Grundidee für die Berechnung von kürzesten Wegen

Für jeden Knoten v wird aufrechterhalten:

- $\delta[v] \ldots$ Länge des kürzesten schon bekannten Wegs von s
- $\pi[v] \ldots$ der Vorgängerknoten auf diesem Weg

Diese Werte werden initialisiert auf:

$\delta[s] = 0$ und $\delta[v] = \infty$ für $v \neq s$
$\pi[x] = \text{NIL}$ für alle x.

Diese Werte werden mit Hilfe von *Kantenrelaxationen* sukzessive verbessert.

Relaxieren einer Kante $[u, v\rangle$: *relax*(u, v)

$$\begin{aligned} &\textbf{if } \delta[v] > \delta[u] + w([u, v\rangle) \textbf{ then} \\ &\qquad \delta[v] = \delta[u] + w([u, v\rangle) \\ &\qquad \pi[v] = u \end{aligned}$$

Man nennt $[u, v\rangle$ *relaxierbar*, wenn $\delta[v] > \delta[u] + w([u, v\rangle)$, wenn also ihre Relaxation eine Auswirkung auf $\delta[]$ und $\pi[]$ hat.

Naiver Kürzeste-Wege-“Algorithmus”

1. Initialisiere $\delta[]$ und $\pi[]$
2. Solange es eine relaxierbare Kante gibt,
   wähle eine und relaxiere sie.

Probleme:

- Ist das überhaupt ein Algorithmus, also, terminiert diese Methode überhaupt?
  Nein, nicht wenn es einen negativen Zyklus gibt, der von s erreichbar ist.
- Wenn diese Methode terminiert, wie lange braucht sie?
  Das hängt stark von der Wahl der zu relaxierenden Kante ab.

Bellman-Ford-Algorithmus

Idee: Finde für steigendes k die kürzesten Wege mit höchstens k Kanten.

Lemma

*Es sei $e_1, \ldots, e_\ell$ ein kürzester Weg von s nach v.
Wenn eine Folge R von Kantenrelaxationen relax(e_1),relax(e_2), $\ldots$,relax(e_ℓ) als Teilfolge enthält (nicht unbedingt zusammenhängend), dann wurde mit R ein kürzester Weg von s nach v berechnet, $\delta[v] = d_s(v)$ und die entsprchenden Vorgänger $\pi[]$ ergeben diesen Weg.*

Bellman-Ford-Algorithmus

Idee: Finde für steigendes k die kürzesten Wege mit höchstens k Kanten.

Wenn es keine negativen Zyklen gibt, besteht jeder kürzeste Weg aus höchstens $n - 1$ Kanten.

Bellman-Ford-Algorithmus:

1. Initialisiere $\delta[]$ und $\pi[]$
2. wiederhole $n - 1$ mal
   relaxiere jede Kante
3. Falls es noch immer eine relaxierbare Kante gibt, dann gibt es einen von s erreichbaren negativen Zyklus. Andernfalls gitl für alle Knoten x, dass $\delta[x] = d_s(x)$ und $\pi[]$ gibt einen kürzesten Wege Baum.

Laufzeit: Insgesamt $O(n \cdot m)$.

Dijkstra Algorithmus

Wenn kein Kantengewicht negativ ist, gibt es eine Methode, die jede Kante nur einmal relaxiert.
Man lässt ein t von 0 nach ∞ wachsen und hält folgendes aufrecht, wobei $V_t = \{x \in V | d_s(x) \leq t\}$:
Invariante

1. Für alle $x \in V$ mit $d_s(x) \leq t$ gilt: $\delta[x] = d_x(s)$ und $\pi[x]$ ist Vorgänger auf kürzesten Pfad von s nach x
2. Für alle $x \in V$ mit $d_s(x) > t$ gilt: $\delta[x]$ ist Länge eine kürzesten Pfades von s nach x mit vorletzten Knoten u in V_t; und, falls so ein Pfad existiert, dann $\pi[x] = u$

Dijkstra Algorithmus

Der Algorithmus verwendet eine Prioritätenschlange für V mit $\delta[x]$ als Schlüssel von x.

Initialisierung: Für alle $x \in V$ setze $\delta[x] = \infty$, $\pi[x] = \text{NIL}$
Für Startknoten s setze $\delta[s] = 0$. Bilde (Min) Prioritätenschlange Q für V.

while Q not emptry **do**
  $u = \textsc{DeleteMin}(Q)$
  for each $v \in \text{Out}(u)$ **do** // relaxiere Kante (u, v)
    if $\delta[v] > \delta[u] + w([u, v\rangle)$ **then**
      $\delta[v] = \delta[u] + w([u, v\rangle)$
      $\textsc{DecreaseKey}(Q, v, \delta[v])$
      $\pi[v] = u$

Dijkstra Algorithmus

Initialisierung: Für alle $x \in V$ setze $\delta[x] = \infty$, $\pi[x] = \text{NIL}$
Für Startknoten s setze $\delta[s] = 0$.
Bilde (Min) Prioritätenschlange Q für V.

while Q not emptry **do**
  u = DELETEMIN(Q)
  for each $v \in \text{Out}(u)$ **do** // relaxiere Kante (u, v)
    if $\delta[v] > \delta[u] + w([u, v\rangle)$ **then**
      $\delta[v] = \delta[u] + w([u, v\rangle)$
      DECREASEKEY($Q, v, \delta[v]$)
      $\pi[v] = u$

Laufzeit

- Initialisierung: $O(n)$
- While Loop: n DELETEMINS
  m Relaxierungen
  höchstens m DECREASEKEYS

Dijkstra Algorithmus

while Q not emptry **do**
  $u = \textsc{DeleteMin}(Q)$
  for each $v \in \text{Out}(u)$ **do** // relaxiere Kante (u, v)
    if $\delta[v] > \delta[u] + w([u, v\rangle)$ **then**
      $\delta[v] = \delta[u] + w([u, v\rangle)$
      $\textsc{DecreaseKey}(Q, v, \delta[v])$
      $\pi[v] = u$

Laufzeit

- Initialisierung: $O(n)$
- While Loop: n DELETEMINS $O(n \log n)$
  m Relaxierungen $O(m)$
  höchstens m DECREASEKEYS $O(m)$

Mit Hollow-Heaps DELETEMIN $O(\log n)$ und DECREASEKEY $O(1)$
Gesamtlaufzeit $O(m + n \log n)$

All-Pair-Shortest-Paths (APSP)

Aufgabe: Finde für jedes Paar u, v von Knoten einen kürzesten Pfad von u nach v

Naiver Ansatz: n mal Single-Source-Shortest-Path Problem lösen

- Keine negative Kanten:
  n mal Dijkstra Laufzeit $O(nm + n^2 \log n)$
- Negative Kanten:
  n mal Bellman-Ford Laufzeit $O(n^2 m + n)$

  Verbesserung von Don Johnson:
  1 mal Bellman-Ford und $n - 1$ mal Dijkstra
  Laufzeit $O(nm + n^2 \log n)$

Johnson's APSP Algorithm

$G = (V, E, w)$ mit Kantengewichten $w(e) < 0$ für manche $e \in E$

Idee: Modifiziere w, sodass

(i) kürzeste Wege erhalten bleiben, und

(ii) alle Kantengewichte nicht-negativ sind

Für $h : V \to \mathbb{R}$ definiere h-modifiziertes Kantengewicht

$$w_h([u, v\rangle) = w([u, v\rangle) + h(u) - h(v)$$

Für jeden Pfad p von x nach y gilt dann
$w_h(p) = w(p) + h(x) - h(y)$.
Das bedeutet, kürzeste Wege bzgl. $w_h()$ sind genau auch kürzeste Wege bzgl. $w()$ und damit gilt (i) für jedes h.

Johnson's APSP Algorithm

Idee: Modifiziere w, sodass

(i) kürzeste Wege erhalten bleiben, und
(ii) alle Kantengewichte nicht-negativ sind

Für $h : V \to \mathbb{R}$ definiere h-modifiziertes Kantengewicht

$$w_h([u, v\rangle) = w([u, v\rangle) + h(u) - h(v)$$

Wir brauchen ein h, sodass für jede Kante e gilt $0 \leq w_h(e)$.

$$0 \leq w_h([u, v\rangle) = w([u, v\rangle) + h(u) - h(v)$$

also $h(v) \leq w([u, v\rangle) + h(u)$.
Das ist genau die Terminierungsbedingung im Bellman-Ford-Algorithms (keine Kante relaxierbar), also gilt (ii) mit $h(x) = \delta[x]$.

Johnson's APSP Algorithm

1. Wähle irgendeinen Knoten $s \in V$ und wende Bellman-Ford auf Startknoten s an.
2. Wenn negativer Zyklus gefunden wurde, dann Abbruch.
3. Verwende das berechnete $\delta[]$ und berechne für jede Kante $[u, v\rangle$ modifiziertes Gewicht

$$w_\delta([u, v\rangle) = w([u, v\rangle) + \delta[u] - \delta[v]$$

4. Für jedes $t \in V \setminus \{s\}$ wende Dijkstra's Algorithmus mit Startpunkt t bygl. w_δ an.

Laufzeit Insgesamt $O(n \cdot m + n^2 \log n)$.

1. $O(n \cdot m)$
2. $O(1)$
3. $O(m)$
4. $(n-1) \cdot O(m + n \log n)$

Floyd-Warshall Algorithmus

(Spezialalgorithmus, hier nur für Distanzen (nicht die Wege))

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

Wir bezeichnen einen Pfad von i nach j als *k-niedrig*, wenn die inneren Knoten (also alle außer den beiden Endpunkten) Name höchstens k haben.
Der kürzeste Weg von i nach j ist auf jeden Fall n-niedrig.

$d_{i,j}^k$ sei Länge eines kürzesten k-niedrigen Weges von Knoten i zu Knoten j

$$d_{i,j}^0 = w([i,j\rangle) \quad \text{oder } \infty\text{, fall } [i,j\rangle \text{ keine Kante}$$

für $k > 0$ gilt induktiv, die Fälle unterscheidend, ob Knoten k auf kürzstem k-niedrigen Weg von i nach j liegt, oder nicht:

$$d_{i,j}^k = \min\{d_{i,k}^{k-1} + d_{k,j}^{k-1}, d_{i,j}^{k-1}\}$$

Das ergibt simplen $O(n^3)$ Algorithmus für Berechnung aller $d_{i,j}^n$.