

Grafalgoritmer 2

Douglas Wikström

KTH Stockholm

`popup-help@csc.kth.se`

Kortaste avstånd från given nod

Problem. Låt $G = (V, E)$ vara en **riktad viktad graf** och $s \in V$. Hitta för varje v vägen från s till v med lägst totalvikt.

Vi kan lösa detta med Dijkstra.

Kortaste avstånd från given nod

Vad händer om vikterna är negativa?

Kortaste avstånd från given nod (negativa vikter)

Input: En graf G med viktade kanter och en startnod s

Output: Minsta avståndet från en nod s till alla andra noder i G

BELLMANFORD($G, w(\cdot), s$)

- (1) $p(u) \leftarrow \perp$ och $d(u) \leftarrow \infty$ för $u \in V$.
- (2) $d(s) \leftarrow 0$
- (3) **for** $i = 1$ **to** $n - 1$
- (4) **foreach** $(u, v) \in E$
- (5) **if** $d(u) + w(u, v) < d(v)$
- (6) $d(v) \leftarrow d(u) + w(u, v)$
- (7) $p(v) \leftarrow u$
- (8) **return** $(d(\cdot), p(\cdot))$

Att upptäcka negativa cykler

Input: En graf G med viktade kanter och en startnod s , och en Bellman-Ford lösning $(d(\cdot), p(\cdot))$.

Output: *true* eller *false* beroende på om G har en negativ cykel eller inte.

NEG_CYCLES($G, w(\cdot), s, d(\cdot), p(\cdot)$)

- (1) **foreach** $(u, v) \in E$
- (2) **if** $d(u) + w(u, v) < d(v)$
- (3) **return** *true*
- (4) **return** *false*

Varför fungerar algoritmen?

Lemma. Efter i iterationer i for-loopen gäller:

- ▶ Om $d(u) \neq \infty$ så finns en väg av längd $d(u)$ från s till u .
- ▶ Om det finns en väg från s till u med högst i kanter, så är $d(u)$ längden av en kortaste väg från s till u med högst i kanter.

Kortaste avstånd mellan alla nodpar

Problem. Låt $G = (V, E)$ vara en **riktad viktad graf**. Hitta för varje $s, v \in V$ vägen från s till v med lägst totalvikt.

Hur gör vi detta?

Kortaste avstånd mellan alla nodpar

- ▶ Upprepad Dijkstra: $O(nm \log n)$ (icke-negativa vikter).
- ▶ Upprepad Bellman-Ford: $O(n^2 m)$.

Liknande uppdateringar sker flera gånger i dessa. Kan man använda dynamisk programmering?

Rekursivt samband

Givet är en graf $G = (V, E)$.

- ▶ $d_{k,u,v}$ är minsta avståndet från u till v på en väg som endast passerar noderna $1, \dots, k$ förutom start u och slut v .
- ▶ Vi tänker på $e \notin E$ som om $e \in E$ och $w(e) = \infty$.

Då gäller:

$$d_{0,u,v} = w(u, v)$$

$$d_{k,u,v} = \min\{d_{k-1,u,v}, d_{k-1,u,k} + d_{k-1,k,v}\} \text{ för } k = 1, \dots, n$$

Floyd-Warshall

I algoritmen bygger vi en n^3 tabell D , där så småningom $D[k, u, v] = d_{k,u,v}$.

Vi håller också reda på vilken den optimala vägen är med en tabell där $P[k, u, v]$ är sista noden på en optimal väg från u till v som endast besöker $1, \dots, k$. Den optimala vägen erhålls då genom att följa dessa pekare baklänges.

Floyd-Warshall (Initiering)

Input: En graf G med viktade kanter.

Output: Ingen.

INIT(G)

- (1) **foreach** $u \in V$
- (2) **foreach** $v \in V$
- (3) **for** $k = 0$ **to** n
- (4) $D[k, u, v] \leftarrow \infty$
- (5) $P[k, u, v] \leftarrow \perp$
- (6) **foreach** $u \in V$
- (7) **foreach** $k \in V$
- (8) $D[k, u, u] = 0$
- (9) **foreach** $(u, v) \in E$
- (10) $D[0, u, v] \leftarrow w(u, v)$
- (11) $P[0, u, v] \leftarrow u$

Floyd-Warshall

Input: En graf G med viktade kanter.

Output: Kortaste vägen mellan varje par av noder.

FLOYDWARSHALL(G)

```
(1)  INIT()
(2)  for  $k = 1$  to  $n$ 
(3)    foreach  $u \in V$ 
(4)      if  $D[k - 1, u, k] < \infty$ 
(5)        foreach  $v \in V$ 
(6)          if  $D[k - 1, u, k] + D[k - 1, k, v] \geq D[k - 1, u, v]$ 
(7)             $D[k, u, v] \leftarrow D[k - 1, u, v]$ 
(8)             $P[k, u, v] \leftarrow P[k - 1, u, v]$ 
(9)          else
(10)             $D[k, u, v] \leftarrow D[k - 1, u, k] + D[k - 1, k, v]$ 
(11)             $P[k, u, v] \leftarrow P[k - 1, k, v]$ 
(12)  return ( $D[n, \cdot, \cdot], P[n, \cdot, \cdot]$ )
```

Floyd-Warshall

- ▶ Vi kan se tabellen D som en $O(n^2)$ -stor tabell $D[k, \cdot, \cdot]$ för varje $k = 0, \dots, n$. Motsvarande gäller för P .
- ▶ Notera att tabellen $D[k, \cdot, \cdot]$ är definierad i termer av tabellen $D[k - 1, \cdot, \cdot]$. Motsvarande gäller för P .
- ▶ Alltså kräver algoritmen endast $O(n^2)$ minne om vi implementerar den så att vi endast spar tabellvärden för $k - 1$ när vi räknar ut tabellen för k .
- ▶ Observation: u ingår i en negativ cykel precis då $D[k, u, u] < 0$.

Glesa grafer

Floyd-Warshall går i tid $O(n^3)$, vilket gör Dijkstra snabbare $O(nm \log n)$ i glesa grafer, dvs när $m < n^2 / \log n$ i detta fall, men Dijkstra fungerar inte med negativa vikter.

Vad gör vi åt det?

Johnson's Algorithm

Inför potentialfunktion $\pi : V \rightarrow \mathbb{R}$ och skapa ny viktfunction från den gamla enligt:

$$w'(u, v) = w(u, v) + \pi(u) - \pi(v)$$

Notera att:

- En väg $e_1, \dots, e_t$ med kostnad $\sum_{i=1}^t w(e_i)$ får kostnaden

$$\sum_{i=1}^t w'(e_i) = \sum_{i=1}^t w(e_i) + \pi(u_1) - \pi(u_t)$$

- Varje väg mellan u_1 och u_t blir precis $\pi(u_1) - \pi(u_t)$ dyrare, så mängden optimala vägar är oförändrad.

Johnson's Algorithm

Input: En graf G med viktade kanter.

Output: Kortaste vägen mellan varje par av noder.

JOHNSON(G)

- (1) $G' \leftarrow (V \cup \{n+1\}, E \cup \{(n+1, u) : u \in V\})$
- (2) $(d, p) \leftarrow \text{BELLMANFORD}(G', w(\cdot), n+1)$
- (3) Definiera: $w'(u, v) = w(u, v) + d(u) - d(v)$
- (4) **foreach** $u \in V$
- (5) $(D[u], P[u]) \leftarrow \text{DIJKSTRA}(G, u, w')$
- (6) **return** (D, P)

Eulercykel

Definition. Låt $G = (V, E)$ vara en **oriktad/riktad graf**. En Eulerväg är en väg i grafen som utnyttjar varje **kant** precis en gång. En Eulercykel är en Eulerväg som slutar där den börjar.

Observera skillnaden mellan en Eulercykel och en Hamiltoncykel. Den senare besöker varje **nod** en gång och är NP-svår att hitta.

Eulercykel

Problem. Låt $G = (V, E)$ vara en **oriktad graf**. Avgör om det finns en Eulercykel och hitta en om det finns en sådan.

Hur gör vi detta?

Eulercykel

Vi antar att isolerade noder inte förekommer i grafen eftersom dessa aldrig behöver besökas. Då gäller:

- ▶ En oriktad graf har:
 - ▶ en Eulerväg om den är sammanhängande och alla noder utom högst två har ett jämnt antal kanter, och
 - ▶ en Eulercykel om alla noder har ett jämnt antal kanter.

- ▶ En riktad graf har en:
 - ▶ en Eulerväg om den är sammanhängande, alla noder utom högst två har lika många inkommande som utgående kanter, och de två speciella noderna har precis en utkant/inkant för mycket, och
 - ▶ en Eulercykel om dessutom alla noder har lika många inkommande som utgående kanter.

Eulercykel

Input: En graf $G = (V, E)$.

Output: En Eulerväg.

EULERPATH(G)

(1) $\text{vis}(e) = \text{false}$ för $e \in E$.

(2) **return** EULERPATHINNER(G, u)

EULERPATHINNER(G, u)

(1) **if** $\text{vis}(u, v) = \text{true}$ for all $(u, v) \in E(u)$

(2) **return** $\emptyset$

(3) **else**

(4) Take $(u, v) \in E$ with $\text{vis}(u, v) = \text{false}$.

(5) $\text{vis}(u, v) \leftarrow \text{true}$.

(6) $t \leftarrow \text{CONC}(((u, v)), \text{EULERPATHINNER}(G, v)), p \leftarrow \emptyset$

(7) **foreach** $(u, v) \in E$ with $\text{vis}(u, v) = \text{false}$

(8) $p \leftarrow \text{CONC}(p, \text{EULERPATHINNER}(G, u))$

(9) **return** $\text{CONC}(p, t)$

Maximalt flöde

Definition. Ett flöde i en riktad viktad graf $G = (V, E)$ från $s \in V$ till $t \in V$ är en funktion $f : E \rightarrow \mathbb{R}$ så att $0 \leq f(e) \leq w(e)$ för $e \in E$, och $\sum_{e \in \text{In}(u)} f(e) = \sum_{e \in \text{Out}(u)} f(e)$ för alla $u \neq s, t$.

Problem. Låt $G = (V, E)$ vara en **riktad viktad graf** och låt $s, t \in V$. Hitta ett maximalt flöde från s till t .

Hur gör vi detta?

FlowFinder

Givet en viktad riktad graf $G = (V, E)$ och ett flöde f så kan vi försöka hitta en väg mellan s och t där det finns utrymme för mer flöde. Mer precist:

FLOWFINDER()

- (1) Sätt $p = (e_1, \dots, e_k)$ med $e_1 \in \text{Out}(s)$, $e_k \in \text{In}(t)$ och $w(e_i) - f(e_i) > 0$ om det går och annars $p = \perp$.
- (2) **return** p

Detta kan enkelt implementeras med BFS.

Ford-Fulkerson (Edmonds-Karp eller snarare Dinic)

Input: En graf $G = (V, E)$.

Output: Ett maximalt flöde f .

FORDFULKERSON(G)

(1) $w(u, v) \leftarrow 0$ för $(u, v) \in E^T$

(2) $E \leftarrow E \cup E^T$

(3) $f(e) \leftarrow 0$ för $e \in E$.

(4) **while** *true*

(5) $p \leftarrow \text{POSFLOWFINDER}()$

(6) **if** $p = \perp$

(7) **return** f

(8) **else**

(9) Skriv $(e_1, \dots, e_k) = p$.

(10) $d \leftarrow \min_i \{w(e_i) - f(e_i)\}$

(11) $f(e_i) \leftarrow f(e_i) + d$

(12) $f(e_i^{-1}) \leftarrow f(e_i^{-1}) - d$