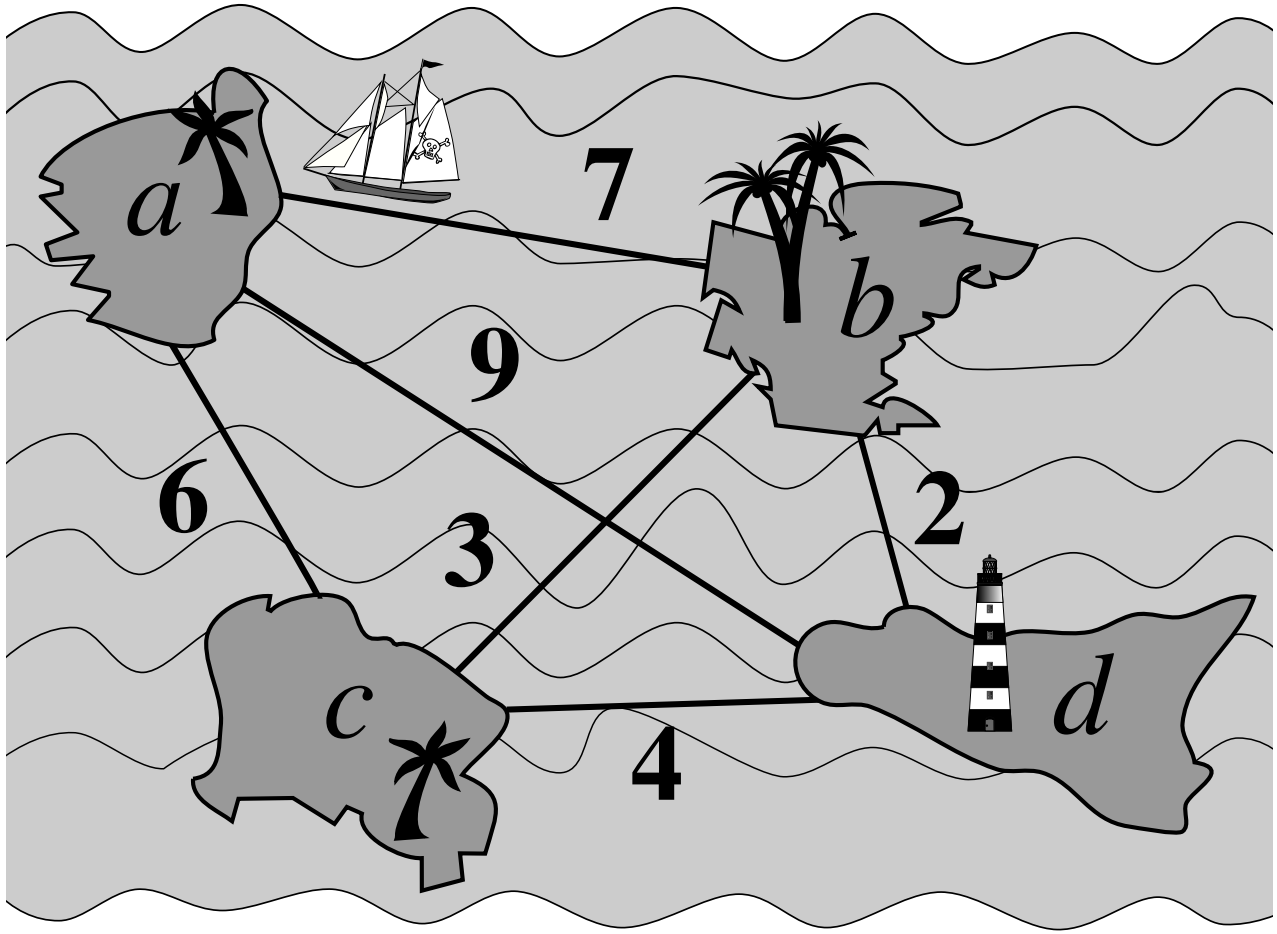
11 Minimale Spannbäume



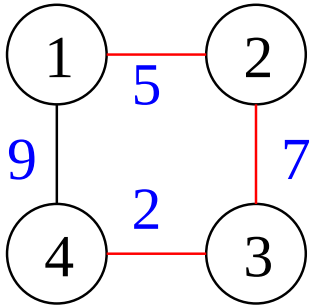
Minimale Spannbäume

ungerichteter Graph $G = (V, E)$.

Knoten V, $n = |V|$, e.g., $V = \{1, \ldots, n\}$

Kanten $e \in E$, $m = |E|$, two-element subsets of V.

Kantengewichte $c(e) \in \mathbb{R}_+$.



G ist zusammenhängend, d.h., $\exists$ Pfad zwischen beliebigen Knoten.
Finde Baum (V, T) mit minimalem Gewicht $\sum_{e \in T} c(e)$ der alle Knoten verbindet.

Falls G nicht zusammenhängend, finde

minimalen spannenden Wald T der alle

Zusammenhangskomponenten von G aufspannt.

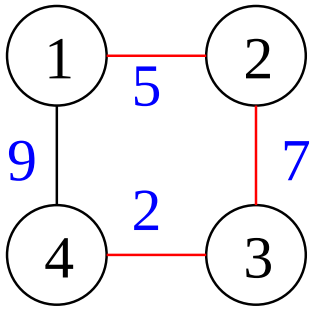
Anwendungen

- Netzwerk-Entwurf
- Bottleneck-Shortest-Paths:

  Suche s–t-Pfad,

  dessen max. Kantengewicht minimal ist.

  Dies ist der Pfad im MST!



- Clustering: Lass schwere MST-Kanten weg.

  Teilbäume definieren Cluster

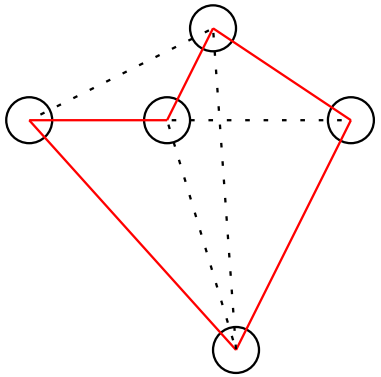
- 2-Approximation Handlungsreisendenproblem

  (auch untere Schranken) siehe Info III WS 06/07.

- 2-Approximation Steinerbaumproblem, siehe VL Approx- und Onlinealgorithmen

Beispiel: Handlungsreisendenproblem

[Der Handlungsreisende - wie er sein soll und was er zu thun hat, um Auftraege zu erhalten und eines gluecklichen Erfolgs in seinen Geschaeften gewiss zu sein - Von einem alten Commis-Voyageur, 1832].



Gegeben ein Graph $G = (V, V \times V)$, finde einen einfachen Kreis $C = (v_1, v_2, \ldots, v_n, v_1)$ so dass $n = |V|$ und $\sum_{(u,v)\in C} d(u,v)$ minimiert wird.

2-Approximation: Bastle Tour aus MST

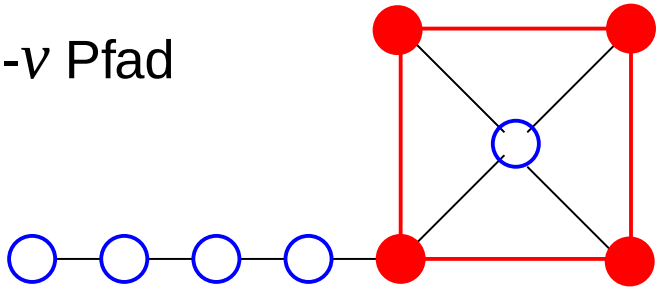
Steinerbäume [C. F. Gauss 18??]

Geg: Graph $G = (V, E)$, mit positiven Kantengewichten $c : E \to \mathbb{R}_+$

$V = R \cup F$, i.e., Pflichtknoten and Steinerknoten.

Finde $T \subseteq E$ der mit minimalen Kosten alle Pflichtknoten verbindet.

$\forall u, v \in R$: T enthält u-v Pfad



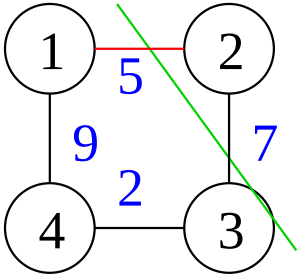
DAS Netzwerkentwurfsproblem

2-Approximation: MST in $(R, R \times R)$ mit

$w(s,t) = d(s,t)$ (kürzester Weg, metrische Vervollständigung)

MST-Kanten auswählen und verwerfen

Die Schnitteigenschaft (Cut Property)

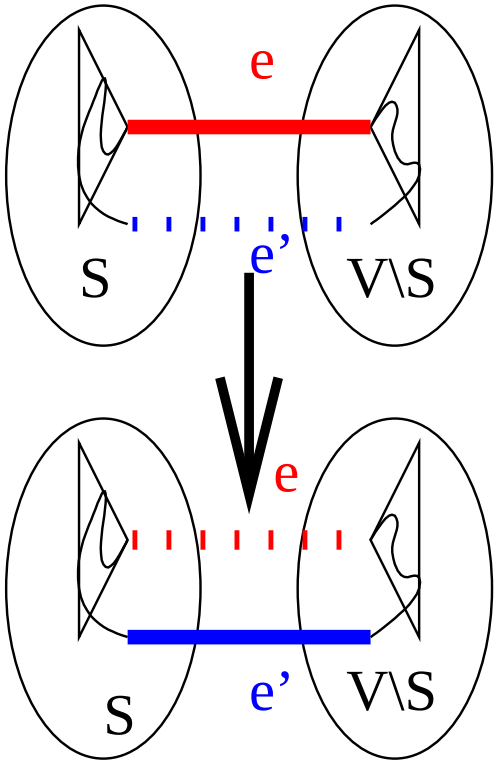


Für jede Teilmenge $S \subset V$ betrachte die Schnittkanten

$C = \{\{u,v\} \in E : u \in S, v \in V \setminus S\}$

Die leichteste Kante in C

kann in einem MST verwendet werden.



Beweis:

Angenommen MST T' benutzt Kante e' zwischen S und $V \setminus S$ mit $c(e) \leq c(e')$.

Dann ist $T = T' \setminus \{e'\} \cup \{e\}$ auch ein MST.

(Es gilt sogar, $c(e) = c(e')$.)

Die Kreiseigenschaft (Cycle Property)

Kreis: eine Kantenmenge, die einen zyklischen Weg

$\langle v_1, v_2, \ldots, v_k, v_1 \rangle$ definiert

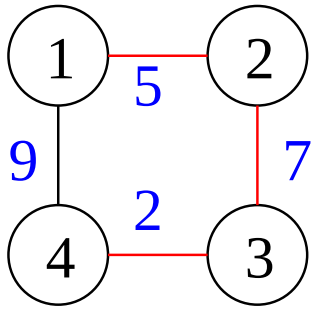
Die schwerste Kante auf einem Kres wird nicht für einen MST benötigt

Beweis.

Angenommen MST T' benutzt die schwerste Kante e' auf Kreis C mit $c(e) \leq c(e')$.

Dann ist $T = T' \setminus \{e'\} \cup \{e\}$ auch ein MST.

(Es gilt sogar, $c(e) = c(e')$.)



□

Der Jarník-Prim Algorithmus

[Jarník 1930, Prim 1957]

Idee: Lasse einen Baum wachsen

$T := \emptyset$

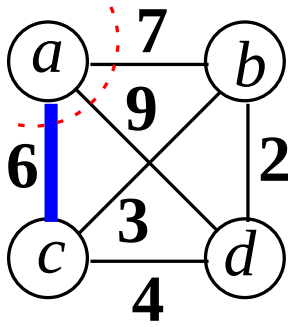
$S := \{s\}$ for arbitrary start node s

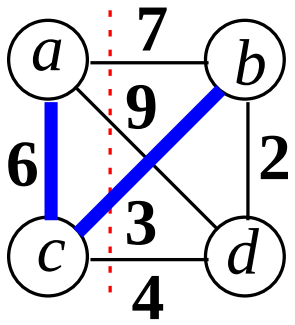
repeat $n-1$ times

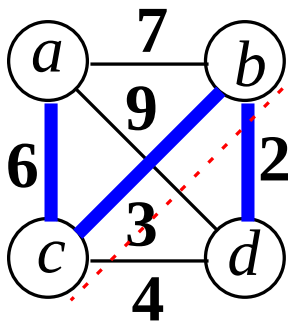
find (u,v) fulfilling the cut property for S

$S := S \cup \{v\}$

$T := T \cup \{(u,v)\}$







Function jpMST : Set **of** Edge

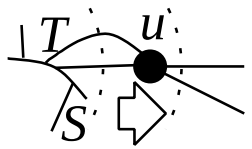
$d = \langle \infty, \ldots, \infty \rangle$: NodeArray$[1..n]$ **of** $\mathbb{R} \cup \{\infty\}$

parent : NodeArray **of** NodeId

Q : NodePQ; Q.insert(s) for some arbitrary $s \in V$

while $Q \neq \emptyset$ **do**

u := Q.deleteMin$()$

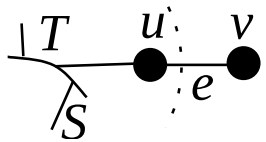


$d[u] := 0$

foreach edge $e = (u,v) \in E$ **do**

if $c(e) < d[v]$ **then**

$d[v] := c(e)$



parent$[v]$:= u

if $v \in Q$ **then** Q.decreaseKey(v) **else** Q.insert(v)

invariant $\forall v \in Q : d[v] = \min\{c((u,v)) : (u,v) \in E \wedge d[u] = 0\}$

return $\{(v, \text{parent}[v]) : v \in V \setminus \{s\}\}$

Analyse

- $O(m+n)$ Zeit ausserhalb der PQ
- n deleteMin (Zeit $O(n \log n)$)
- $O(m)$ decreaseKey (Zeit $O(1)$ amortisiert)

$\rightsquigarrow$ $O(m + n \log n)$ mit Fibonacci Heaps

- Im Durchschnitt: Zeit $O\left(m + n \log \frac{m}{n} \log n\right)$ mit binären Heaps

  Beweis: analog Dijkstra

Kruskals Algorithmus [1956]

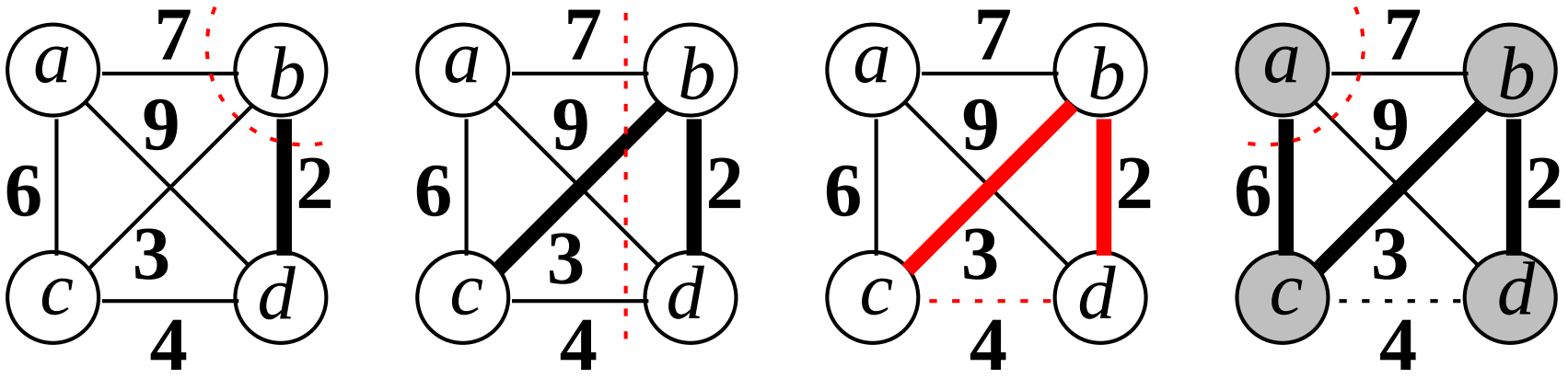
$T := \emptyset$ // subforest of the MST

foreach $(u,v) \in E$ in ascending order of weight **do**

  if u and v are in different subtrees of T **then**

    $T := T \cup \{(u,v)\}$ // Join two subtrees

return T



Union-Find Datenstruktur

Class UnionFind$(n : \mathbb{N})$

parent$=\langle 1, 2, \ldots, n\rangle$: **Array** $[1..n]$ **of** $1..n$

rank$=\langle 0, \ldots, 0\rangle$: **Array** $[1..n]$ **of** $0..\log n$

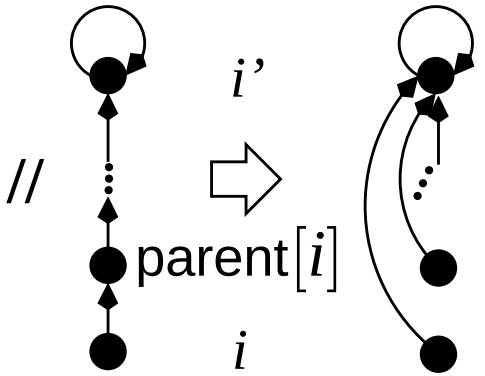
Function find$(i : 1..n) : 1..n$

if parent$[i] = i$ **then return** i

else $i' :=$ find(parent[i])

parent$[i] := i'$

return i'

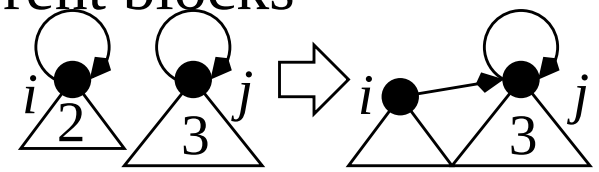


Union-Find Datenstruktur

Procedure link$(i, j : 1..n)$

assert i and j are representatives of different blocks

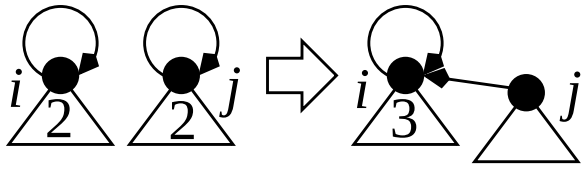
if rank$[i] <$ rank$[j]$ **then** parent$[i] := j$



else

parent$[j] := i$

if rank$[i] =$ rank$[j]$ **then** rank$[i]{+}{+}$



Procedure union$(i, j : 1..n)$

if find$(i) \neq$ find(j) **then** link(find(i), find(j))

Analyse

invariant parent-Zeiger führen uns zu eindeutigen Partitions-Repräsentanten

invariant Der Pfad zum Repr. x hat Länge höchstens gen$[x]$

invariant x ist Repr. $\Rightarrow$ x's Menge hat Größe mindestens $2^{\text{gen}[x]}$

Korollar 1. find *braucht Zeit* $O(\log n)$

Eigentlich: Zeit $O(\alpha_T(\#\text{find}, n))$ mit

$\alpha_T(m,n) = \min\{i \geq 1 : A(i, \lceil m/n \rceil) \geq \log n\}$

(inverse Ackermannfunktion)

Kruskal using Union-Find

Assume $V = 1..n$

Tc : UnionFind(n) // encodes components of forest T

foreach $(u,v) \in E$ in ascending order of weight **do** // sort

  if Tc.find$(u) \neq$ Tc.find(v) **then**

    output $\{u,v\}$

    Tc.union(u,v)

  //otherwise the cycle property excludes $\{u,v\}$

Union and find take “almost” constant amortized time

(a fascinating story by itself)

Time $O(m \log m)$. Faster for integer weights.

Graph representation: sequence of edges

Mehr MST-Algorithmen

- Zeit $O(m \log n)$ [Boruvka 1926]

  Zutat vieler fortgeschrittener Algorithmen

- Erwartete Zeit $O(m)$ [Karger Klein Tarjan 1995],

  parallelisierbar, externalisierbar

- Det. Zeit $O(m\alpha_T(m,n))$ [Chazelle 2000]

- “optimaler” det. Algorithmus [Pettie, Ramachandran 2000]

- Praktikabler externer Algorithmus [Sanders Schultes Sibeyn 2004]

  $O\left(\text{sort}\left(m \log \frac{n}{M}\right)\right)$ I/Os