Dijkstra algoritmus

Csima Judit
BME SZIT
csima@cs.bme.hu

2019. december 4.

Ebben a részben azt a speciális esetét nézzük meg a legrövidebb út keresési feladatnak, amikor a gráf minden élének nemnegatív az élsúlya. Adott tehát egy irányított vagy irányítatlan, élsúlyozott G gráf, ahol minden élsúly nemnegatív és adott továbbá egy s kezdőcsúcs G-ben és szeretnénk meghatározni a legrövidebb utat s-ből minden v csúcsába a gráfnak. Ebben az esetben nem kell aggódnunk a negatív összsúlyú kör miatt, hiszen nincsenek is negatív élek egyáltalán.

A Dijkstra algoritmus elve

A DAG-ban működő, topologikus sorrendet használó legrövdebb utat kereső algoritmusban minden lépésben meg tudtuk határozni egy újabb v csúcsra a *távolság*$[v]$ értéket, a topologikus sorrend szerint haladva. A *távolság* tömbön kívül egy *honnan* tömböt is feltöltöttünk, ahol minden v csúcsra *honnan*$[v]$ adta meg az s-ből v-be vezető legrövidebb úton a v-t megelőző csúcsot.
Dijkstra algoritmusában is igaz lesz, hogy minden körben egy újabb csúcsra határozzuk meg az s-től való távolságot, de előre nem ismert, hogy milyen sorrendben tudjuk ezeket a távolságokat kiszámolni. Az azonban itt is igaz lesz, hogy magukat az utakat egy *honnan* tömb segítségével tároljuk majd, itt is *honnan*$[v]$ adja majd meg az s-ből v-be vezető legrövidebb úton a v-t megelőző csúcsot. Ezen *honnan* értékek segítségével, v-ből vissafelé lépegetve meg tudjuk majd kapni a legrövidebb utat.

Az algoritmus elve a következő:

1. Lesz egy KÉSZ halmaz, ebben azok a csúcsok lesznek, amikre már tudom az s-től vett távolság értékét. Ezeket a távolságokat egy *távolság* tömbben fogom tárolni, *távolság*$[v]$ egy valós érték lesz, ha a v csúcs már KÉSZ-ben van, egyébként, a KÉSZ-en kívüli csúcsokra *távolság*$[v] = \infty$.

2. Azokra a csúcsokra, amik még nincsenek benne a KÉSZ halmazban (tehát még nem tudom a távolságukat s-től) egy d tömbben fogok értéket nyilvántartani: $d[v]$ a legrövidebb olyan s-ből v-be vezető út hossza lesz, ami végig a KÉSZ-ben halad, csak az utolsó, a v-be vezető éllel lép ki a KÉSZ-ből. Úgy is fogalmazhatjuk, hogy a KÉSZ-en kívüli v csúcsokra

$$d[v] = \min_{u \to v, u \in KÉSZ} \{távolság[u] + c(u, v)\}$$

Ez a képlet azért helyes, azaz azért adja meg a legrövidebb olyan s-ből v-be vezető út hosszát, ami végig a KÉSZ-ben halad és csak az utolsó, a v-be vezető éllel lép ki a KÉSZ-ből, mert egy ilyen út a KÉSZ-en belül eljut valami u csúcsba (ha a legrövidebb utat keressük, akkor ezen rész hossza *távolság*[u]), majd innen a $c(u, v)$ súlyú uv éllel éri el v-t. Azért kell minimumot venni az összes lehetséges KÉSZ-beli u csúcsra, ahonnan vezet él v-be, mert minden lehetőséget figyelembe kell vennünk a v-t megelőző utolsó, KÉSZ-beli u csúcsra.

3. Egy *honnan* tömbben fogjuk tárolni azt az információt, hogy a legrövidebb út honnan jön. Ha a v csúcs már KÉSZ-ben van, akkor *honnan*[v] értéke az a csúcs, ami a legrövidebb, *távolság*[v] hosszú s-ből v-be vezető úton v előtt következik.
Ha a v csúcs még nincs a KÉSZ-ben, akkor *honnan*[v] értéke az a csúcs, ami a legrövidebb, s-ből v-be v kivételével végig a KÉSZ-ben haladó, $d[v]$ hosszú úton v előtt következik.

4. Minden fázisban a KÉSZ-en kívüli csúcsok közül kiválasztjuk azt a v^* csúcsot, melynek d értéke a legkisebb és ezt a csúcsot KÉSZ-be tesszük, *távolság*[v^*] = $d[v^*]$ távolság értékkel. Ezzel egyidejűleg $d[v^*] = *$ lesz, hiszen v^* ekkor már nem KÉSZ-en kívüli.

5. Miután v^* a KÉSZ-be került, lehetséges, hogy v^* egy w szomszédjára rövidebb utat kapunk végig a KÉSZ-en át haladva úgy, hogy az utolsó csúcs w előtt v^*, ezért v^* minden w szomszédjára megnézzük, hogy *távolság*[v^*]+$c(v^*, w)$ kisebb-e, mint $d[w]$ (azaz rövidebb-e a v^*-on keresztül most megnyíló út w-be) és ha igen, akkor $d[w]$-t *távolság*[v^*]+$c(v^*, w)$-re, *honnan*[w]-t pedig v^*-ra állítjuk (hiszen a legrövidebb ismert utat v^* felől találtuk).

Az algoritmus szövegesen

- Az eljárás elején KÉSZ =$\{s\}$, mert ekkor csak az s csúcsra ismert a távolság értéke.

- A csúcsokkal indexelt *távolság* tömböt úgy töltjük fel, hogy *távolság*[s] = 0, minden más v csúcsra *távolság*[v] = ∞ (mert a többi csúcs még nincs KÉSZ-ben).

- A csúcsokkal indexelt d tömböt úgy töltjük fel, hogy $d[s] = *$ (mert s KÉSZ-ben van), $d[v] = (s, v)$, amennyiben van él s-ből v-be és végtelen egyébként (mert ekkor $d[v] = \min_{u \to v, u \in KÉSZ} \{távolság[u] + c(u, v)\} = távolság[s] + c(s, v) = 0 + c(s, v) = c(s, v)$, hiszen s az egyetlen csúcs KÉSZ-ben.)

- A csúcsokkal indexelt *honnan* tömböt úgy töltjük fel, hogy *honnan*[s] = s és *honnan*[v] = s, ha van él s-ből v-be (hiszen ekkor a $d[v]$-ben tárolt hosszú eddig ismert legrövidebb út s-ből jön) és *honnan*[v] = $*$, ha nincs él s-ből v-be (mert ilyenkor még semmilyen utat nem találtunk v-be a KÉSZ-en keresztül).

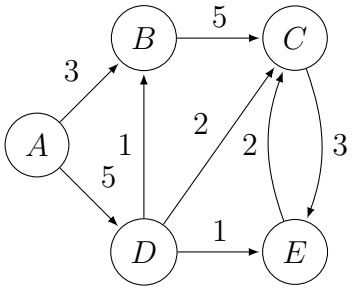
- Ezután a következőt csináljuk mindaddig, míg van olyan KÉSZ-en kívüli v csúcs melynek $d[v]$ értéke nem végtelen:
  - megkeressük azt a KÉSZ-en kívüli v^* csúcsot, melyre a d érték a legkisebb
  - v^*-ot a KÉSZ-be rakjuk
  - beállítjuk *távolság*[v^*]-ot *távolság*[v^*] = $d[v^*]$-vel

– beállítjuk $d[v^*] = *$-ot (mert v^* már KÉSZ-ben van)

– v^* minden olyan w szomszédjára, ami még nincs KÉSZ-ben ha *távolság*$[v^*]+c(v^*, w) <$ $d[w]$ (azaz ha v^*-on át van rövidebb út w-be a KÉSZ-en belül), akkor $d[w] :=$ *távolság*$[v^*] + c(v^*, w)$ és *honnan*$[w] := v^*$.

Egy példa az algoritmus futására

Tekintsük az alábbi irányított gráfot és futtassuk le benne Dijkstra algoritmusát az A csúcsból:



Az algoritmus kezdetén a KÉSZ halmaz és a három tömb így néz ki:

$KÉSZ = \{A\}$

távolság:

A	B	C	D	E
0	∞	∞	∞	∞

d:

A	B	C	D	E
*	3	∞	5	∞

honnan:

A	B	C	D	E
A	A	*	A	*

Most a legkisebb d érték a B csúcsnál van, ezért B megy a KÉSZ-be és ennek a csúcsnak a *távolság* értékét állítjuk be *távolság*$[B] = d[B] = 3$ módon, illetve $d[B] = *$ lesz, majd B KÉSZ-en kívüli szomszédait kell vizsgálni (ami most csak a C csúcs), hogy $d[C] = \infty$-nél kisebb-e *távolság*$[B] + c(B, C) = 3 + 5 = 8$ és mivel igen, ezért $d[C] = 8$ és *honnan*$[C] = B$ lesz:

$KÉSZ = \{A, B\}$

távolság:

A	B	C	D	E
0	3	∞	∞	∞

d:

A	B	C	D	E
*	*	8	5	∞

honnan:

A	B	C	D	E
A	A	B	A	*

Most a legkisebb d érték a D csúcsnál van, ezért D megy a KÉSZ-be és ennek a csúcsnak a *távolság* értékét állítjuk be $távolság[D] = d[D] = 5$ módon, $d[D] = *$ lesz, majd D KÉSZ-en kívüli szomszédait (azaz C-t és E-t) kell vizsgálni, hogy $d[C] = 8$-nál kisebb-e $távolság[D] + c(D, C) = 5 + 2 = 7$ és mivel igen, ezért $d[C] = 7$ és $honnan[C] = D$ lesz,
illetve $d[E] = \infty$-nél kisebb-e $távolság[D] + c(D, E) = 5 + 1 = 6$ és mivel igen, ezért $d[E] = 6$ és $honnan[E] = D$ lesz:

$KÉSZ = \{A, B, D\}$

távolság:

A	B	C	D	E
0	3	∞	5	∞

d:

A	B	C	D	E
*	*	7	*	6

honnan:

A	B	C	D	E
A	A	D	A	D

Most a legkisebb d érték az E csúcsnál van, ezért E megy a KÉSZ-be és ennek a csúcsnak a *távolság* értékét állítjuk be $távolság[E] = d[E] = 6$ módon, $d[E] = *$ lesz, majd E KÉSZ-en kívüli szomszédait (azaz C-t) kell vizsgálni, hogy $d[C] = 7$-nél kisebb-e $távolság[E] + c(E, C) = 6 + 2 = 8$, de mivel nem, ezért nem kell itt módosítani:

$KÉSZ = \{A, B, D, E\}$

távolság:

A	B	C	D	E
0	3	∞	5	6

d:

A	B	C	D	E
*	*	7	*	*

honnan:

A	B	C	D	E
A	A	D	A	D

Most a legkisebb d érték a C csúcsnál van, ezért C megy a KÉSZ-be és ennek a csúcsnak a *távolság* értékét állítjuk be $távolság[C] = d[C] = 7$ módon, $d[C] = *$ lesz. Mivel C-nek nincsenek KÉSZ-en kívüli szomszédai, ezért módosítani nem kell:

$KÉSZ = \{A, B, D, E, C\}$

távolság:

A	B	C	D	E
0	3	7	5	6

d:

A	B	C	D	E
*	*	*	*	*

honnan:

A	B	C	D	E
A	A	D	A	D

Mivel C volt az utolsó csúcs, aki még nem került a KÉSZ-be, ezért már nincs olyan csúcs, ahol d ne lenne $*$, az algoritmus leáll, a legrövidebb utak és hosszaik: B-be AB, hossza 3; C-be ADC, hossza 7; D-be AD, hossza 5, E-be ADE, hossza 6.

Az algoritmus helyessége

Indukciós bizonyítást használunk, a KÉSZ-be kerülés sorrendje szerinti indukcióval belátjuk, hogy minden csúcsra helyes a kiszámolt *távolság* érték. Azt fogjuk tehát belátni, hogy

- a legelső csúcsra, azaz s-re helyes a $távolság[s] = 0$ érték. (Ez az indokció alap esete.)

- ha minden csúcsra, ami már a KÉSZ-ben, van helyes a *távolság*, akkor a következőnek bekerülő v^* csúcsra is helyes értéket kapunk az algoritmussal.

Nézzük meg először, hogy miért helyes $távolság[s] = 0$, azaz miért igaz, hogy a legrövidebb út s-ből s-be 0 hosszú. Egyrészt s-ből s-be van 0 hosszú út (ami egy élet sem tartalmaz), másrészt ennél rövidebb út nem lehet, hiszen a gráfban nincsenek negatív élek.

Az általános lépéshez tegyük fel, hogy már néhány csúcs KÉSZ-ben van, ezekre mind jó a *távolság* érték és most éppen abban a helyzetben vagyunk, hogy a legkisebb d értékkel rendelkező v^* csúcsot készülünk KÉSZ-be tenni $távolság[v^*] = d[v^*]$ értéket beállítva. Azt kell tehát megmutatnunk, hogy ez a $távolság[v^*]$ érték helyes lesz, azaz valóban ennyi a legrövidebb út hossza s-ből v^*-ba.

Ehhez két dolgot kell belátnunk:

(A) Van $távolság[v^*]$ hosszú út s-ből v^*-ba.

(B) Minden olyan U út, ami s-ből v^*-ba vezet, az legalább $távolság[v^*]$ hosszú. (Tehát $távolság[v^*]$ a legrövidebb lehetséges hossz.)

(A)
Mivel $távolság[v^*] = d[v^*]$, ezért azt nézzük meg, hogy $d[v^*]$ micsoda. Ez a d értékek definíciója miatt $d[v^*] = \min_{u \to v^*, u \in KÉSZ} \{távolság[u] + c(u, v^*)\} = távolság[u^*] + c(u^*, v^*)$
ahol u^* az a KÉSZ-beli csúcs, amire $honnan[v^*] = u^*$, azaz az a csúcs KÉSZ-ben, ahonnan a legjobb él érkezik v^*-ba.
Ez azt jelenti, hogy $távolság[v^*] = távolság[u^*] + c(u^*, v^*)$, azaz az (A) részben azt kell megmutatnunk, hogy van $távolság[u^*] + c(u^*, v^*)$ hosszú út s-ből v^*-ba. Ez pedig azért igaz, mert egy ilyen hosszúságú utat kaphatunk például úgy, hogy a legrövidebb s-ből u^*-ba vezető utat (melynek hossza $távolság[u^*]$, hiszen azt már tudjuk, hogy a KÉSZ-ben levő u^* csúcsra $távolság[u^*]$ helyes) megtoldjuk az (u^*, v^*) éllel, melynek hossza $c(u^*, v^*)$.

(B)
Azt kell tehát még belátnunk, hogy ha veszünk egy akármilyen U utat s-ből v^*-ba, akkor ennek hossza legalább $távolság[v^*] = d[v^*] = távolság[u^*] + c(u^*, v^*)$, azaz minden s-ből v^*-ba vezető út legalább $d[v^*]$ hosszú.

Ha U egy olyan út, ami s-ből v^*-ba vezet, akkor nézzünk rá erre az útra abban a pillanatban, amikor az algoritmus v^*-ot éppen beválasztja a KÉSZ halmazba. Az ebben a pillantaban, még v^* beválasztása előtti helyzetben ránézve az U útra és a KÉSZ halmazra igaz, hogy az U úton van egy első olyan él, ami kivezet KÉSZ-ből (hiszen kezdete, az s csúcs, KÉSZ-beli, végpontja, a v^* csúcs pedig KÉSZ-en kívül van). Legyen ez az él, ahol U elhagyja a KÉSZ halmazt az (u', v') él.

Az U út hossza három részből áll össze: az s-ből u'-be vezető rész hosszából, az (u', v') él hosszából és a v'-ből v^*-ba vezető út hosszából. Az első rész, az s-ből u'-be vezető rész, legalább $távolság[u']$ hosszú, hiszen ez a rész egy s-ből u'-be vezető út, aminek hossza legalább annyi, mint a legrövidebb ilyen út hossza, ami pedig, az indukciós feltevés szerint, miszerint a $távolság$ értékek helyesek a KÉSZ-ben, éppen $távolság[u']$.

Azt kaptuk tehát eddig, hogy az U út hossza legalább $távolság[u'] + c(u', v')+$ a v'-ből v^*-ba vezető rész hossza. Ez utóbbi, azaz a v'-ből v^*-ba vezető út hossza nemnegatív (hiszen

minden út hossza nemnegatív, mert minden élsúly nemnegatív), ezért az U út hossza legalább $távolság[u'] + c(u', v')$. Vegyük észre ekkor, hogy v' egy KÉSZ-en kívüli csúcs és $távolság[u'] + c(u', v')$ éppen egy olyan összeg, amiknek a legkisebbjeként definiáltuk a d értéket: $d[v'] = \min_{u \to v', u \in KÉSZ} \{távolság[u] + c(u, v')\}$ vagyis $távolság[u'] + c(u', v')$ legalább $d[v']$. Azt kaptuk tehát, hogy az U út hossza legalább $d[v']$. De ez az a pillanat, amikor az algoritmus v^*-ot választja be a KÉSZ-be azért, mert ennek a csúcsnak a legkisebb a d értéke, vagyis $d[v'] \geq d[v^*]$, azaz az U út hossza legalább $d[v^*]$, pont amit be akartunk látni.

Az algoritmus pszeudokódja

A korábban látott szöveges változathoz képest egy részben kell csak pontosabban fogalmaznunk a pszeudokódban: hogyan frissítjük az éppen bekerülő v^* csúcs szomszédainak d és *honnan* értékét. Ezt egy ciklussal fogjuk megtenni, ahol végigmegyünk a gráf összes csúcsán és minden olyan w cscsra, amibe megy él v^*-ból ($A[v^*, w]$ nem végtelen) és amely w csúcs még nincs KÉSZ-ben (azaz $d[w]$ nem *) megnézzük, hogy $távolság[v^*] + c(v^*, w)$ kisebb-e, mint $d[w]$ és ha igen, akkor $d[w]$-t $távolság[v^*] + c(v^*, w)$-re, $honnan[w]$-t pedig v^*-ra állítjuk.

A teljes kód a következő lesz:

```
KÉSZ = {s}
távolság[v] = 0, ha v = s, egyébként távolság[v] = végtelen
d[v] = *, ha v = s, d[v] = c(s,v), ha van él s-ból v-be, egyébként d[v] = végtelen
honnan[v] = s, ha v = s vagy ha van él s-ból v-be, egyébként honnan[v] = *

ciklus amíg van KÉSZ-en kívüli csúcs, ahol d[v] nem végtelen:
    v* := az a csúcs, ahol d[v] minimális
    v* KÉSZ-be megy
    távolság[v*] := d[v*]
    d[v*] : = *
    ciklus w = 1-tól n-ig:
        ha A[v*,w] nem végtelen és d[w] nem *:
            ha távolság[v*] + c(v*, w) < d[w]:
                d[w] := távolság[v*] + c(v*, w)
                honnan[w] := v*
    ciklus vége
ciklus vége
```

Az algoritmus lépésszáma

A kezdeti lépések $O(n)$-ben mennek, mert három darab n méretű tömböt kell feltöltenünk és KÉSZ-t kell beállítanunk s-re.
Az amíg ciklus magja legfeljebb n-szer fut le, mert minden lépésben egy csúcs KÉSZ-be kerül és leállunk, ha már nincs csúcs KÉSZ-en kívül. A ciklusmagban van egy minimumkeresés a d tömbben, ez $O(n)$, aztán konstans sok lépés, majd egy belső ciklus, aminek a magja n-szer

fut le és a mag konstans, tehát a belső ciklus $O(n)$-es. Vagyis az amíg ciklus ciklusmagja $O(n) + O(1) + O(n) = O(n)$ és mivel n-szer fut el, ezért ez $O(n^2)$.