Legrövidebb és leghosszabb út keresése DAG-ban Legrövidebb és leghosszabb út keresése általános gráfokban

Csima Judit
BME SZIT
csima@cs.bme.hu

2019. december 4.

Legrövidebb út keresése élsúlyozott gráfban

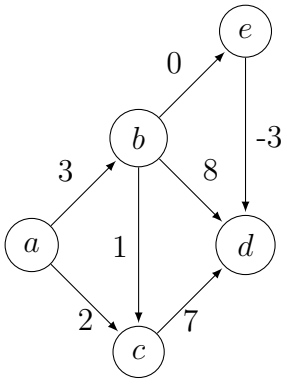
Az előző részben említettük, hogy a topologikus sorrend azért is hasznos, mert vannak olyan feladatok, amiket könnyebben meg tudunk oldani, ha az inputot alkotó gráfban tudunk topologikus sorrendet találni. Az egyik ilyen feladat a legrövidebb út keresése irányított vagy irányítatlan élsúlyozott gráfokban.

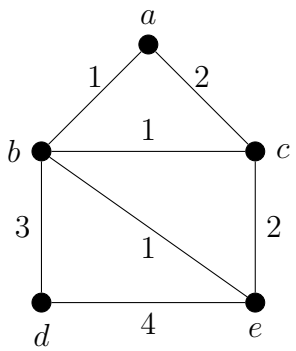
Élsúlyozott gráfok

Most először megismerkedünk az élsúlyozott gráf fogalmával és azzal, hogy hogyan lehet egy élsúlyozott gráfot reprezentálni szomszédossági mátrix-szal.

Egy élsúlyzott gráf (lehet irányított vagy irányítatlan is) esetén definiálva van az éleken egy $c(e)$ súlyfüggvény, ami a gráf minden éléhez egy valós számot rendel hozzá, ez az él súlya vagy hossza. Ez a szám nem feltételenül pozitív, lehet 0 vagy negatív érték is.

Az alábbi gráfok élsúlyozott gráfok, az első irányított, a második irányítatlan:





Az élsúlyozott gráfokat is szomszédossági mátrix-szal fogjuk megadni, de most ennek az A mátrixnak az elemei nem 0-1 értékek lesznek, hanem tetszőleges valós számok és ∞ értékek az alábbi szabály szerint:

- ha az irányított esetben az i csúcsból nincs él a j csúcsba, illetve az irányítatlan esetben nincsen él i és j között, akkor $A[i,j] = \infty$
- ha az irányított esetben az i csúcsból $c(i,j)$ súlyú él vezet a j csúcsba, illetve az irányítatlan esetben $c(i,j)$ súlyú él van i és j között, akkor $A[i,j] = c(i,j)$

A fenti gráfok szomszédossági mátrixai ezek lesznek:

$$\begin{bmatrix} \infty & 3 & 2 & \infty & \infty \\ \infty & \infty & 1 & 8 & 0 \\ \infty & \infty & \infty & 7 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & -3 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & 1 & 2 & \infty & \infty \\ 1 & \infty & 1 & 3 & 1 \\ 2 & 1 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty & 4 \\ \infty & 1 & 2 & 4 & \infty \end{bmatrix}$$

Azért kell ∞ értéket használnunk a nem létező él jelzésére, mert a 0 használata félrevezető lenne, hiszen előfordulhat 0 súlyú él is.

Az eddig tanult, nem élsúlyozott gráfokon futó algoritmusok kis változtatással ezután is használhatók lesznek (pl. BFS, DFS), annyi módosításra lesz csak szükség, hogy minden esetben amikor eddig az $A[i,j] == 1$ feltétel ellenőrzésével azt néztük meg, hogy egy nem élsúlyozott gráfban van-e él i-ből j-be, akkor most ezt a feltételt $A[i,j]! = \infty$ ellenőrzésre kell cserélnünk. Ez a módosítás nem érinti a lépésszámot, így az összes eddigi algoritmus a korábban tanult lépésszámmal fut élsúlyozott gráfok esetén is.

Legrövidebb út keresése adott kezdőcsúcsból

Ebben a feladatban az input egy élsúlyozott (irányított vagy irányítatlan) G gráfból és a G gráf egy s csúcsából áll, a feladat pedig az, hogy megkeressük a gráf minden v csúcsára az s-ből v-be vezető legrövidebb utat, ahol az út hossza alatt az utat alkotó élek élsúlyainak összegét értjük.

Vegyük észre, hogy ezt a feladatot már megoldottuk egy nagyon speciális esetben a BFS (szélességi bejárás) távolságot számoló változatával $O(n^2)$ időben. Ezt az eljárást akkor használtuk, amikor nem voltak élsúlyok és az út hossza az út éleinek számát jelentette, de ez az eset felfogható úgy is, hogy vannak élsúlyok, csak minden élsúly 1, erre az esetre tehát már van algoritmusunk.

Legrövidebb út keresése DAG-ban, tetszőleges élsúlyok mellett

Most egy másik speciális esetben fogjuk megoldani a legrövidebb utak problémáját: akkor, amikor a gráf egy DAG.

Ötlet

Az algoritmus ötlete a következő: topologikus sorrendet keresünk a gráfban (ami létezik, hiszen G DAG) és ezen sorrend szerint haladva minden v csúcsra meghatározzuk az s-től vett távolságot (távolság alatt a legrövidebb út hosszát értjük) és azt is, hogy a legrövidebb s-ből v-be vezető úton melyik a v-t megelőző csúcs, honnan érkezik az út v-be.
Ezeket az információkat két tömbben fogjuk tárolni:

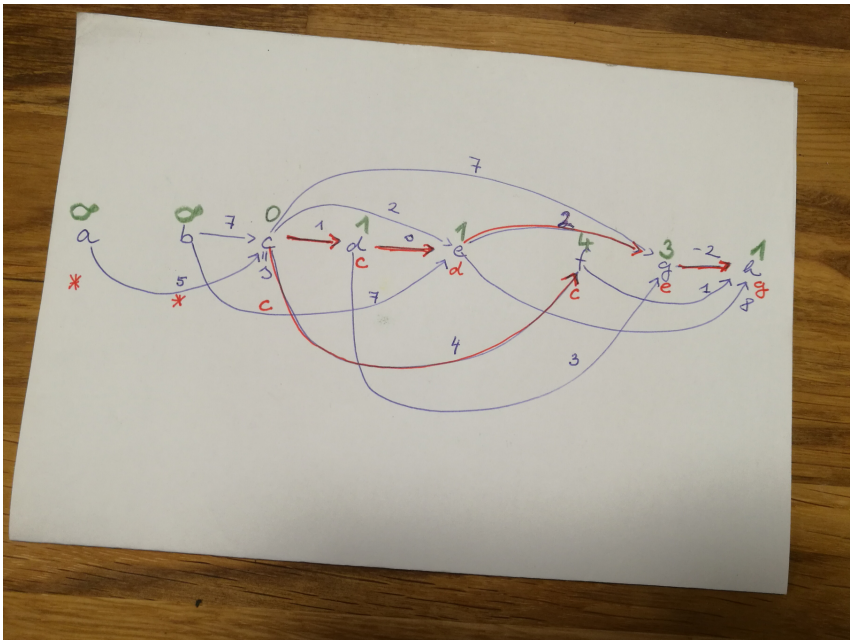
- a *távolság* tömb tárolja majd a csúcsok s-től vett távolságát
- a *honnan* tömb pedig minden csúcsra megadja, hogy melyik csúcsból érkezik a legrövidebb út

Az algoritmus szövegesen és az algoritmus jóságának indoklása

1. Toplogikus sorrendet keresünk G-ben a DFS-et hsználó eljárással.
2. *távolság[v]*$= \infty$ minden v-re és *honnan*$[v] = *$ minden v-re (még nem ismerünk egy távolságot sem és nem tudjuk, hogy honnan jönnek a legrövidebb utak)
3. Végigmegyünk a csúcsokon a toplogikus sorrend szerint és
   (a) Azokra a csúcsokra, amik s előtt vannak a topologikus sorrendben marad a *távolság[v]*$= \infty$ és *honnan*$[v] = *$ (mert az s-et megelőző csúcsokba nem lehet s-ből eljutni, hiszen a topologikus sorrendben minden él balról jobbra halad, vagyis ezekbe a csúcsokba egyáltalán nincsen út s-ből))
   (b) *távolság[s]*$= 0$ és *honnan*$[s] = s$ (mert s-ből saját magába csak az az egyetlen út van, amiben nincs él, hiszen a gráfban nincs kör)
   (c) Ha egy v csúcs s után van és s-ből el akarunk jutni v-be, akkor ezen az úton van egy v-t megelőző u csúcs, vagyis mindegyik, az s-ből v-be vezető út egy s-ből u-ba vezető útból és az u-ból v-be menő élből áll. Ez az u csúcs a topologikus sorrendben biztosan korábban van, mint v, hiszen v-be csak korábbi csúcsokból vezetnek élek. Mivel a csúcsokon a topologikus sorrend szerint haladva megyünk végig, ez azt jelenti, hogy amikor v-hez érünk, akkor *távolság*$[u]$ már ismert és a legrövidebb olyan s-ből v-be vezető útnak a hossza, ahol a v előtt közvetlenül u áll *távolság*$[u] + c(u, v)$. A legrövidebb s-ből v-be vezető út tehát az összes ilyen *távolság*$[u] + c(u, v)$ érték közül a legkisebb lesz (figyelembe vesszük az összes lehetséges u csúcsot), vagyis
   $$\textit{távolság}[v] = \min_{u \to v} \{\textit{távolság}[u] + c(u, v)\}$$
   Miközben meghatározzuk a fenti képlettel a *távolság*$[v]$ értéket, az is kiderül, hogy melyik u csúcsnál találtuk meg a minimumot, azaz honnan jön a legrövidebb út, erre az u csúcsra állítjuk be a *honnan*$[v]$-t.

Mielőtt megnéznénk a fenti algoritmus pszeudokódját és lépésszámát nézzük meg az algoritmus futását egy konkrét példán.

Az alábbi gráfot már rögtön úgy rajzolzuk fel, hogy a csúcsok topologikus sorrendben vannak, a csúcsok fölé írt zöld szám a *távolság*, a csúcsok alá írt piros érték a *honnan* tömb értéke, a pirossal jelölt élek mutatják ugyanezt az információt, hogy honnan érkezik a legrövidebb út. A kezdőcsúcs a c csúcs.



Nézzük végig csúcsról csúcsra, hogyan találjuk meg a *távolság* és a *honnan* tömb értékeit:

- Mivel a és b a c előtt vannak a toplogikus sorrendben, ezért rájuk *távolság[a]*$= \infty$, $honnan[a] = *$ és *távolság[b]*$= \infty$, $honnan[b] = *$.
- Mivel c a kezdőcsúcs: *távolság[c]*$= 0$, $honnan[c] = c$.
- A d csúcsba csak a c-ből vezet él, ezért
  $távolság[d] = \min\limits_{u \to d} \{távolság[u] + c(u,d)\} = távolság[c] + c(c,d) = 0 + 1 = 1$ és
  $honnan[d] = c$.
- Az e csúcsot három irányból lehet elérni: a b, a c vagy a d csúcsból, ezért:
  $távolság[e] = \min\limits_{u \to e} \{távolság[u] + c(u,e)\} =$
  $= \min\{távolság[b] + c(b,e),\ távolság[c] + c(c,e),\ távolság[d] + c(d,e)\} =$
  $= \min\{\infty + 7, 0 + 2, 1 + 0\} = 1$
  Az is kiderült, hogy a minimum a d csúcsnál volt, vagyis $honnan[e] = d$.
- Az f csúcsba csak a c-ből vezet él, ezért
  $távolság[f] = \min\limits_{u \to f} \{távolság[u] + c(u,f)\} = távolság[c] + c(c,f) = 0 + 4 = 4$ és
  $honnan[f] = c$.
- A g csúcsot három irányból lehet elérni: a c, a d vagy az e csúcsból, ezért:
  $távolság[g] = \min\limits_{u \to g} \{távolság[u] + c(u,g)\} =$

$= \min\{\textit{távolság}[c] + c(c,g),\ \textit{távolság}[d] + c(d,g),\ \textit{távolság}[e] + c(e,g)\} =$
$= \min\{0+7, 1+3, 1+2\} = 3$
Az is kiderült, hogy a minimum az e csúcsnál volt, vagyis $\textit{honnan}[g] = e$.

- A h csúcsot is három irányból lehet elérni: az e, az f vagy a g csúcsból, ezért:
$\textit{távolság}[h] = \min\limits_{u \to h} \{\textit{távolság}[u] + c(u,h)\} =$
$= \min\{\textit{távolság}[e] + c(e,h),\ \textit{távolság}[f] + c(f,h),\ \textit{távolság}[g] + c(g,h)\} =$
$= \min\{1+8, 4+1, 3-2\} = 1$
Az is kiderült, hogy a minimum a g csúcsnál volt, vagyis $\textit{honnan}[h] = g$.

Az algoritmus pszeudokódja és lépésszáma

A DAG-ban legrövidebb utat kereső algoritmus három részből áll:

1. Az algoritmus úgy kezdődik, hogy DFS segítségével topologikus sorrendet készítünk, ennek a pszeudokódját már korábban megtárgyaltuk.

2. Létrehozunk két tömböt, $\textit{távolság}[v] = \infty$ minden v-re és $\textit{honnan}[v] = *$ minden v-re.

3. Ezután a következő pszeudokóddal megyünk végig a csúcsokon a topologikus sorrendet követve, a kódbna A a szomszédossági mátrixot jelöli:

```
ciklus v-vel végig a csúcsokon a topologikus sorrendet követve:
    ha v az s csúcs előtt van:
        távolság[v] := végtelen  // marad végtelen
        honnan[v] := *                  // marad *
    egyébként ha v == s:
        távolság[v] : = 0
        honnan[v] := s
    egyébként:
        ciklus u = 1-től n-ig:
            ha A[u,v] != végtelen:                          // ha van él u-ból v-be
                ha távolság[u] + c(u,v) < távolság[v]:     // u-n át rövidebb az út
                    távolság[v] := távolság[u] + c(u,v)
                    honnan[v] := u
        ciklus vége
    elágazás vége
ciklus vége
```

A fenti kódban egyetlen rész van, ami a korábbiak után magyarázatra szorul: a belső ciklus az a rész, ahol megtaláljuk a minimális $\textit{távolság}[u] + c(u,v)$ értéket.

Az algoritmus lépésszámáról a következőt mondhatjuk:

1. A DFS-en alapuló, toplogikus sorrendet találó eljárásról már láttuk, hogy $O(n^2)$ lépés.

2. A két n méretű tömb létrehozása és feltöltése $O(n)$.

3. A fenti pszeudokódot alkotó külső ciklus magja n-szer fut le, mert minden csúcsot egyszer nézünk meg. A ciklusmag az s előtti v-k esetén $O(1)$, s-re szintén $O(1)$, az s utáni csúcsokra pedig egy n-szer lefutó ciklusmagú belső ciklus, ahol a ciklusmag $O(1)$, vagyis ebben az esetben a ciklusmag $O(n)$. Látjuk tehát, hogy a külső ciklus magja $O(n)$-es és mivel n-szer fut le, ezért a külső ciklus, vagyis a kód lépésszáma $O(n^2)$.

Azt kaptuk tehát, hogy ha a gráf DAG, akkor $O(n^2)$ lépésben tudunk legrövidebb utat találni egy s kezdőcsúcsból minden más csúcsba.

Leghosszabb út keresése DAG-ban egy adott kezdőcsúcsból

Ha nem a legkisebb összsúlyú, hanem a legnagyobb összsúlyú utat akarjuk megtalálni egy kezdő s csúcsból egy DAG-ba, akkor az előbbiekhez hasonló algoritmust használhatunk. Annyi módosításra van csak szükség, hogy a toploogikus sorrend megkeresése után a *távolság* tömb helyett egy *leghosszabb*[] tömböt hozunk létre úgy, hogy $leghosszabb[v] := -\infty$ minden v-re (a *honnan* tömb marad, ahogy volt, azaz $honnan[v] := *$).
Amikor pedig végigmegyünk a csúcsokon, akkor a bejövő $u \to v$ éleket végignézve a legnagyobb értéket választjuk, azaz a képletben

$$leghosszabb[v] = \max_{u \to v} \{leghosszabb[u] + c(u, v)\}$$

A pszeudokódban pedig:

```
ciklus v-vel végig a csúcsokon a topologikus sorrendet követve:
    ha v az s csúcs előtt van:
        leghosszabb[v] :=  - végtelen  // marad - végtelen
        honnan[v] := *                  // marad *
    egyébként ha v == s:
        leghosszabb[v] : = 0
        honnan[v] := s
    egyébként:
        ciklus u = 1-tól n-ig:
            ha A[u,v] != végtelen:                          // ha van él u-ból v-be
                ha leghosszabb[u] + c(u,v) > leghosszabb[v]:     // u-n át hosszabb az út
                    leghosszabb[v] := leghosszabb[u] + c(u,v)
                    honnan[v] := u
        ciklus vége
    elágazás vége
ciklus vége
```

A legrövidebb utak eseténél látott érveléshez teljesen hasonló módon látható be, hogy ez az eljárás helyes és mivel a kódben a lépésszámot érintő változtatás nem történt, ezért a lépésszám a leghosszabb út keresése esetén is $O(n^2)$.

Az, hogy mind legrövidebb, mind leghosszabb utat tudunk $O(n^2)$ lépésben találni, amennyiben a gráf DAG nagyon különleges és jó hír. Általános esetben, ha a gráf nem DAG sokkal nehezebb ezeket a feladatokat megoldani. A legrövidebb út keresését általános gráfokban a következő

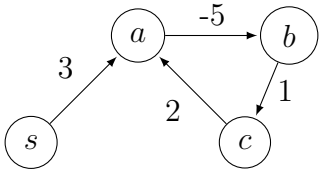
fejezetben tárgyaljuk (kicsit) és azt fogjuk látni, hogy a legjobb ismert algoritmus is $O(n^3)$ lépésszámú (szemben a DAG-os $O(n^2)$-es lépésszámmal).

Még nagyobb a különbség a DAG-os és az általános eset között a leghosszabb út keresése esetén. Ismert (de ezt nem tárgyaljuk részletesen), hogy a leghosszabb út keresése általános gráfban úgynevezett NP-teljes feladat, ami azt jelenti, hogy jelenleg nincsen rá hatékony algorimus, a legjobb ismert eljárások is exponenciális lépésszámúak, sőt egy híres sejtés szerint az NP-teljesség miatt soha nem is lesz erre a feladatra hatákony eljárás. Ezért kell nagyon örülni annak, hogy amennyiben a gráf DAG, akkor van egy $O(n^2)$-es algoritmusunk.

Legrövidebb út keresése adott kezdőcsúcsból általános esetben

Ebben a részben azt vizsgáljuk meg, anélkül, hogy a részletekbe nagyon belemennénk, hogy hogyan lehet legrövidebb utat keresni egy adott kezdőcsúcsból az általános esetben, ha sem a gráfra, sem az élsúlyokra nincsen semmi megkötés.

Azt vesszük észre először, hogy a teljesen általános esetben nem is biztos, hogy értelmes a feladat, előfordulhat, hogy nincsen legrövidebb összsúlyú elérése egy v csúcsnak az s kezdőcsúcsból. Ez például az alábbi gráfban is így van:



Ebben a gráfbna van egy -2 összhosszú elérés s, a, b úton s-ből b-be, de van -4-es is s, a, b, c, a, b-n át, sőt minél többször megyünk körbe az a, b, c körön, egyre kisebb és kisebb értékeket tudunk kapni.

Azt vettük most észre, hogy ha a gráfban van olyan kör, aminek összsúlya negatív és ez a kör elérhető az s-ből, akkor nem létezik legkisebb összsúlyú elérés. Ebből a helyzetből két kiút kínálkozik:

- Szorítkozzunk utakra, azaz olyan elérésekre, ahol egy csúcsot csak egyszer lehet használni, azaz keressük a legrövidebb valódi utat, ahol nincs ismétlődő csúcs. Sajnos ez a feladat is egy NP-teljes feladat, azaz nincs rá gyors algoritmus és valószínűleg nem is lesz soha.

- Zárjuk ki az olyan gráfokat a feladatból, amikben van negatív összsúlyú kör. Ebben az esetben már megoldható lesz a feladat, ismert rá $O(n^3)$ lépésszámú algoritmus, a Bellman-Ford algoritmus. Ezt nem fogjuk tanulni, csak annyit kell róla tudni, hogy létezik és hogy $O(n^3)$ lépéssben megtalálja a legrövidebb összsúlyú utat a kezdő s csúcsból minden más v csúcsba. Az is igaz ráadásul, hogy a Bellman-Ford algoritmus nemcsak megtalálja a legrövidebb utakat, ha nincsen a gráfban negatív kör, de egyúttal arra is képes, hogy észrevegye, ha a gráfban van negatív kör. Vagyis ha a legrövidebb út keresését akarjuk megoldani egy G gráfban egy adott s csúcsból, akkor bátran elindíthatjuk a Bellman-Ford algoritmust, ami vagy jelzi, hogy a gráfban van negatív kör és így a feladat nem értelmes

vagy pedig (ha nem ez az eset van, akkor) megtalálja a legrövidebb utat s-ből minden v csúcsba.

A tanult legrövidebb út keresésére használható algoritmusok összefoglalása

Eddig három algoritmusról tanultunk, amivel a legrövidebb utakat lehet megkeresni egy adott kezdőcsúcsból. Foglaljuk most össze, hogy melyik mikor használható és mennyi a lépésszáma:

- Ha a gráfban minden élsúly 1, akkor szélességi bejárást használhatunk irányított és irányítatlan esetben is, a lépésszám $O(n^2)$.
- Ha a gráf egy DAG (ekkor a gráf biztosan irányított is, hiszen ennek a fogalomnak csak irányított gráfban van értelme), akkor tetszőleges élsúlyok esetén használhatjuk a most tanult, a topologikus sorrendet használó eljárást, aminek lépésszáma $O(n^2)$.
- Ha a gráfra csak annyi a megkötés, hogy ne legyen benne negatív összsúlyú kör, akkor az ebben a tárgyban nem részletezett, $O(n^3)$ lépésszámú Bellman-Ford algoritmust lehet használni.

Vegyük észre, hogy az első két esetben nem kellett külön kikötni, hogy a gráfban nincs negatív kör, mert az első esetben minden élsúly 1, azaz egyáltalán nincsenek neatív élek, a második esetben pedig egyáltalán nincs kör a gráfban.