
Algorithmen & Komplexität

Angelika Steger
Institut für Theoretische Informatik

steger@inf.ethz.ch

Kürzeste Pfade Problem

Gegeben

Netzwerk: Graph $G = (V, E)$, Gewichtsfunktion $w: E \rightarrow \mathbf{N}$

Zwei Knoten: s, t

Kantenzug/Weg

Folge von Knoten $v_0, v_1, v_2, \dots, v_{n-1}, v_n$ wobei

$\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}$ Kanten des Graphen sind.

Pfad

Ein Kantenzug, bei dem jeder Knoten **höchstens einmal** vorkommt.

Gewicht eines Pfades

Summe der Kantengewichte des Pfades

Gesucht

Kürzester Pfad (d.h. mit geringstem Gewicht) zwischen s und t .

Idee des Algorithmus von Dijkstra

Datenstrukturen

W	(Menge der Knoten v für die kürzester s - v -Pfad bekannt)
$\rho[v]$	(aktuell bekannte Länge eines kürzesten Pfades)
$\text{pred}[v]$	(Pointer zu einem Vorgänger)

Invarianten

1. Für alle Knoten v aus W ist der Wert der Markierung $\rho[v]$ gleich der Länge eines kürzesten s - v Pfades; und es gibt einen solchen Pfad, der mit der Kante $\{\text{pred}[v], v\}$ endet.
2. Für alle Knoten v aus $V - W$ mit Markierung $\rho[v]$ kleiner ∞ ist diese gleich einem kürzesten Pfad in dem Netzwerk auf den Knoten $W \cup \{v\}$; und es gibt einen solchen Pfad, der mit der Kante $\{\text{pred}[v], v\}$ endet.

Algorithmus von Dijkstra

Algorithmus 2.5 Algorithmus von Dijkstra

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$

Ausgabe: Ein kürzester s - t -Pfad in N

{ *Initialisierung* }

$W := \emptyset$; $\rho[s] := 0$; $\text{pred}[s] := \text{nil}$;

for all $v \in V \setminus \{s\}$ **do** $\rho[v] := \infty$;

{ *Traversieren und Markieren* }

while $t \notin W$ **do**

 { *Traversieren* }

 Finde ein $x_0 \in V \setminus W$ mit $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\}$;

$W := W \cup \{x_0\}$;

 { *Markieren* }

for all $v \in \Gamma(x_0) \cap (V \setminus W)$ mit $\rho[v] > \rho[x_0] + \ell(x_0, v)$ **do**

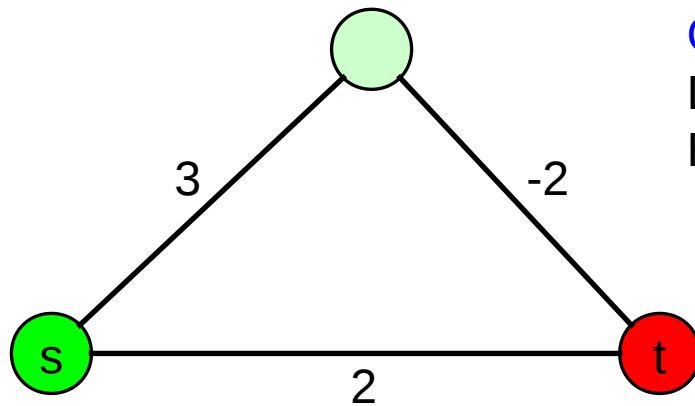
$\rho[v] := \rho[x_0] + \ell(x_0, v)$; $\text{pred}[v] := x_0$;

Algorithmus von Dijkstra

Satz

Für **nicht-negative** Gewichtsfunktionen berechnet der Algorithmus von Dijkstra zu einem Startknoten s und einen Zielknoten t einen kürzesten Pfad von s zu t . Die Laufzeit des Algorithmus ist $O(n^2)$.

Die Voraussetzung nicht-negativer Gewichte ist wesentlich.



Gewicht eines kürzesten s - t -Pfad

Dijkstra: 2

Korrekt: 1

Initialisierung: $O(n)$

while-Schleife

Speicherung der Markierungen in einem [Array](#).

$O(n)$ mal wird ein Element mit kleinster Markierung gesucht.

Jede solche Suche kostet $O(n)$ Vergleichsoperationen.

Ändern der Markierungen der Nachbarn kostet $O(n)$ Speicherzugriffe.

Insgesamt: $O(n^2)$ Operationen.

Ausblick

Verwendung von [Fibonacci Heaps](#) Laufzeit $O(n \log n + m)$ möglich.

Satz

Für **nicht-negative** Gewichtsfunktionen berechnet der Algorithmus von Dijkstra zu einem Startknoten s **für alle Knoten** $v \in V$ einen kürzesten Pfad von s zu v .

Wie müssen wir den Algorithmus abändern?
eigentlich fast gar nicht ...

Algorithmus von Dijkstra

Algorithmus 2.5 Algorithmus von Dijkstra

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$

Ausgabe: Ein kürzester s - t -Pfad in N

{ *Initialisierung* }

$W := \emptyset$; $\rho[s] := 0$; $\text{pred}[s] := \text{nil}$;

for all $v \in V \setminus \{s\}$ **do** $\rho[v] := \infty$;

{ *Traversieren und Markieren* }

while $W \neq V$ **do**

 { *Traversieren* }

 Finde ein $x_0 \in V \setminus W$ mit $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\}$;

$W := W \cup \{x_0\}$;

 { *Markieren* }

for all $v \in \Gamma(x_0) \cap (V \setminus W)$ mit $\rho[v] > \rho[x_0] + \ell(x_0, v)$ **do**

$\rho[v] := \rho[x_0] + \ell(x_0, v)$; $\text{pred}[v] := x_0$;

Satz

Für **nicht-negative** Gewichtsfunktionen berechnet der Algorithmus von Dijkstra zu einem Startknoten s **für alle Knoten** $v \in V$ einen kürzesten Pfad von s zu v . Die Laufzeit des Algorithmus ist $O(n^2)$.

Ausblick

Verwendung von **Fibonacci Heaps** Laufzeit $O(n \log n + m)$ möglich.

Single-Source vs. All-Pairs Shortest Path

Gegeben

Netzwerk $N = (V, E, w)$

w nicht-negativ

Single-Source Shortest Path Problem

Berechne alle kürzesten Pfade ab **einem** Startknoten s

Algorithmus von Dijkstra

Laufzeit: $O(n^2)$ [bzw. $O(n \log n + m)$... später!]

All-Pairs Shortest Path Problem

Berechne die kürzesten Pfade zwischen **allen** Knotenpaaren

Algorithmus von Floyd-Warshall

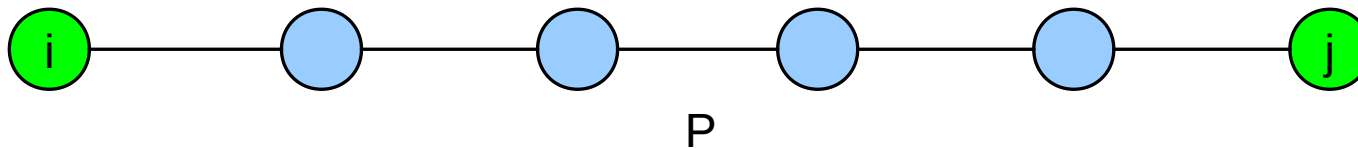
Laufzeit: $O(n^3)$

Idee des Algorithmus von Floyd-Warshall

Knotenmenge $V = \{1, 2, \dots, n\}$

P sei beliebiger Pfad zwischen den Knoten i und j

Endknoten  Innere Knoten 



Invariante

Berechne eine Folge F_k von Matrizen für $k = 0, 1, 2, \dots, n$

$F_k[i, j]$ speichert die Länge eines kürzesten Pfades zwischen i und j der nur innere Knoten aus der Menge $\{1, 2, \dots, k\}$ hat.

Algorithmus von Floyd-Warshall

Algorithmus 2.6 Algorithmus von Floyd-Warshall

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und $V = \{1, \dots, n\}$

Ausgabe: Die Längen aller kürzesten i - j -Pfade in N , für alle Paare $i, j \in V$

{ *Initialisierung* }

$$\forall i, j \in V : F_0[i, j] := \begin{cases} \ell(\{i, j\}) & \text{falls } \{i, j\} \in E \\ 0 & \text{falls } i = j \\ \infty & \text{sonst} \end{cases} .$$

{ *Rekursion* }

for $k = 1$ **to** n **do**

$$\forall i, j \in V : F_k[i, j] := \min\{F_{k-1}[i, j], F_{k-1}[i, k] + F_{k-1}[k, j]\};$$

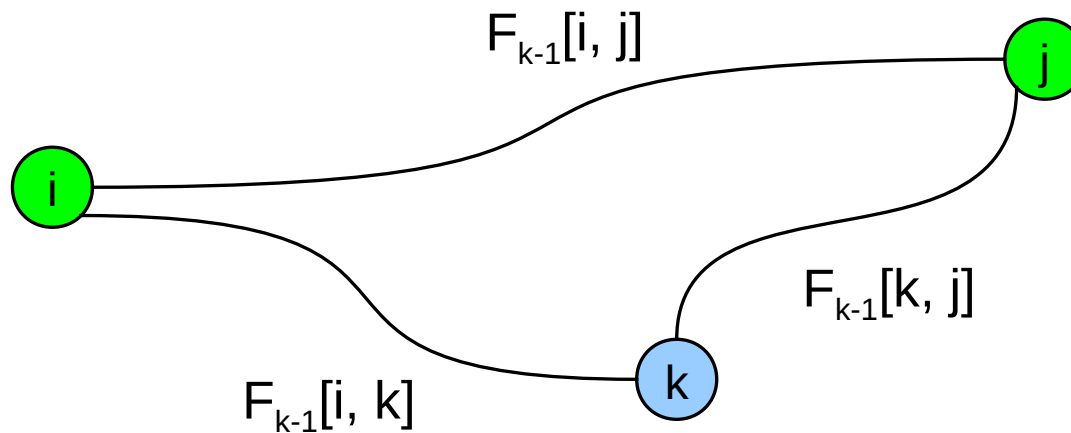
{ *Ausgabe* }

$F_n[i, j]$ gibt die Länge eines kürzesten i - j -Pfades an.

Induktion nach k

Verankerung: klar

Induktionsschritt: $k - 1$ nach k



- Betrachte neuen Zwischenknoten k .
- Entweder ist der „alte Pfad“ mit Länge $F_{k-1}[i, j]$ kürzer.
- Oder der „neue Pfad“ über Zwischenknoten k mit Länge $F_{k-1}[i, k] + F_{k-1}[k, j]$ ist kürzer.

Satz:

Der Algorithmus von Floyd-Warshall berechnet die kürzesten Pfadlängen aller Knotenpaare in Netzwerken mit nicht-negativen Kantengewichten in Zeit $O(n^3)$.

Korrektheit: s.o.

Laufzeit:

Initialisierung: $O(n^2)$

for-Schleife:

läuft von $k=1$ bis n

Für jedes k sind n^2 Knotenpaare zu überprüfen

Insgesamt $O(n^3)$ Operationen

Übungsaufgabe: Modifiziere Alg. so, dass statt Pfadlängen die kürzesten Pfade berechnet werden.

Graphen mit negativen Kantengewichten

Was passiert wenn Kantengewichte auch **negativ** sind?

D.h.: Gewichtsfunktion. $w : E \rightarrow \mathbf{Z}$ (statt $w : E \rightarrow \mathbf{N}$)

Wenn es eine negative Kante gibt, so ist die Länge eines **kürzesten Weges** $-\infty$.

[Begr.: Durchlaufe negative Kante „unendlich oft“ hin und zurück ...]

Das Problem einen **kürzesten Pfad** zu finden ist „schwierig“.

[vgl. Kapitel 5 der Vorlesung]

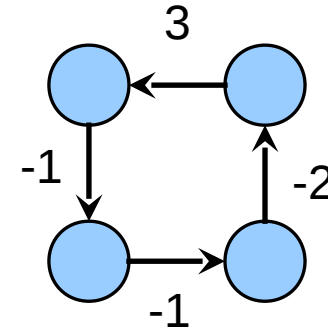
Gerichtete Graphen

Gegeben

Gerichteter Graph: $D = (V, A)$

Kanten haben Richtungen: 

Allgemeine Gewichtsfunktion. $w : A \rightarrow \mathbb{Z}$



Problem

Wenn der Graph einen negativen gerichteten Kreis enthält, kann dieser beliebig oft durchlaufen werden.

Ziel

Entscheide, ob es einen negativen gerichteten Kreis gibt;
falls nicht: bestimme alle kürzesten gerichteten Pfade.

Beachte: Wenn es keinen negativen gerichteten Kreis gibt, so enthält ein kürzester gerichteter Weg immer einen kürzesten gerichteten Pfad!

Kürzester Pfad vs. kürzester Weg

Betrachte Algorithmus von Floyd-Warshall:

Funktioniert auch für gerichtete Graphen (nach leichter Abänderung

der Initialisierung):

- In $F_k[i, j]$ ist nur die Länge eines kürzesten Pfades gespeichert, nicht aber welche Knoten dieser Pfad enthält.
- Bei der Rekursion

$$F_k[i, j] = \min \{ F_{k-1}[i, j], F_{k-1}[i, k] + F_{k-1}[k, j] \}$$

kann daher nicht a priori sicher gestellt werden, dass Knoten nicht doppelt vorkommen.

ABER: kein gerichteter Kreis => können Weg zu Pfad abkürzen

Modifizierter Floyd-Warshall

Algorithmus 2.7 Modifizierter Algorithmus von Floyd-Warshall

Eingabe: Gerichtetes Netzwerk $N = (V, A, \ell)$ mit $V = \{1, \dots, n\}$.

Ausgabe: Die Längen aller kürzesten i - j Pfade in N , für alle Paare $i, j \in V$, oder ein Beweis dafür, dass das Netzwerk einen negativen gerichteten Kreis enthält.

{ *Initialisierung* }

$$\forall i, j \in V : F_0[i, j] := \begin{cases} \ell((i, j)) & \text{falls } (i, j) \in A \\ 0 & \text{falls } i = j \\ \infty & \text{sonst} \end{cases} .$$

{ *Rekursion* }

for $k = 1$ **to** n **do**

$$\forall i, j \in V : F_k[i, j] := \min\{F_{k-1}[i, j], F_{k-1}[i, k] + F_{k-1}[k, j]\};$$

{ *Ausgabe* }

if $(\exists i \in V \text{ mit } F_n[i, i] < 0)$ **then**

return "Graph enthält einen negativen Kreis"

else

$F_n[i, j]$ gibt die Länge eines kürzesten i - j -Pfades an.

Satz:

Für gerichtete Netzwerke $N = (V, A, w)$ mit beliebiger Gewichtsfunktion w gilt: der modifizierte Floyd-Warshall Algorithmus berechnet in Zeit $O(n^3)$ Werte $F_n[i, j]$ mit:

- Enthält N einen negativen gerichteten Kreis, so gilt $F_n[i, i] < 0$ für mindestens ein i .
- Ansonsten bezeichnet $F_n[i, j]$ die Länge eines kürzesten i - j -Pfades.

Beweisidee:

(1) kein negativer Kreis: Zeige durch Induktion über k :

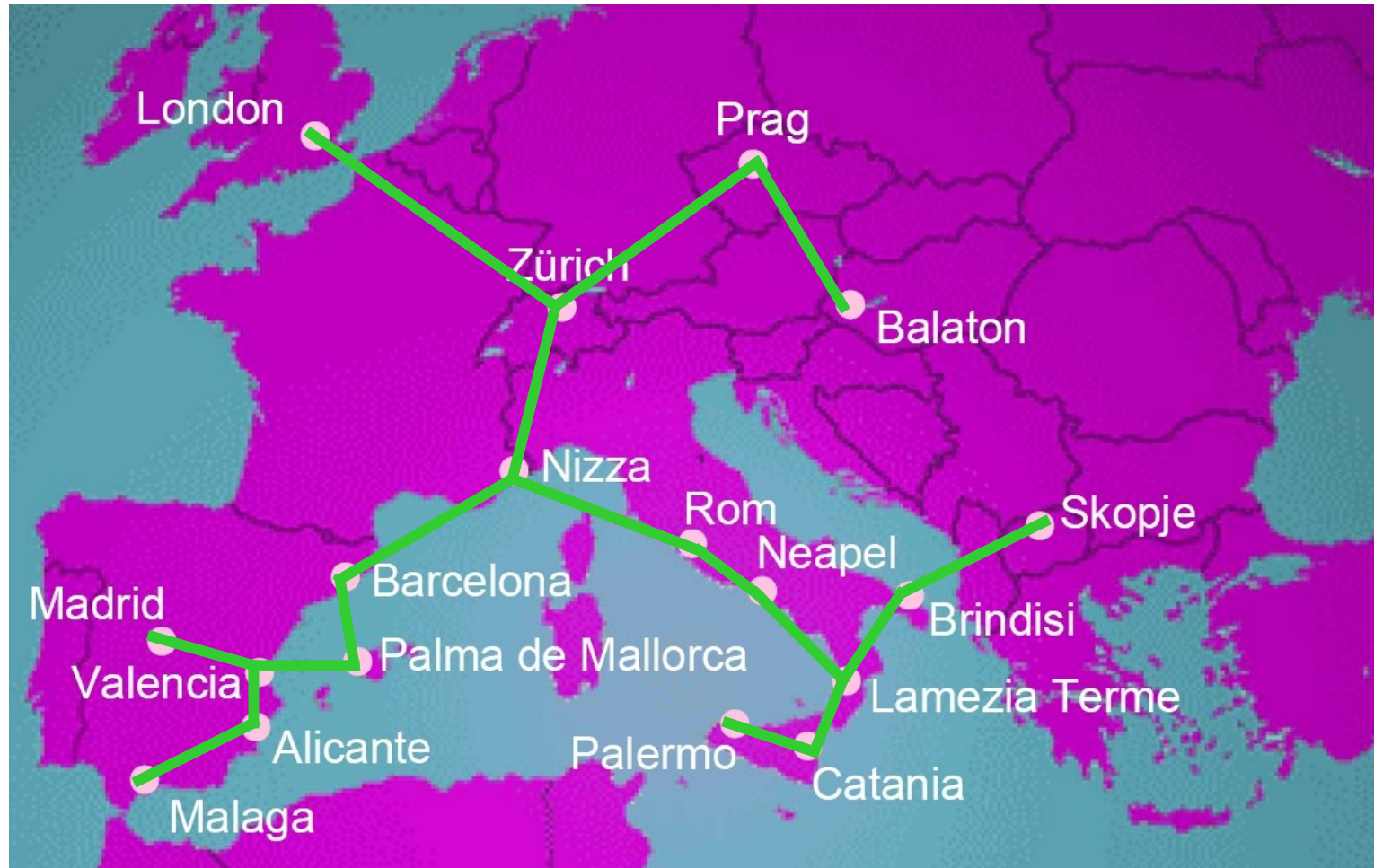
$F_k[i, j]$ = Länge eines kürzesten Pfades von i nach j , dessen
innere Knoten alle in der Menge $\{1, \dots, k\}$ enthalten sind

(2) es gibt negativer Kreis: Betrachte negativen Kreis mit
grösstem Knoten u ...

Kapitel 2.5:

Minimale Spannbäume

Minimaler Spannbaum



Problemstellung

Gegeben

Graph $G = (V, E)$, Gewichtsfunktion $w : E \rightarrow \mathbf{R}$

Spannbaum

$T = (V, E_T)$ Baum mit $E_T \subseteq E$

Gewicht eines Spannbaums

Summe der Kantengewichte des Baumes: $w(T) := \sum_{e \in E_T} w(e)$

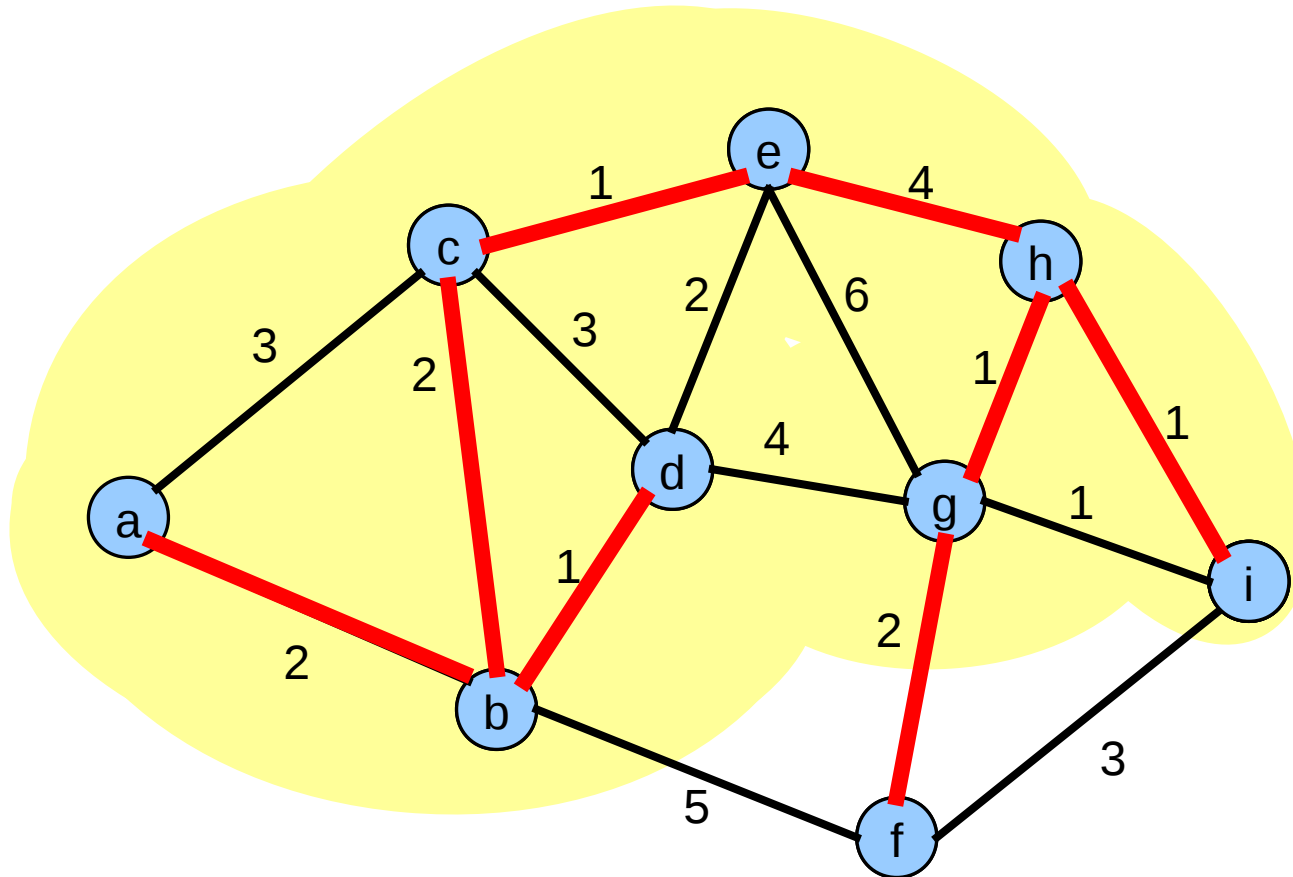
Minimaler Spannbaum

$T = (V, E_T)$ mit $E_T \subseteq E$ mit $w(T) = \min \{ w(T') : T' \text{ Spannbaum} \}$

Gesucht

Minimaler Spannbaum

Beispiel: Algorithmus von Prim



W

Algorithmus von ~~Dijkstra~~ **Prim**

Algorithmus 2.5 Algorithmus von ~~Dijkstra~~ **Prim**

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$

Ausgabe: Ein ~~kürzester $s-t$ Pfad in N~~ **minimaler Spannbaum**

{ *Initialisierung* } **Wähle $s \in V$ beliebig**

$W := \emptyset$; $\rho[s] := 0$; $\text{pred}[s] := \text{nil}$;

for all $v \in V \setminus \{s\}$ **do** $\rho[v] := \infty$;

{ *Traversieren und Markieren* }

while ~~$t \notin W$~~ **do** **$W \neq V$**

 { *Traversieren* }

 Finde ein $x_0 \in V \setminus W$ mit $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\}$;

$W := W \cup \{x_0\}$;

 { *Markieren* }

for all $v \in \Gamma(x_0) \cap (V \setminus W)$ mit $\rho[v] > \rho[x_0] + \ell(x_0, v)$ **do**

$\rho[v] := \rho[x_0] + \ell(x_0, v)$; $\text{pred}[v] := x_0$;

Algorithmus von Prim

Satz

Der Algorithmus von Prim berechnet für alle Gewichtsfunktionen einen minimalen Spannbaum und benötigt höchstens $O(n^2)$ Schritte.

Korrektheit:

Wir zeigen folgende Invarianten:

(i) Die Kanten in F bilden eine **Spannbaum** für die Knoten in W .

(ii) Für alle $v \in V \setminus W$ gilt:

$$\sigma[v] = \min\{ w(\{x,v\}) : x \in W \text{ und } \{x,v\} \in E \},$$

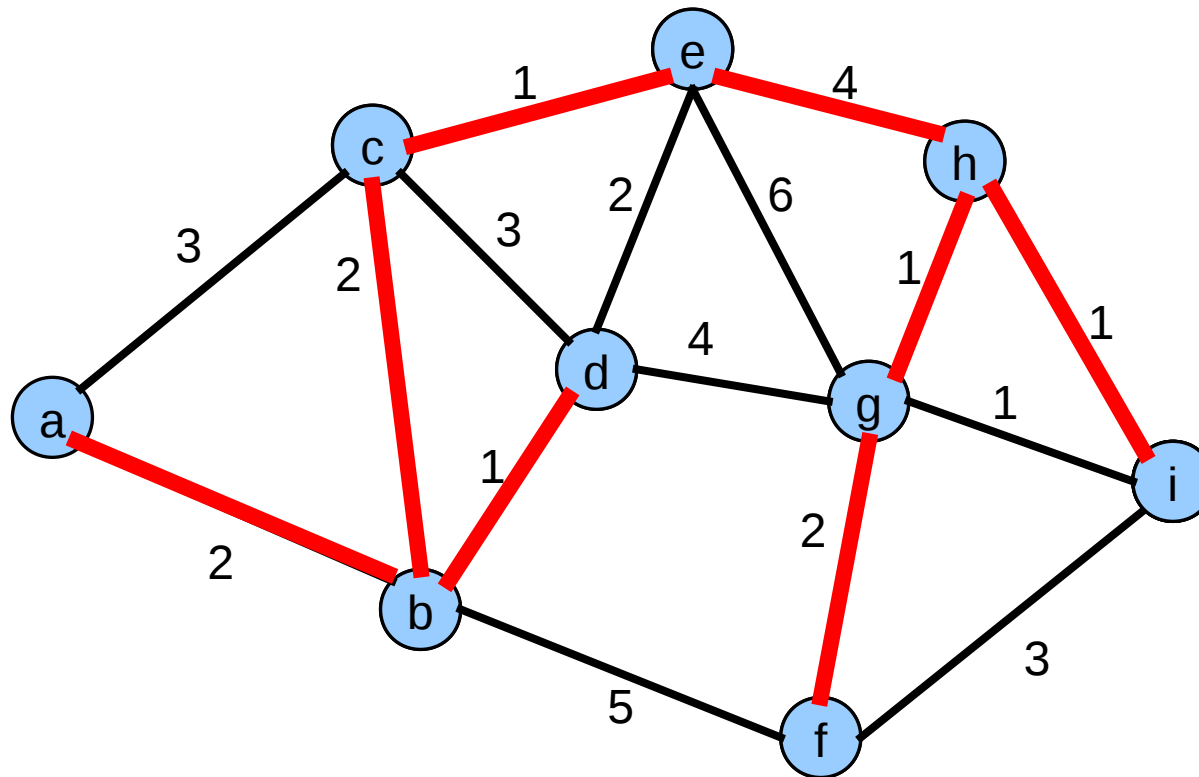
und

$$\sigma[v] = w(\{\text{pred}[v],v\}).$$

[D.h. $\sigma[v]$ ist die Länge der kürzesten Kante, die v mit einem Knoten in W verbindet, und $\{\text{pred}[v],v\}$ ist so eine Kante.]

(iii) Es gibt einen minimalen Spannbaum $T^*=(V,F^*)$, so dass $F \subseteq F^*$.

Beispiel: Algorithmus von Kruskal



Algorithmus von Kruskal

Algorithmus 2.9 Algorithmus von Kruskal

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$

Ausgabe: Ein minimaler Spannbaum $T = (V, F)$ in N

{ *Initialisierung* }

Sortiere die Kanten, so dass $\ell(e_1) \leq \dots \leq \ell(e_m)$.

Setze $F := \emptyset$.

{ *Aufbau des Baums* }

for $i := 1$ **to** m **do**

if $(V, F \cup \{e_i\})$ ist kreisfrei **then** $F := F \cup \{e_i\}$.

Algorithmus von Kruskal

Satz

Der Algorithmus von Kruskal berechnet für alle Gewichtsfunktionen einen minimalen Spannbaum und benötigt höchstens $O(nm)$ Schritte.

Korrektheit: später ...

Laufzeit:

Initialisierung:

Sortieren von $m=|E|$ Zahlen: $O(m \log(m)) = O(m \log(n))$

Schleife:

- wird maximal m oft durchlaufen
- pro Durchlauf ein Test auf Kreisfreiheit ... $O(n)$

Verbesserte Implementierung: $O(m \log n)$... später!

Problemstellung

Gegeben

Graph $G = (V, E)$, Gewichtsfunktion $l : E \rightarrow \mathbf{R}$

Spannbaum

$T = (V, E_T)$ Baum mit $E_T \subseteq E$

Gewicht eines Spannbaums

Summe der Kantengewichte des Baumes: $l(T) := \sum_{e \in E_T} l(e)$

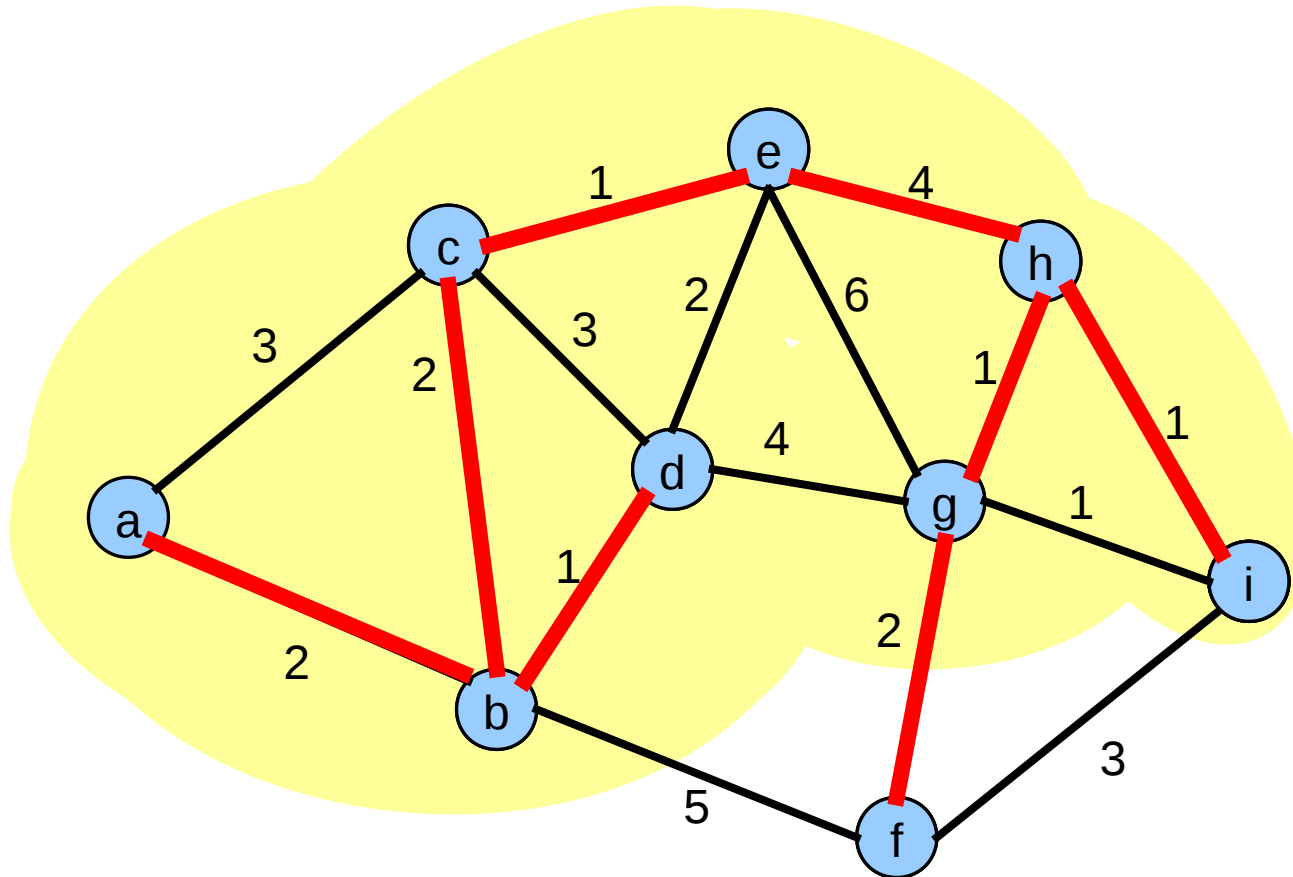
Minimaler Spannbaum

$T = (V, E_T)$ mit $E_T \subseteq E$ mit $l(T) = \min \{l(T') : T' \text{ Spannbaum}\}$

Gesucht

Minimaler Spannbaum

Beispiel: Algorithmus von Prim



W

Algorithmus von ~~Dijkstra~~ **Prim**

Algorithmus 2.5 Algorithmus von ~~Dijkstra~~ **Prim**

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$

Ausgabe: Ein ~~kürzester $s-t$ Pfad in N~~ **minimaler Spannbaum**

{ *Initialisierung* } **Wähle $s \in V$ beliebig**

$W := \emptyset$; $\rho[s] := 0$; $\text{pred}[s] := \text{nil}$;

for all $v \in V \setminus \{s\}$ do $\rho[v] := \infty$; $F := \emptyset$;

{ *Traversieren und Markieren* }

while ~~$t \notin W$~~ do $W \neq V$

{ *Traversieren* }

Finde ein $x_0 \in V \setminus W$ mit $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\}$;

$W := W \cup \{x_0\}$; **$F := F + \{x_0, \text{pred}[x_0]\}$;**

{ *Markieren* }

for all $v \in \Gamma(x_0) \cap (V \setminus W)$ mit $\rho[v] > \rho[x_0] + \ell(x_0, v)$ do

$\rho[v] := \rho[x_0] + \ell(x_0, v)$; $\text{pred}[v] := x_0$;

Beweisidee:

Zeige, dass vor und nach jeder Iteration der while-Schleife gilt:

- es gibt einen minimalen Spannbaum **T** mit **$F \subseteq T$**

- für alle $v \in V \setminus W$ gilt: $\rho[v] = \min\{l(v, w) : w \in W\}$ und $\rho[v] = l(v, \text{pred}[v])$

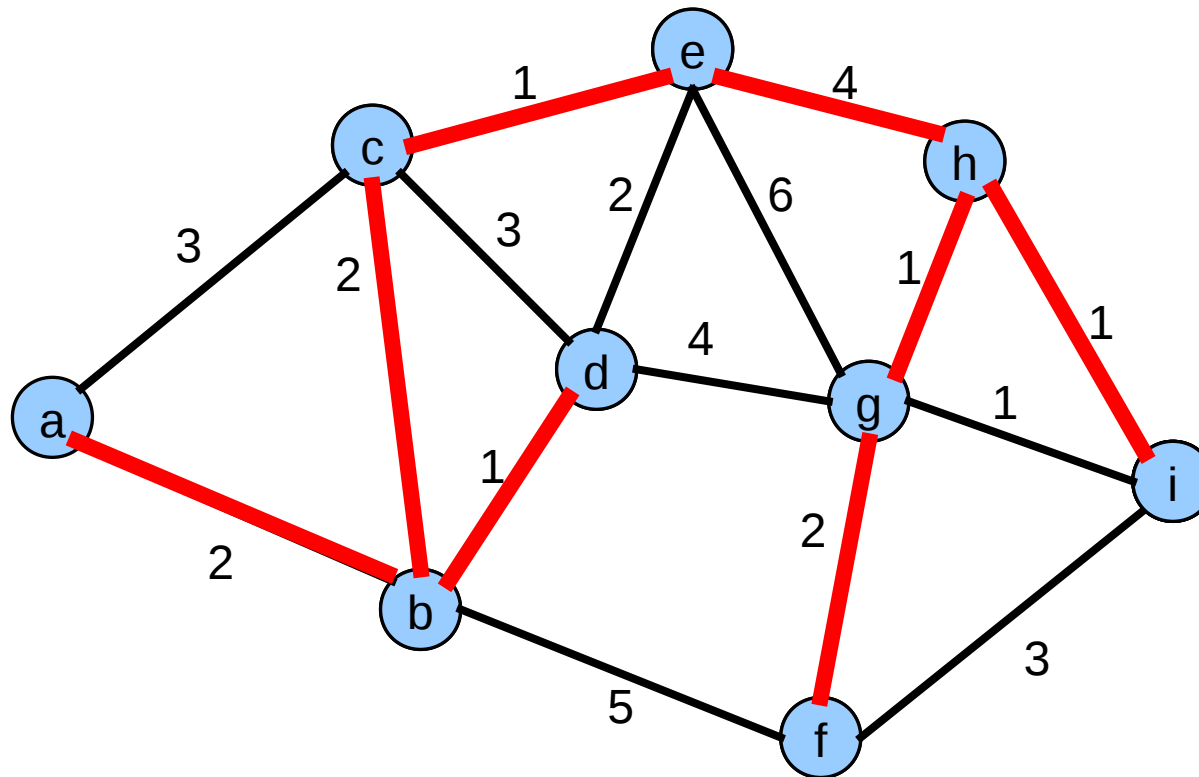
Algorithmus 2.5 Algorithmus von ~~Dijkstra~~ **Prim**

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$ mit $\ell \geq 0$ und Knoten $s, t \in V$

Ausgabe: Ein ~~kürzester $s-t$ Pfad in N~~ **minimaler Spannbaum**

```
{ Initialisierung }   Wähle  $s \in V$  beliebig  
 $W := \emptyset; \rho[s] := 0; \text{pred}[s] := \text{nil};$   
for all  $v \in V \setminus \{s\}$  do  $\rho[v] := \infty; \mathbf{F} := \emptyset;$   
{ Traversieren und Markieren }  
while  $t \notin W$  do  $\mathbf{W} \neq V$   
  { Traversieren }  
  Finde ein  $x_0 \in V \setminus W$  mit  $\rho[x_0] = \min\{\rho[v] \mid v \in V \setminus W\};$   
   $W := W \cup \{x_0\}; \mathbf{F} := \mathbf{F} + \{\mathbf{x}_0, \text{pred}[\mathbf{x}_0]\};$   
  { Markieren }  
  for all  $v \in \Gamma(x_0) \cap (V \setminus W)$  mit  $\rho[v] > \rho[x_0] + \ell(x_0, v)$  do  
     $\rho[v] := \rho[x_0] + \ell(x_0, v); \text{pred}[v] := x_0;$ 
```

Beispiel: Algorithmus von Kruskal



Algorithmus von Kruskal

Algorithmus 2.9 Algorithmus von Kruskal

Eingabe: Ein zusammenhängendes Netzwerk $N = (V, E, \ell)$

Ausgabe: Ein minimaler Spannbaum $T = (V, F)$ in N

{ *Initialisierung* }

Sortiere die Kanten, so dass $\ell(e_1) \leq \dots \leq \ell(e_m)$.

Setze $F := \emptyset$.

{ *Aufbau des Baums* }

for $i := 1$ **to** m **do**

if $(V, F \cup \{e_i\})$ ist kreisfrei **then** $F := F \cup \{e_i\}$.

Algorithmus von Kruskal

Satz

Der Algorithmus von Kruskal berechnet für alle Gewichtsfunktionen einen minimalen Spannbaum und benötigt höchstens $O(nm)$ Schritte.

Korrektheit: später ...

Laufzeit:

Initialisierung:

Sortieren von $m=|E|$ Zahlen: $O(m \log(m)) = O(m \log(n))$

Schleife:

- wird maximal m oft durchlaufen
- pro Durchlauf ein Test auf Kreisfreiheit ... $O(n)$

Verbesserte Implementierung: $O(m \log n)$... später!