

0.1 Legrövidebb utak konzervatív hosszfüggvény esetén 1

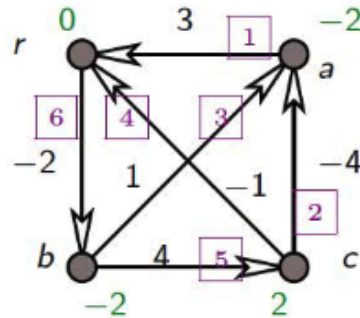
Könnyű olyan példát találni, ahol a Dijkstra-algoritmus konzervatív hosszfüggvény esetén hibás eredményt ad. Azonban konzervatív hosszfüggvény esetén is igaz, hogy

- (r, l) -fb élmenti javítása (r, l) -fb-t eredményez, ill.
- ha egy (r, l) -fb-ben nem végezhető érdemi élmenti javítás, akkor pontos.

konzervatív hosszfüggvény esetén is hasonló startégiát követünk: Élmenti javításokat végzünk a triviális (r, l) -fb-en, míg van érdemi javítás.

Ford-algoritmus: Input: $G = (V, E), l : E \rightarrow \mathbb{R}, r \in V$. Output: $dist_l(r, l) \forall v \in V$ Működés: f_0 a triviális (r, l) -fb, $|V| = n, E = \{e_1, e_2, \dots, e_m\}$. Az i -dik fázis $i = 1, 2, \dots, n-1$ -re az alábbi f_i -t f_{i-1} -ből kapjuk, az e_1, \dots, e_m élmenti javítások után. OUTPUT: $dist_l(r, v) = f_{n-1}(v) \forall v \in V$.

3. fázis



	r	a	b	c
f_0	0	∞	∞	∞
f_1	0	∞	∞	∞
f_2	0	∞	-2	∞
f_3	0	-1	-2	2

Állítás: Ha l konzervatív, akkor $dist_l(v) \forall v \in V$.

Biz: $f_1(v) = dist_l(r, v)$ ha $\exists \leq 1$ -élű legrövidebb rv -út. $f_2(v) = dist_l(r, v)$ ha $\exists \leq 2$ -élű legrövidebb rv -út. ... $f_{n-1}(v) = dist_l(r, v)$ ha $\exists \leq (n-1)$ -élű legrövidebb rv -út. Tehát $f_{n-1}(v) = dist_l(r, v) \forall v \in V$. \square

Megf: Ha $f_i = f_{i-1}$, akkor a Ford-algoritmust az i -dik fázis után be lehet fejezni, hisz nincs érdemi élmenti javítás, így $f_{n-1} = f_i$.

Megj: Az $f_{n-1}(v)$ -t beállító élek legrövidebb utak fáját alkotják.

Biz: A Dijkstra esethez hasonló. Tetszőleges v csúsból visszafelé követve a végső értékeket beállító éleket $f_{n-1}(v)$ hosszúságú rv -utat találunk. \square

"Lépésszámanalízis": Ha a $|V(G)| = n$ és $|E(G)| = m$, akkor minden fázisban $\leq m$ élmenti javítás, ami $konst \cdot m$ lépés. Ez összesen $\leq konst \cdot (n-1) \cdot m \leq konst \cdot n^3$ lépés, az algoritmus hatékony.

0.2 Legrövidebb utak konzervatív hosszfüggvény esetén 2

Tegyük fel, hogy $G = (V, E), l : E \rightarrow \mathbb{R}$ és $V = \{v_1, v_2, \dots, v_n\}$. Jelölje $d^{(k)}(i, j)$ a legrövidebb olyan $v_i v_j$ -út hosszát, aminek belső csúcsai csak v_1, v_2, \dots, v_k lehetnek.

Megf: (1) $d^{(n)}(i, j) = dist_l(v_i, v_j), v_i v_j \in E \Rightarrow d^{(0)}(i, j) = l(v_i, v_j)$ (2) $d^{(0)}(i, j) = 0$, különben $d^{(0)}(i, j) = \infty$. (3) Ha l konzervatív, akkor tetszőleges i, j ill. $k \leq n$ esetén $d^{(k+1)}(i, j) = \min\{d^{(k)}(i, j), d^{(k)}(i, k+1) + d^{(k)}(k+1, j)\}$ teljesül.

Biz: Tekintsünk egy $d^{(k+1)}(i, j)$ -t meghatározó P utat.

I. eset: $v_{k+1} \notin P$. Ekkor $d^{(k+1)}(i, j) = d^{(k)}(i, j)$, és $d^{(k+1)}(i, j) \leq d^{(k)}(i, k+1) + d^{(k)}(k+1, j)$.

II. eset: $v_{k+1} \in P$. Ekkor $d^{(k+1)}(i, j) \leq d^{(k)}(i, j)$, és $d^{(k+1)}(i, j) = d^{(k)}(i, k+1) + d^{(k)}(k+1, j)$.

Mindkét esetben helyes a képlet. \square

Floyds-algoritmus: Input: $G = (V, E)$, konzervatív $l : E \rightarrow \mathbb{R}$. Output: $dist_l(u, v) \forall u, v \in V$ Működés: $d^{(0)}$ felírása (2) alapján. Az i -dik fázis: $d^{(i-1)}$ -ből meghatározzuk $d^{(i)}$ -t (3) alapján. OUTPUT: $d^{(n)}(u, v) = dist_l(u, v) \forall u, v \in V$.

”Lépésszámanalízis”: A $d^{(0)}$ felírása $konst \cdot n^2$ lépés. Minden fázis $konst' \cdot n^2$. Mivel összesen n fázis van, a lépésszám legfeljebb $konst'' \cdot n^3$ lépés, az algoritmus hatékony.

Ford vs Floyd: Konzervatív hosszfüggvényre működnek helyesen. Mindkét algoritmus talál bizonyítékot, ha l nem konzervatív. (!!)

A Ford csak egy gyökérből, a Floyd bármely két csúcs között talál legrövidebb utat. (!!)

A Ford ritka gráfokra jelentősen olcsóbb, sok él esetén a Floyd nem sokkal drágább.

0.3 Depth First Search (DFS)

”Mélységi bejárás (DFS): A bejárás során mindig a legutolsónak elért csúcsot választjuk az 1. esetben.

Mélységi és befejezési számozás: DFS után $m(v)$ ill. $b(v)$ a v csúcs elérési ill. befejezési sorrendben kapott sorszáma.

Megj: A BFS konkrét megvalósításában szükség van arra, hogy az **elért** csúcsokat úgy tároljuk, hogy könnyű legyen kiválasztani az **elért** csúcsok közül a legkorábban elértet. Erre egy célszerű adatstruktúra a *sor* (avagy *FIFO lista*). Ha a BFS megvalósításában ezt az adatstruktúrát *veremre* (más néven *FIFO listára*) cseréljük, akkor a DFS egy megvalósítása adódik.

Megf: Tegyük fel, hogy a G gráf éleit DFS után osztályoztuk.

(1) Ha uv **faél**, akkor $m(u) < m(v)$ és $b(u) > b(v)$.

Biz: v -t u -ból értük el, ezért $m(u) < m(v)$. A v elérésekor u és v elért állapotúak. A DFS szerint v -t u elptt fejezzük be. \square

(2) Ha uv **előreél**, akkor $m(u) < m(v)$ és $b(u) > b(v)$.

Biz: u -ból v -be faéleken keresztül vezet irányított út. (1) miatt az út mentén a mélységi szám növekszik, befejezési csökken. \square

(3) Ha uv **visszaél**, akkor $m(u) > m(v)$ és $b(u) < b(v)$.

Biz: v -ből u -ba faéleken keresztül vezet irányított út. (1) miatt az út mentén a mélységi szám növekszik, a befejezési csökken. \square

Biz: $m(u) < m(v)$ esetén a DFS miatt v az u leszármazottja lenne. Ezért $m(u) > m(v)$. Ha u -t a v befejezése előtt értenék el, akkor u a v leszármazottja lenne. Ezért az alábbi sorrendben történik u és v evolúciója: **v** **elérése**, **v** **befejezése**, **u** **elérése**, **u** **befejezése**. \square

(4) Ha uv **keresztél**, akkor $m(u) > m(v)$ és $b(u) > b(v)$.

(5) Irányítatlan gráf DFS bejárása után nincs keresztél.

Biz: Indirekt. Ha uv keresztél, akkor (4) miatt $m(u) > m(v)$, továbbá vu is keresztél, ezért $m(v) > m(u)$. Ellentmondás. \square

(6) Ha DFS után van visszaél, akkor G tartalmaz irányított kört.

Biz: A DFS fa visszaélhez tartozó alapköre a G egy irányított köre. \square

(7) Ha DFS után nincs visszaél, akkor G -ben nincs irányított kör.

Biz: Bmely irányított körnek van olyan uv éle, amire $b(u) < b(v)$. Ez az él csak visszaél lehet. \square

0.4 Direct Acyclic Graphs

Def: A $G = (V, E)$ irányított gráf **aciklikus** (más néven **DAG**), ha G nem tartalmaz irányított kört.

Példa: DAG-ot úgy kaphatunk, hogy egy G irányítatlan gráf csúcsait csupa különbözőszámmal megszámozzuk, és minden élt a kisebb számot viselő csúcsból a nagyobbba irányítunk.

Ha ugyanis lenne az így megirányított gráfban irányított kör, akkor az élei mentén a számok végig növekednének, ami lehetetlen. Azt fogjuk ihazolni, hogy a fenti példa minden DAG-ot leír.

Def: A $G = (V, E)$ irányított gráf csúcsainak **topologikus sorrendje** alatt a csúcsok olyan sorrendjét értjük, amire igaz, hogy minden irányított él a sorban előbb álló csúcsból vezet a sorban későbbi csúcsba. ($V = \{v_1, v_2, \dots, v_n\}, v_i v_j \in E \Rightarrow i < j$)

Tétel: (G irányított gráf DAG) $\Leftrightarrow (V(G)$ -nek \exists topologikus sorrendje).

Biz: Tegyük fel, hogy \exists topologikus sorrend. Láttuk, hogy G ekkor DAG. \checkmark

Biz: Most tegyük fel, hogy G DAG, és futtassunk rajra egy DFS-t. Láttuk, hogy a DFS után nem lesz visszaél, ezért minden uv irányított élre $b(u) > b(v)$ teljesül. Ezért a csúcsok befejezési sorrendjének megfordítása a G csúcsainak egy topologikus sorrendje. \square

Köv: Irányított gráf aciklikussága DFS-sel gyorsan eldönthető: ha van visszaél, akkor a visszaél DFS-fabeli alapköre G egy irányított köre, így G nem DAG. Ha pedig nincs visszaél, akkor a fordított befejezési sorrend a G egy topologikus sorrendje, G tehát DAG.

Megj: DAG-ban topologikus sorrendet forráskeresések és forrástörlések alkalmazásával is találhatunk.

0.5 Leghosszabb út keresése

Ötlet: Az $l'(uv) = -l(uv)$ élhosszokkal a leghosszabb utak legrövidebbekké válnak. Olyanokat pedig tudunk keresni.

Gond: A módszerünk csak konzervatív élhosszokra működik. Irányítatlan gráfon ez nemnegatív élhosszokat jelent, ezért ez az ötlet itt nem segít. Irányított esetben nem baj a negatív élhossz, feltéve, hogy G DAG. Ekkor Ford, Floyd bármelyike használható.

Jó hír: Van egy még gyorsabb módszer: a dinamikus programozás. Ennek segítségével tetszőleges G DAG minden v csúcsához ki tudjuk számítani a v -be vezető leghosszabb utat. (Sőt! ...)

Leghosszabb út DAG-ban: Input: $G = (V, E)$ DAG, $l : E \rightarrow \mathbb{R}$. Output: $\max\{l(P) : P v\text{-be vezető út}\}$ minden $v \in V$ csúcsra. Működés: $[1] V = \{v_1, v_2, \dots, v_n\}$ topologikus sorrend meghatározása. $[2] i = 1, 2, \dots, n : f(v_i) = \max\{\max\{f(v_j) + l(v_j v_i) : v_j v_i \in E\}, 0\}$ Output: $f(v) \forall v \in V$

Helyesség: Ha a v_i -be vezető leghosszabb út utolsó előtti csúcsa v_j , akkor $f(v_i) = f(v_j) + l(v_j v_i)$.

Megj: Ha a fenti algoritmusban minden csúcsra megjelöljük az $f(v)$ értéket beállító élt (éleket), akkor a megjelölt élek minden v csúcsba megadnak egy leghosszabb utat. Sőt: minden v -be vezető leghosszabb megkapható így.

0.6 A PERT probléma

Egy a, b, \dots tevékenységekből álló projektet kell végrehajtanunk.

Precedenciafeltételek: bizonyos (u, v) párok esetén előírás, hogy az u tevékenységet a v előtt kell elvégezni, ezért v az u kezdetét követően $c(uv)$ időkorlát elteltével kezdhető.

Cél: minden v tevékenységhez olyan $k(v) \geq 0$ kezdési időpont meghatározása, ami nem sérti a preferenciafeltételeket, és a projekt végrehajtási ideje (a legnagyobb $k(v)$ érték) minimális.

G **irányított gráf** csúcsai a tevékenységek, élei pedig a precedenciafeltételek, az uv él hossza $c(uv)$.

Megf: (1) Ha G nem DAG, akkor a projekt nem hajtható végre. (2) Ha G DAG, akkor minden v tevékenység legkorábbi kezdési időpontja a v -be vezető leghosszabb út hossza.

Köv: A PERT probléma megoldása nem más, mint a G DAG minden csúcsára az oda vezető leghosszabb út meghatározása.

Terminológia: G leghosszabb útja **kritikus út**, amivől több is lehet. Kritikus út csúcsai a **kritikus tevékenységek**.

Megf: Ha egy kritikus tevékenység nem kezdődik el a lehető legkorábbi időpontban, akkor az egész projekt végrehajtása csúszik.