04.06.13

Idee: Der Algorithmus berechnet die Funktion

$\ell_i(v)$: Länge eines kürzesten Weges zu v mit höchstens i Kanten.

Berechnung aus $\ell_{i-1}(w)$!

Satz 3.6

Algorithmus 3.5 liefert ein korrektes Ergebnis.
Die Laufzeit ist $O(nm)$.

Beweis:

Die Laufzeit ist klar aufgrund der Schleifenstruktur.

Für die Korrektheit betrachte zu jedem Zeitpunkt

$R := \{v \in V(G) \mid \ell(v) < \infty\}$ (erreichte Knoten)

$F := \{(x,y) \in E(G) \mid x = p(y)\}$. (benutzte Kanten)

Wir behaupten:

(a) $\ell(y) \geq \ell(x) + c((x,y))$ für alle $(x,y) \in F$

(b) Wenn F einen Kreis enthält, dann hat C negatives Gesamtgewicht.

(c) Wenn c konservativ ist, dann ist (R,F) eine in s verwurzelte Arboreszenz.

Für (a):

Beachte, dass beim Setzen von $p(y)$ auf x
$\ell(y) = \ell(x) + c((x,y))$ gesetzt wird;
$\ell(x)$ wird nicht mehr erhöht.

Für (b):

Angenommen, wir erzeugen irgendwann einen
Kreis C in F, indem wir $p(y) := x$ setzen.
Vor dieser Einfügung gilt $\ell(y) > \ell(x) + c((x,y))$
und $\ell(w) \geq \ell(v) + c((v,w))$ für alle $(v,w) \in E(C) \setminus \{(x,y)\}$
wegen (a).
Wenn man all diese Ungleichungen für alle $v \in C$
addiert, erhält man

$$\sum_{v \in C} \ell(v) > \sum_{v \in C} \ell(v) + \sum_{v \in C} c((p(v),v)),$$

also $$0 > c(C).$$

Für (c):

Da c konservativ ist, muss wegen (b) F azyklisch sein. Außerdem folgt aus $x \in R \setminus \{s\}$, dass $p(x) \in R$ ist, also ist (R, F) eine in s verwurzelte Arboreszenz.

Daher ist zu jedem Zeitpunkt für $x \in R$

- $\ell(x)$ mindestens die Länge eines s-x-Pfades in (R, F).

Jetzt behaupten wir: Nach k Iterationen ist $\ell(x)$ höchstens die Länge eines kürzesten s-x-Pfades mit höchstens k Kanten.

- Beweis durch Induktion:
  Sei P ein kürzester s-x-Pfad mit höchstens k Kanten, sei (w, x) die letzte Kante in P. Dann muss $P_{[s,w]}$ ein kürzester s-w-Pfad mit höchstens $k-1$ Kanten sein. In der k-ten Iteration wird aber auch die Kante (w, x) untersucht, wonach $\ell(x) \leq \ell(w) + c((w, x)) \leq c(P)$ gilt.

Da kein Pfad mehr als $n-1$ Kanten haben kann, ist der Algorithmus korrekt. □

Bemerkung 3.7

Der Algorithmus 3.6 funktioniert nach dem Prinzip des sogenannten "Dynamic Programming" („dynamischen Programmierens"):

Man berechnet den Wert für „größere" Wege aus den Werten für „kleinere Wege", indem man eine Tabelle aufstellt:

$\ell(i,v)$: Kürzester Weg nach v mit höchstens i Kanten

und induktiv berechnet:

$$\ell(i,v) = \min_{w \in V} \left(\ell(i-1,w) + c(w,v) \right)$$

Dieses Prinzip ist sehr universell einsetzbar!

3.3 Fibonacci Heaps

Ideen:
- Datenstruktur, um dynamische Operationen mit sich ändernden Labels zu unterstützen
- Kritische Operationen:
  - Delete Min
  - Delete
  - Decrease Key

  } $O(\log n)$* ← amortisiert
- Verwende viele „schnelle" Operationen, um Zeit zu sparen, die für gelegentliche „langsame" Operationen aufgebracht werden kann → amortisierte Laufzeit

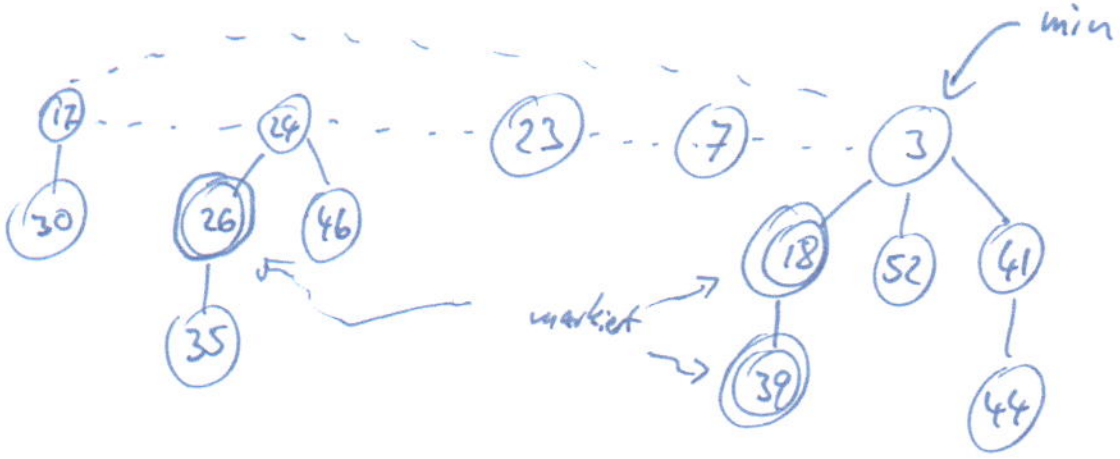
Fredman + Tarjan (1987)

„Once you succeed in writing the program for these complicated algorithms, they usually run extremely fast. The computer doesn't need to understand the algorithm, its task is only to run the programs."

Details: Kapitel 20 in Cormen et al. (p. 476 - 497)

~~Operatio~~

Struktur:



- Vereinigung von Bäumen, in denen die Elemente jeweils nach oben hin kleiner werden

  → Heap

- Minimum steht jeweils oben in der Wurzel jedes Teilbaumes
- ~~Globale~~ Wurzeln min doppelt verkoppelter Liste.
- Globales Minimum ~~markiert~~ per Pointer
- Manche Knoten „markiert".

→ Mehr siehe Folien + Webseite!