibt (eindeutigen) Repräsentanten der Teilmenge aus, zu der Element x gehört

Minimaler Spannbaum

Beispiel: (—: Kanten im MSB)



Kruskal-Algorithmus

```
function MinSpanningTree((V, E), c):
    T:={ }
    init(V) // initialisiere einelem. Mengen für V
    S:=mergesort(E)  // aufsteigend sortiert
    for {u,v}∈ S do
      if find(u)≠find(v) then // versch. Mengen
         T:=T ∪ { {u,v} }
         union(u, v)  // u und v in einer Menge
    return T
```

Joseph Kruskal: On the shortest spanning subtree and the traveling salesman problem. In: Proceedings of the American Mathematical Society. 7, S. 48–50, **1956**

Kruskal-Algorithmus

Laufzeit:

- Mergesort: $O(m \log m)$ Zeit
- $2m$ Find-Operationen und $n-1$ Union-Operationen: $O(m \cdot \log^* n)$ Zeit

Insgesamt Zeit $O(m \log m)$.

- Mit Sortieren durch Verteilen (Counting Sort, Bucket Sort…) weiter reduzierbar bei "kleinen" Graphen und Kantenkosten
- Dann dominiert $O(m \cdot \log^* n)$

Minimaler Spannbaum

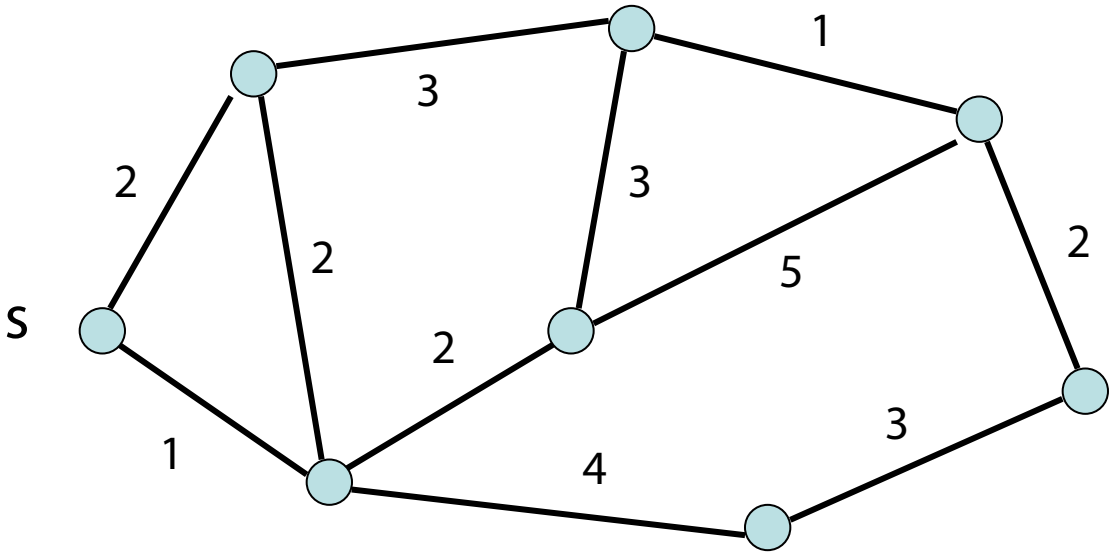
Alternative Strategie (motiviert aus Beh 2):

- Starte bei beliebigem Knoten s, MSB T besteht anfangs nur aus s
- Ergänze T durch günstigste Kante zu äußerem Knoten w und füge w zu T hinzu bis T alle Knoten im Graphen umfasst

Jarník, V., "O jistém problému minimálním" [About a certain minimal problem], Práce Moravské Přírodovědecké Společnosti (in Czech) 6: S. 57–63, **1930**
Prim, R. C.., "Shortest connection networks And some generalizations", Bell System Technical Journal 36 (6): S. 1389–1401, **1957**
Dijkstra, E. W., "A note on two problems in connexion with graphs“, Numerische Mathematik 1: S. 269–271, **1959**

Minimaler Spannbaum

Beispiel:



Jarnik-Prim Algorithmus

```
procedure JarnikPrim(s: NodeId)
    d=<∞,…,∞>: Array of ℝ ∪ {-∞, ∞}
    parent=<⊥,…,⊥>: Array of NodeId
    d[s]:=0; parent[s]:=s // T anfangs nur aus s
    q=<s>: PQ with key as lambda(x) d[x]
    while not mtQueue?(q) do
       u:= deleteMin(q) // u: min. Distanz zu T in q
       for e={u,v} ∈ E do
            if c(e) < d[v] then // aktualisiere d[v] zu T
                d := d[v]; d[v] := c(e); parent[v] := u
                if d=∞ // v noch nicht in q?
                    then insert(v, q)
                    else decreaseKey(v, d-d[v], q)
     return constructTree(s, parent)
```

Jarnik-Prim Algorithmus

Laufzeit:

$T_{JP} = O(n(T_{DeleteMin}(n)+T_{Insert}(n)) + m \cdot T_{decreaseKey}(n))$

Binärer Heap: alle Operationen $O(\log n)$, also

$T_{JP} = O((m+n)\log n)$

Fibonacci Heap:

- $T_{DeleteMin}(n)=T_{Insert}(n)=O(\log n)$
- $T_{decreaseKey}(n)=O(1)$
- Damit $T_{JP} = O(n (\log n) + m)$

Vergleich: $O(m \log m)$ bei Kruskal ($m > n$)

Zusammenfassung

- Transitive Hülle
- Minimaler Spannbaum
  - Kruskal-Algorithmus
  - Jarnik-Prim-Algorithmus