Kap. 6.5: Minimale Spannbäume

Professor Dr. Petra Mutzel
Lehrstuhl für Algorithm Engineering, LS11
Fakultät für Informatik, TU Dortmund

19./20. VO DAP2 SS 2009 30.6./2.7.2009

Anmeldung zur Klausur 31.07.2009 um 10:15 Uhr

- Informatik AI/KI Bacc+Diplom: BOSS System 16.7.-27.7.
- Informatik AI/KI Diplom: BOSS System ab sofort
- IKT/ET:Bacc/Diplom: BOSS System?
- Physik Bacc/Diplom: ihr Dekanat
- Datenanalyse und Datenmanagement / Statistik Bacc/ Diplom: ihr eigenes Prüfungsamt?
- Mathematik Bacc/Diplom: BOSS System?
- Lehramt, BaMaLa: direkt bei uns?

Bei Problemen mit BOSS: schriftlich beim Prüfungsamt anmelden

Nachtest für Ausnahmefälle

- **Di 14. Juli 2009, voraussichtlich 16:00 Uhr, OH14, R. 202**
- Anmeldung bis 9. Juli erforderlich via Email an Nicola Beume
- Stoff: Alles bis inkl. Hashing (inkl.1.-10. Übungsblatt)
- Teilnahmevoraussetzungen:
  - Attest/Entschuldigt beim 1. oder 2. Übungstest
  - oder besondere Härtefälle anerkannt in meiner Sprechstunde Di 7.7.09,14:15 Uhr-15:15 Uhr

Überblick

- Minimale Spannbäume (Einführung)

- Algorithmus von Kruskal

- ADT Union Find
- extrem langsam wachsende Funktion $\alpha(n)$

Motivation

„Was gibt es heute Besonderes?"
einfacher und schöner Greedy-Algorithmus

„Was gibt es heute Besonderes?"
ADT UNION-FIND: „So kam ich zur Informatik"

„Und was noch?"
extrem langsam wachsende Funktion

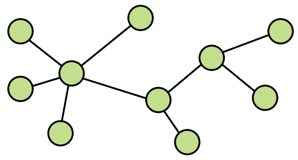
Kap. 6.5 Minimale Spannbäume

Achtung: in diesem Abschnitt **ungerichtete gewichtete** Graphen! ⚠

ungerichteter Graph $G=(V,E)$
Gewichtsfunktion $w : E \to R$
Kantengewichte: Kosten, Länge, …

Bäume

Definition: Ein ungerichteter Graph heißt *Wald* (engl. *forest*), wenn er kreisfrei ist. Wenn er zusätzlich zusammenhängend ist, dann heißt er *Baum* (engl. *tree*).

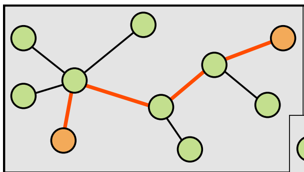


auch: *freier* Baum (kein *gewurzelter* Baum)

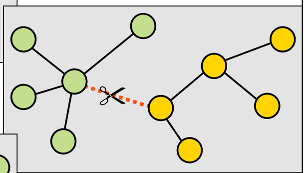
Theorem: Sei G=(V,E) ungerichteter Graph (ohne Mehrfachkanten, Schleifen). Dann sind äquivalent:

1) G ist ein Baum
2) Jedes Paar von Knoten ist durch einen eindeutigen Weg verbunden.
3) G ist zshgd., zerfällt aber durch Entfernen einer beliebigen Kante in zwei Komponenten.
4) G ist zshgd. und $|E| = |V|-1$
5) G ist kreisfrei und $|E|=|V|-1$
6) G ist kreisfrei, aber durch Hinzufügen einer beliebigen Kante entsteht ein Kreis.

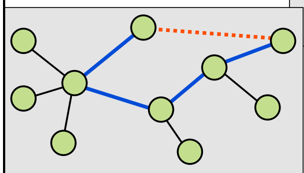
Erläuternde Bilder zum Theorem



zu 2)



zu 3)

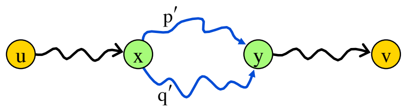


zu 6)

Beweis: durch Ringschluss 1) ➔ 2) ➔ … 6) ➔ 1)

1) ➔ 2)

- Da Baum zshgd. ➔ ex. Weg zwischen u und v
- Annahme: zwei Wege p und q



- ➔ $p' + q'$ bilden Kreis! Widerspruch

2) ➔ 3)

- zusammenhängend klar
- Annahme: G zerfällt nach Löschen von (u,v) nicht
- ➔ Ex. Weg zwischen u und v, der (u,v) nicht benutzt
- ➔ Weg nicht eindeutig! Widerspruch

3) ➔ 4)

- zusammenhängend klar
- ➔ G besitzt DFS-Baum mit $|V|-1$ T-Kanten
- ➔ Falls G mehr Kanten besitzt ➔ ex. B-Kante
- Durch Löschen einer B-Kante zerfällt G aber nicht
- ➔ G hat $|V|-1$ Kanten

4) ➔ 5)

- Da G zshgd. und $|E| = |V|-1$
  ➔ die T-Kanten eines DFS-Baums sind genau die Kanten von G
- ➔ keine B-Kanten
- ➔ G kreisfrei

5) ➔ 6)

- G kreisfrei und besteht aus k Komponenten
- ➔ G hat $|V|-k$ Kanten
- ➔ $k=1$ (d.h. G ist zusammenhängend)
- ➔ Hinzufügen einer Kante erzeugt Kreis

6) ➔ 1)

- Sei u,v beliebiges Knotenpaar
- ➔ Entweder $(u,v) \in E$ oder
  - Hinzufügen von (u,v) erzeugt Kreis
  - ➔ ex. Weg von u nach v in G
- ➔ G ist zusammenhängend

Spannbaum

Definition: Sei G=(V,E) ein ungerichteter, zshgd. Graph. Ein Untergraph $T=(V,E_T)$ von G heißt *Spannbaum* (engl. *spanning tree*) von G, falls T ein Baum ist.

Gewicht eines Spannbaums:

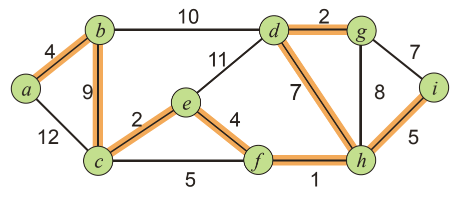
$$w(T) := w(E_T) := \sum_{e \in E_T} w(e)$$

Minimaler Spannbaum

Minimum Spanning Tree (MST)	
Gegeben:	ungerichteter, zshgd. Graph G = (V,E) Gewichtsfunktion $w : E \rightarrow R$
Gesucht:	ein Spannbaum T von G mit minimalem Gewicht w(T)

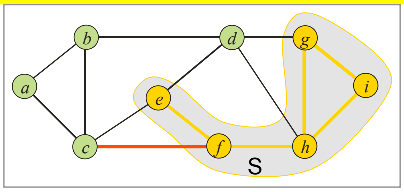
Anwendung: Konstruktion von Netzwerken

Beispiel für MST

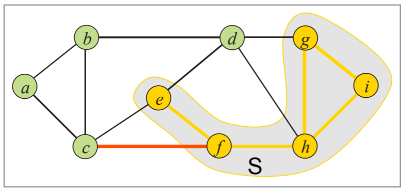


Definition:

- Eine Teilmenge E' der Kanten heißt *aussichtsreich*, wenn es einen MST gibt, der alle Kanten aus E' enthält.
- Eine Kante (u,v) *verlässt* eine Knotenmenge $S \subset V$, wenn $u \in S$ und $v \notin S$



- $E' := \emptyset$ ist immer aussichtsreich.
- E' aussichtsreich und $|E'| = |V|-1$
  ➔ $G' := (V,E')$ ist ein MST

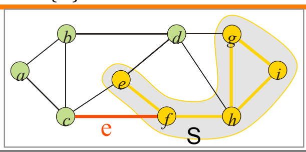


MST Eigenschaft

Lemma: Sei G=(V,E) zshgd. mit Gewichtsfunktion w : E→R. Seien:

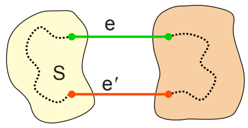
- $S \subset V$
- $T \subset E$ aussichtsreich und *keine* Kante aus T verlässt S
- $e \in E$ eine Kante mit minimalem Gewicht, die S verlässt

Dann ist $T \cup \{e\}$ aussichtsreich.



Beweis:

- Ex. MST $T=(V,E_T)$ mit $E' \subseteq E_T$
- Falls $e \in E_T \Rightarrow$ fertig!
- Sonst: $E_T \cup \{e\}$ enthält Kreis:



- $\Rightarrow$ Ex. Kante $e' \in E_T$ auf Kreis, die S verlässt
- Da $w(e) \leq w(e') \Rightarrow T - e' + e$ ist ein MST
- $\Rightarrow E' \cup \{e\}$ ist aussichtsreich

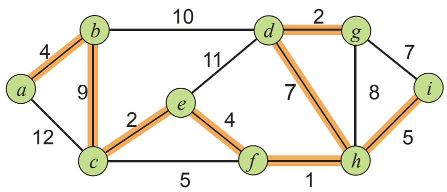
6.5.1 Algorithmus von Kruskal

Idee:

- Sortiere die Kanten nach aufsteigendem Gewicht: $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$
- $T := \emptyset$
- Für $i := 1,\ldots,m$:
- Falls $T \cup \{e_i\}$ kreisfrei, dann: $T := T \cup \{e_i\}$

Greedy-Algorithmus („gefräßig"): iterative Konstruktion einer Lösung, die immer um die momentan besten Kandidaten erweitert wird.

Beispiel für Kruskal



Korrektheit von Kruskal

Theorem:
Sei G=(V,E) zshgd. mit Gewichtsfunktion w : E→R. Dann berechnet der Algorithmus von Kruskal einen MST von G.

Beweis:
- T ist aufspannender Baum: offensichtlich, denn G ist zshgd., und wir fügen Kanten nur dann nicht hinzu, wenn sie einen Kreis schließen würden.
- Minimalität: mittels obigem Lemma (MST Eigenschaft):

MST Eigenschaft

Lemma:
Sei G=(V,E) zshgd. mit Gewichtsfunktion w : E→R. Seien:
- $S\subset V$
- $T\subset E$ aussichtsreich und *keine* Kante aus T verlässt S
- $e\in E$ eine Kante mit minimalem Gewicht, die S verlässt

Dann ist $T \cup \{e\}$ aussichtsreich.

Beweis ff: Minimalität
Seien $t_1,t_2,\ldots,t_k$ die Kanten, die Kruskal in dieser Reihenfolge zu T hinzufügt. Wir zeigen: $T_i:=\{t_1,t_2,\ldots,t_i\}$ mit $0\le i\le k$ ist aussichtsreich.

Induktion: $i=0$: $T_0=\emptyset$ → o.k.
Sei nun $1\le i\le k$: Ind.Ann.: T_{i-1} ist aussichtsreich.
Sei $t_i=(u,v)$ und $C_u=(U,E_u)$ die Komponente des Graphen $G_{i-1}=(V,T_{i-1})$ die u enthält.
- t_i verläßt U (da $T_{i-1} \cup \{t_i\}$ kreisfrei bleibt)
- keine Kante aus T_{i-1} verläßt U
- keine billigere Kante als t_i verläßt U
- t_i ist eine Kante minimalen Gewichts, die U verläßt

→ $T_i:= T_{i-1} \cup \{t_i\}$ ist aussichtsreich.

Realisierung von Kruskal

Idee:
- Sortiere die Kanten nach aufsteigendem Gewicht: $w(e_1)\le w(e_2)\le\ldots\le w(e_m)$
- $T:=\emptyset$
- Für $i:=1,\ldots,m$:
- Falls $T \cup \{e_i\}$ kreisfrei, dann: $T:=T \cup \{e_i\}$

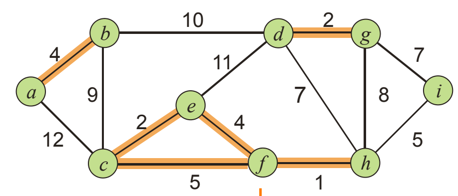
Problem: Teste, ob $T \cup \{e_i\}$ kreisfrei

Teste, ob $T \cup \{e_i\}$ kreisfrei

Idee:
- Aufruf der Funktion ISACYCLIC(V, $T \cup \{e_i\}$)
- Gesamtlaufzeit für alle Kreis-Tests: $O(|V|\,|E|)$, da $|E|$ Aufrufe nötig sind, und T pro Aufruf bis zu $|V|$-1 Kanten besitzen kann.

Besser: Datenstruktur Union-Find
Idee: „Lasse Wälder wachsen" bzw. Partitionierung von Mengen
→ fast „Linearzeit" erreichbar

Kruskal mit Mengenpartitionierung



{a},{b},{c},{d},{e},{f},{g},{h},{i}	(f,h)
{a},{b},{c},{d},{e},{f,h},{g},{i}	(c,e)
{a},{b},{c,e},{d},{f,h},{g},{i}	(d,g)
{a},{b},{c,e},{d,g},{f,h},{i}	(e,f)
{a},{b},{c,e,f,h},{d,g},{i}	(a,b)
{a,b},{c,e,f,h},{d,g},{i}	(c,f)

Partitionierung von Mengen

Idee:
- Zu Beginn ist jeder Knoten in einer eigenen Menge
- Wird eine Kante $e=(u,v)$ zu T hinzugenommen, dann werden die beiden Mengen, in denen u und v jeweils liegen, vereinigt.
- Test auf Kreisfreiheit von $T \cup \{(u,v)\}$: Kreis entsteht genau dann wenn u und v in der gleichen Menge liegen

Operationen zur Partitionierung

- Erzeuge eine einelementige Menge der Partition: MAKESET
- Finde die Partitionsmenge, in der sich ein gegebenes Element befindet: FIND
- Vereinige zwei Partitionsmengen: UNION

→ hierfür: ADT UNION-FIND

6.5.3 Der ADT UNION-FIND

- Zweck: Verwaltung einer Partition einer endlichen Menge, deren Elemente aus einem endlichen Universum U sind.
- Wertebereich: Familie $P=\{S_1,S_2,\ldots,S_k\}$ paarweiser disjunkter Teilmengen einer endlichen Menge U. Für jede Teilmenge gibt es einen eindeutigen Repräsentanten $s_i \in S_i$.
- Sei $P=\{S_1,S_2,\ldots,S_k\}$ mit Repräsentanten s_i die Instanz vor Anwendung der Operation.

Operationen des ADT UNION-FIND

MAKESET(U x)
- Voraussetzung: $x \notin S_1 \cup S_2 \cup \ldots \cup S_k$
- Erzeugt eine neue Menge $S_{k+1}:=\{x\}$ mit Repräsentant $s_{k+1}:=x$ und fügt sie zu P hinzu.

FIND(U x): U
- Voraussetzung: x ist in einer der Mengen $S_1,S_2,\ldots,S_k$ enthalten
- Gibt den Repräsentanten der Menge zurück, die x enthält.

Operationen des ADT UNION-FIND

UNION(U x, U y)
- Voraussetzung: x und y sind jeweils Repräsentant einer der Mengen $S_1,S_2,\ldots,S_k$
- Sei x Repräsentant von S_x, und y Repräsentant von S_y. Dann werden S_x und S_y in P durch $S' := S_x \cup S_y$ ersetzt, d.h. die Instanz P' nach der Operation ist

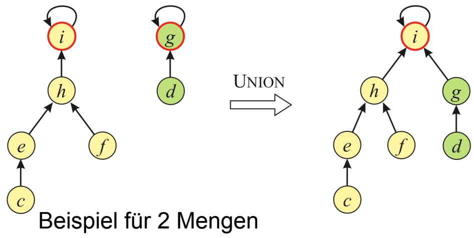
$$P' := P \setminus \{S_x, S_y\} \cup \{S'\}.$$

- Als Repräsentant für S' wird ein beliebiges Element in S' gewählt.

Realisierung des ADT UNION-FIND

Idee: Realisierung durch gewurzelte Bäume:
- Jede Menge S_k wird durch einen gewurzelten Baum dargestellt.
- Die Knoten des Baumes sind die Elemente der Menge.
- Die Wurzel des Baumes ist der Repräsentant der Menge.
- In der Implementierung hat jeder Knoten v nur einen Verweis auf seinen Elter (keine Kindzeiger)

Realisierung durch gewurzelte Bäume



Beispiel für 2 Mengen

- Implementierung:
- Jeder Knoten v hat Verweis auf seinen Elter: $\pi(v)$
- Die Wurzel zeigt auf sich selbst.

Implementierung von Union-Find

```
procedure MAKESET(U x) {
   π[x]:=x
}
procedure UNION(U x, U y) {
   π[x]:=y
}
function FIND(U x):U {
   while π[x]≠x do { x := π[x] }
   return x
}
```


Laufzeitanalyse des ADT

- MAKESET(x): O(1)
- UNION(x,y): O(1)
- FIND(x): $O(h(T_x))$, wobei T_x der Baum ist, der x enthält.

- Gesamtlaufzeit von n MAKESET Operationen und n-1 UNION und 2m FIND Operationen (für Kruskal): O(n+mn), da die Höhe eines Baumes $\Theta(n)$ sein kann.

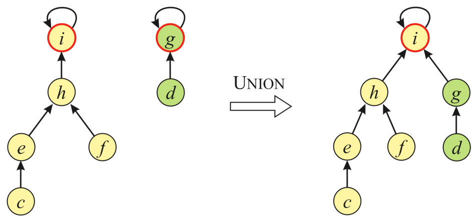
hier: n=|V|, m=|E|

jetzt: Verbesserung der Laufzeiten

Verbesserung 1: Gewichtete Vereinigungsregel

Idee:
Bei UNION: Hänge den Baum mit der kleineren Höhe an den Baum mit der größeren Höhe.

Realisierung durch gewurzelte Bäume



- Wenn $h(T_x) > h(T_y)$, dann ist die neue Höhe $h(T_x)$ → sie nimmt nicht zu
- Wenn beide Bäume gleiche Höhe haben, dann ist die neue Höhe: $h(T_x)+1$

Verbesserung 1: Gewichtete Vereinigungsregel

Lemma: UNION mit gewichteter Vereinigungsregel liefert immer Bäume, deren Höhe h(T) durch log s(T) nach oben beschränkt ist.

Anzahl der Knoten im Baum

Beweis: Wir zeigen, dass jeder erzeugte Baum mit Höhe d mindestens 2^d Knoten besitzt. Induktion:

d=0: Baum besitzt 1 Knoten: $1 \geqq 2^0$

d≧1: Ind.-Ann.: Beh. gilt für alle Bäume mit d´<d

Beweis Lemma ff. z.z.: Jeder erzeugte Baum mit Höhe d hat ≧2^d Knoten

Sei T Baum mit Höhe d mit minimaler Knotenanzahl.

- T entstand dadurch, dass bei UNION der Baum T_1 an T_2 gehängt wurde.
- Wir wissen also: $h(T_1) \le h(T_2)$ und
- $h(T_2) \le d-1$, denn sonst hätte T_2 Höhe d aber weniger Knoten als T, was nach Wahl von T nicht möglich ist.
- $h(T)$ ist also höher als $h(T_1)$ und $h(T_2)$, deswegen müssen beide gleiche Höhe gehabt haben → $h(T_1)=h(T_2)=d-1$
- aus Ind.Vorr. folgt: $s(T_1) \geqq 2^{d-1}$ und $s(T_2) \geqq 2^{d-1}$
- daraus folgt: $s(T) \geqq 2^{d-1} + 2^{d-1} = 2^d$

Analyse der gewichteten Vereinigungsregel

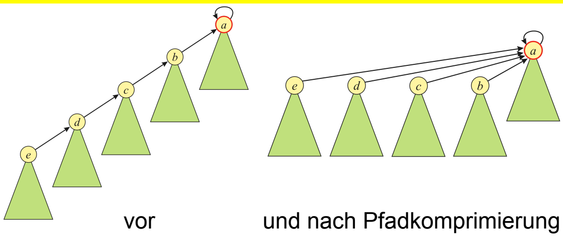
Lemma: Worst-Case Laufzeit einer FIND Operation ist durch $O(\log n)$ beschränkt.

Die Gesamtlaufzeit von n MAKESET Operationen und n-1 UNION und 2m FIND Operationen (für Kruskal) ist also: $O(n+m \log n)$.

Bemerkung: Statt als Rang einer Wurzel die Höhe des Baumes zu benutzen, kann man alternativ auch die Anzahl der Knoten im Baum verwenden → auch Höhe $\log s(T)$.

Verbesserung 2: Pfadkomprimierung

Idee: Bei einem Aufruf von FIND(x) werden alle Knoten auf dem Pfad von x bis zur Wurzel zu direkten Kindern der Wurzel gemacht.



vor und nach Pfadkomprimierung

Analyse der Pfadkomprimierung

Theorem: Wird die gewichtete Vereinigungsregel und Pfadkomprimierung benutzt, so kosten n-1 UNION und 2m FIND Operationen im Worst Case Zeit $O((n+m)\ \alpha(n))$.

Dabei ist $\alpha(n)$ eine extrem langsam wachsende Funktion.

Tarjan machte die ursprüngliche Analyse mit der Inversen der Ackermann-Funktion, die sehr ähnlich zu $\alpha(n)$ ist.

Definition

Für k≧0 und j≧1 definieren wir:

$$A_k(j) := \begin{cases} j+1 & \text{if } k=0 \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1 \end{cases}$$

(j+1)-fache Anwendung der Funktion

Es gilt: $A_2(j)=2^{j+1}(j+1)-1$

$A_0(1) = 1+1=2$

$A_1(1) = A_0^{(2)}(1)=A_0(A_0(1)) = A_0(2) = 2+1=3$

$A_2(1) = A_1^{(2)}(1)=A_1(3) = A_0^{(4)}(3) = A_0^{(3)}(A_0(3)) =7$

$A_3(1) = A_2^{(2)}(1)=A_2(7) = 2^8 \cdot 8-1=2047$

$A_4(1) = A_3^{(2)}(1)=A_3(2047) \gg A_2(2047) > 16^{512} \gg 10^{80}$

Definition

Die Inverse der Funktion A_k ist definiert als

$\alpha(n) := \min \{ k \mid A_k(1) \geqq n\}$

$$\alpha(n) = \begin{cases} 0 & \text{für } 0 \leq n \leq 2 \\ 1 & \text{für } n = 3 \\ 2 & \text{für } 4 \leq n \leq 7 \\ 3 & \text{für } 8 \leq n \leq 2047 \\ 4 & \text{für } 2048 \leq n \leq A_4(1) \end{cases}$$

Wir können also annehmen, dass $\alpha(n) \le 4$ ist für alle praktisch relevanten Werte von n.

Implementierung von UnionFind

```
procedure MAKESET(U x) {
   π[x]:=x
   rank[x]:=0
}
procedure UNION(U x, U y) {
   if rank[x]>rank[y] then
      π[y]:=x
   else π[x]:=y
   if rank[x]==rank[y] then rank[y]:=rank[y]+1
}
```


Implementierung von UnionFind

```
function FIND(U x):U {
   if π[x]≠x then
      π[x]:=FIND(π[x])
   return π[x]
}
```


Realisierung von Kruskal mit UNION-FIND

Idee:

- Sortiere die Kanten nach aufsteigendem Gewicht: $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$
- $T := \emptyset$
- Für $i:=1,\ldots,m$:
- Falls $T \cup \{e_i\}$ kreisfrei, dann: $T := T \cup \{e_i\}$

Problem: Teste, ob $T \cup \{e_i\}$ kreisfrei

Realisierung von Kruskal mit Union-Find

```
Sortiere Kanten, so dass w(e_1)≤w(e_2)≤…≤w(e_m)
T:=∅; i:=1
while |T|<|V|-1 do          // let e_i=(u,v)
   x:=P.FIND(u)
   y:=P.FIND(v)
   if x≠y then {
      T:=T ∪ {e_i}
      P.UNION(x,y)
} // while
i:=i+1
}
```


Analyse von Kruskal mit Union-Find

- Sortieren der Kanten: $O(m \log m)$
- Initialisieren von MAKESET: $O(n)$
- while-Schleife: wird im schlechtesten Fall für alle Kanten durchlaufen: m Mal
- Dabei insgesamt $2m$ FIND-Operationen und $n-1$ UNION-Operationen
- Dies ist also: $O((n+m)\alpha(n))$ und falls G zusammenhängend ist, gilt $n=O(m)$ und damit $O(m\ \alpha(n))$
- **Gesamtlaufzeit** wird von Sortieren dominiert: $O(m \log m)$

Analyse von Kruskal mit Union-Find

Theorem: Der Algorithmus von Kruskal berechnet einen minimalen Spannbaum eines ungerichteten, zusammenhängenden Graphen $G=(V,E)$ in Zeit $O(|E| \log |E|)$.

jetzt: PRIM Algorithmus für MST