Legrövidebb-út algoritmusok és dinamikus programozás

Kátai Zoltán

katai_zoltan@ms.sapientia.ro
Sapientia-EMTE

Absztrakt. A dolgozat egy olyan tanítási módszert mutat be, amely segít a tanulóknak „felülnézet-rálátást" nyerni három híres legrövidebb-út algoritmusról: topologikus sorrenden alapuló legrövidebb-út algoritmus, Dijkstra algoritmusa és Bellman-Ford algoritmus. A módszer célja az, hogy az algoritmusok bemutatásán túl, olyan nézőpontba juttassuk a tanulót, amelyből feltárulnak előtte ezen algoritmusok közötti elvi, alapvető, sőt árnyalatbeli különbségek, illetve hasonlóságok. E cél elérése érdekébe úgy mutatjuk be a szóban forgó algoritmusokat, mint rokon dinamikus programozásos stratégiákat.

1. Bevezetés

Comenius [1] a következő kijelentést tette a tanítási módszereket illetően: „*Tanítani szinte nem is jelent mást, mint megmutatni, miben különböznek egymástól a dolgok a különböző céljukat, megjelenési formájukat, és eredetüket illetően. ... Ezért aki jól megkülönbözteti egymástól a dolgokat, az jól is tanít.*" E cikkben bemutatott didaktikai módszer elsősorban erre az alapelvre épül. Egy olyan tanítási, illetve tanulási módszerről van szó, amely segít a tanulóknak, úgymond felülnézetből látni három híres legrövidebb-út algoritmust: topologikus sorrenden alapuló algoritmus, Dijkstra algoritmusa, Bellman-Ford algoritmus. A módszer célja az, hogy az algoritmusok bemutatásán túl, olyan nézőpontba juttassuk a tanulót, amelyből feltárulnak előtte ezen algoritmusok közötti elvi, alapvető, sőt árnyalatbeli különbségek, illetve hasonlóságok. [3] A comeniusi alapelvvel összhangban ez nélkülözhetetlen, ha uralni szeretnénk a gráfelmélet e területét.

Az említett algoritmusok az alábbi feladatot oldják meg különböző esetekben: Legyen $G = (V, E)$ egy irányított gráf, amelyhez a *súly*: $E \to \mathbf{R}$ súlyfüggvényt rendeljük. Egy út súlyát úgy definiáljuk mint az út menti élek súlyainak összegét. Az `s` és `v` csúcsok között a legrövidebb út súlyát `lru(s,v)` jelöli. Ha `s` és `v` között nincs út, akkor definíció szerint `lru(s,v)` $= \infty$. Ha az `s`-től `v`-hez vezető úton negatív összsúlyú kör található, akkor definíció szerint `lru(s,v)` $= -\infty$. *Határozzuk meg egy adott pontból az összes többihez vezető legrövidebb utakat.* (A súly és hossz fogalmakat egymással párhuzamosan használjuk.)

Az Algoritmusok [2] című könyv azáltal próbál közös nevezőt keresni a legrövidebbút algoritmusok között, hogy mindeniket úgy mutatja be mint a fokozatos közelítés módszerének (lásd lennebb) különböző változatait. Ebben „comeniusi szellem" lelhető fel, hiszen e közös vonás jól felhasználható néhány alapvető hasonlóság, illetve különbség kiemeléséhez. Ezt a gondolatot szeretnénk tovább vinni ebben a cikkben, úgy mutatva be a fentebb megnevezett algoritmusokat, mint dinamikus programozást alkalmazó rokon stratégiákat. Úgy a fokozatos

közelítés módszere, mint a dinamikus programozás mögött, úgyanaz az alapelv húzódik meg, az optimalitás alapelve (lásd lennebb).

Először a fokozatos közelítés módszerét vázoljuk, azután röviden ismertetjük a három algoritmust, végül pedig rámutatunk azokra a hasonlóságokra és különbségekre, amelyek, mint dinamikus programozásos stratégiákat, jellemzik a tárgyalt algoritmusokat.

2. Fokozatos közelítés módszere

Ez a technika sajátosan használja ki az *optimalitás alapelvét*, amelyre különben a mohó és dinamikus programozás algoritmusok is épülnek. Íme az optimalitás alapelve legrövidebb út problémák esetén: *Egy legrövidebb út részútjai is legrövidebb utak.*

Következmény_1:

A kevesebb élből álló legkisebb súlyú utakból felépíthetők a több élű legkisebb súlyú utak. Mindhárom algoritmus ezt a megközelítést alkalmazza.

Következmény_2:

Az *s* pontból induló legrövidebb utak egy s gyökerű fát alkotnak (a *G* gráf egy feszítőfáját). Mindhárom algoritmus úgy határozza meg e legrövidebb utak fáját mint amely az *s* pontból nő ki. Ez azt jelenti, hogy a már meghatározott legrövidebb utak folyamatosan fát alkotnak. A fa fokozatosan épül fel élről élre. Minden él egy újabb csomóponttal bővíti a fát.

Következmény_3:

Ha `s`-ből `v`-be tartó legrövidebb úton `u` az utolsó előtti állomás, akkor

```
lru(s,v) = lru(s,u) + súly(u,v)
```

Következmény_4:

Ha `s` a kezdőcsúcs, akkor bármely `(u,v)` élre fennáll, hogy

```
lru(s,v) ≤ lru(s,u) + súly(u,v)
```

Fokozatos közelítés technikája: Minden `v` csúcsra nyilvántartunk egy `d[v]` értéket, amely felső korlátja az `s` kezdőcsúcsból a `v`-be vezető legrövidebb út súlyának. Kezdetben mindenik csúcs „távolságát” `s`-től ∞-re becsüljük, kivéve `s` „távolságát” `s`-től, amely nyilván 0. Egy `(u,v)` éllel való közelítés a következőképpen történik:

```
ha d[v] > d[u] + súly(u,v) akkor
   d[v] = d[u] + súly(u,v)
vége ha
```

Az alábbi algoritmusok addig végeznek az élekkel egymás után közelítéseket, míg a `d` tömb az `s` kezdőcsúcsból induló legrövidebb utak súlyait fogja tartalmazni.

Az algoritmusok csak abban különböznek, hogy hányszor és milyen sorrendben végeznek közelítéseket az egyes élekkel. A topologikus sorrenden alapuló algoritmus és a Dijkstra

algoritmus minden éllel legtöbb egyszer közelít. A Bellman-Ford algoritmus az egyes élekkel többször is közelít.

Nyilvánvaló, hogy egy adott `v` ponthoz vezető legrövidebb úton utolsó előtti pont csakis `v` valamelyik be-szomszédja lehet. Más szóval az `lru(s,v)` érték felépíthető a `v` be-szomszédjaihoz vezető legrövidebb utak hosszaiból, a `v` pont megfelelő be-élein való közelítések révén. Ezért a legrövidebb utak meghatározásához megfelelő lenne egy olyan pont-sorrend, amelyben a `v` pont `lru(s,v)` értékének kiszámítását megelőzi az összes be-szomszédjához vezető legrövidebb utak meghatározása. A pontok topologikus sorrendje (amennyiben létezik) éppen ezt a sorrendet jelenti.

Ha a gráf minden éle pozitív súlyú, akkor az is természetes, hogy az `lru(s,v)` érték kiszámításánál a `v` pontnak csak azokat a be-szomszédjait (`u`) kell figyelembe venni, amelyeknek `lru(s,u)` értéke kisebb mint `lru(s,v)`. Ez esetben hosszúságaik (súlyuk) szerint növekvő sorrendben fogjuk meghatározni a legrövidebb utakat. Erre a megfigyelésre alapszik a Dijkstra algoritmus.

3. Topologikus sorrenden alapuló „legrövidebb-út algoritmus”

Feltétel: A gráf legyen körmentes.

Az irányított körmentes gráfok esetén beszélhetünk a csomópontok topologikus sorrendjéről. A topologikus sorrendben minden pont megelőzi mindenik ki-szomszédját.

Stratégia, algoritmus: Kezdetben `d[i] = ∞` bármely $i = 1, ..., n$ pontra, kivéve az s kiindulási pontot, amelyre `d[s] = 0`. Azok a pontok, amelyek megelőzik `s`-t a topologikus sorrendben, nyilván nem érhetők el s-ből. Ezek d tömbbeli értéke végtelen marad. Az `s` pontot úgy tekinthetjük, mint amelyikhez megvan a legrövidebb út. Kezdve `s`-el, végigjárjuk a többi csomópontot topologikus sorrendben. Minden csomópont összes ki-éle segítségével megpróbálunk javítani a megfelelő ki-szomszédok `d` tömbbeli értékén. Amikor „megérkezünk” egy adott ponthoz, akkor már minden be-élén keresztül javítottuk (ha esedékes volt) a hozzá vezető utat. Ez azt jelenti, hogy amikor megérkezünk egy `v` ponthoz a `d[v]` tömb-elem már az `lru(s,v)` értéket tartalmazza.

Milyen sorrendben végez az alábbi algoritmus közelítéseket az egyes élekkel? A kezdőpontjaik topologikus sorrendje szerint! Ez az él-sorrend biztosítja, hogy a legrövidebb utakat alkotó élekkel olyan sorrendben történjenek a közelítések, amilyen sorrendben ezek az illető utakon előfordulnak. Ez magyarázza meg miért elég minden `(u,v)` éllel egyszer javítani a végpontjához vezető út hosszán.

4. Dijkstra algoritmusa

Feltétel: A gráf tartalmazhat kört, de ne tartalmazzon negatív súlyú élt.

Stratégia: Dijkstra algoritmusa súlyuk (hosszúságaik) szerint növekvő sorrendben határozza meg az egyes pontokhoz vezető legrövidebb utakat. Tegyük fel, hogy a `v` pont lesz az, amelyikhez a következő hosszúságú legrövidebb út vezet. Ha `u` az utolsó előtti állomás ezen az

úton, akkor u-hoz már meghatároztuk a legrövidebb utat. Következésképpen u már része a legrövidebb utak fájának. Tehát a következő legrövidebb út egy sajátossága, hogy az (s→u) szakasz már része a fának, az utolsó éle (az (u,v) él) pedig a fa valamelyik ki-éle lesz.

A fentiekkel összhangban a következő pont amelyikhez a legrövidebb út meghatározásra kerül, a legrövidebb utak fájának valamelyik ki-szomszédja lesz. Egészen pontosan a fa gyökeréhez legközelebb eső ki-szomszédja. E mohó választást a következő érvelés indokolja: mivel a fa többi ki-szomszédja mind távolabb esik a fa gyökerétől mint e legközelebbi, ezért azok nem képezhetik részét a legközelebbihez vezető legrövidebb útnak (fordítva viszont megtörténhet).

Milyen sorrendben közelít a Dijkstra algoritmus az (u,v) élekkel? A kezdőpontjaik lru(s,u) értéke szerinti növekvő sorrendben! Ez azt jelenti, hogy a Dijkstra algoritmus is tudja biztosítja, hogy a legrövidebb utakat alkotó élekkel olyan sorrendben történjenek a közelítések, amilyen sorrendben ezek az illető utakon előfordulnak.

5. Bellman-Ford algoritmus

Feltétel: A gráf ne tartalmazzon a kezdőpontból elérhető negatív összsúlyú kört. Ha van ilyen kör, az algoritmus jelzi, hogy nincs megoldás.

Stratégia, algoritmus: Jelen esetben a körök miatt a toplogikus sorrend nem értelmezhető, a negatív élek jelenléte pedig kizárja az *lru* értékek szerinti sorrendet, mint nem megfelelőt. Mi a megoldás? Mivel ez esetben nem ismert olyan él-sorrend, amely garantálná a legrövidebb utak egyetlen menetből való előállítását, szükségessé válhat az élek többszöri átpásztázása, újra és újra megpróbálva közelíteni velük. Mindezt addig ismételjük, amíg sikerül anélkül végigjárni az éleket, hogy sor került volna közelítésre.

A Bellman-Ford algoritmus az első menetben csak azokat a legrövidebb utakat határozza meg, amelyeknek esetében a választott él-sorrend biztosítja, hogy az útmenti élekkel az illető utakon való előfordulásuk sorrendjében történjen a közelítés. Ugyanakkor biztos az, hogy az első menetben meghatározásra kerülnek az egy él hosszú legrövidebb utak, a másodikban a két élből állók, és így tovább. Mivel egy legrövidebb út legfennebb $n - 1$ élből áll, így legfennebb $n - 1$ menetre lehet szükség. Ha a Bellman-Ford algoritmus az n-edik menetben is végez közelítő műveletet, akkor ez negatív összsúlyú kör jelenlétéről tanúskodik.

Egészen pontosan hány menetre van szüksége a Bellman-Ford algoritmusnak? Tekintsük azt az irányított gráfot, amelynek a csomópontjai a legrövidebb utak fájának élei (a faélek). E gráf két pontja között akkor van irányított él, ha a kezdőpont-él az eredeti gráf valamelyik legrövidebb útján megelőzi a végpont-élet. Világos, hogy az újonnan definiált gráf körmentes lesz. Ily módon beszélhetünk a csomópontjai toplogikus sorrendjéről. Ez viszont az eredeti gráf éleire vonatkoztatva egy olyan sorrendet jelent, amelyben minden él azok előtt szerepel, amelyeket megelőz a legrövidebb utakon. Bármely él-sorrend, amelyik megfelel az előbbi feltételnek, biztosítani tudja a legrövidebb utaknak egy menetből való meghatározását. Mivel a fenti feltétel mellett több helyes toplogikus él-sorrend is létezhet, ezért megtörténhet, hogy a topologikus sorrenden alapuló, illetve a Dijkstra algoritmusok különböző él-sorrendet generálnak. A Bellman-Ford algoritmusnak annyi menetre lesz szüksége amennyire „távol van” az általa választott él-sorrend egy helyes topologikus él-sorrendtől. A „távolság” alatt azt értjük, hogy hány él nincs a helyén. Pontosabban: Hány lépésből lehetne a választott sorrendből egy helyes topologikus

sorrendet előállítani úgy, hogy minden lépésben egy élet teszünk a helyére. Nevezzük ezeket *gát-él*eknek. A Bellman-Ford algoritmus első menete előállítja azokat a legrövidebb utakat, amelyek nem tartalmaznak gát-éleket, illetve azt amelyik utolsó élként tartalmazza az első gát-élet. A következő menet előállítja azokat a legrövidebb utakat, amelyeket csak az első gát-él „gátolt", beleértve azt is, amelyiken a második gát-él az utolsó él. Mivel minden menetben (kivéve az utolsót) egy gát-él „kerül a helyére" ezért eggyel több menetre lesz szükség mint ahány gát-él van. Bár ennyi menetből összeáll a legrövidebb utak fája, szükség van egy további menetre, hogy az algoritmus „tudomást szerezzen" erről a tényről.

6. Legrövidebb út algoritmusok és dinamikus programozás

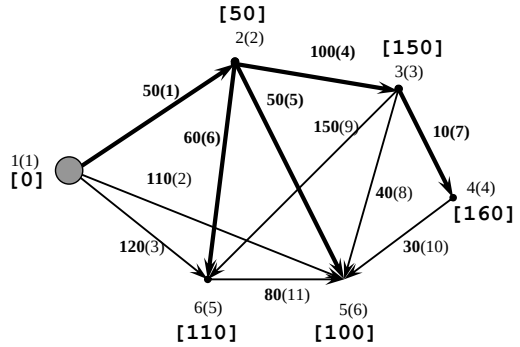
A dinamikus programozás az optimalitás alapelve implementációjának tekinthető. E stratégia lényege, hogy az egyszerűbb részfeladatok optimális megoldásaiból (indulva a triviálisan egyszerűktől) felépíti az egyre bonyolultabb részfeladatok optimális megoldásait (végül az eredeti feladatét). Egy másik jellegzetessége a dinamikus programozásnak, hogy a már megoldott részfeladatok optimális megoldásait (gyakran az ezt képviselő optimum értéket) eltárolja, hogy kéznél legyenek (amennyiben szükség lesz rájuk) az „építkezés" későbbi fázisaiban. [3,4,5]

A fentiekben bemutatott mindhárom algoritmus alapvetően dinamikus programozást alkalmaz. Mindhárom esetben igaz, hogy az élszámban rövidebb legkisebb súlyú utakból építettük fel a hosszabb legkisebb súlyú utakat. A soronkövetkező optimális út (amely meghatározásra került) mindig egy olyan út volt, amelyiknek az utolsó előtti állomásához vezető optimális út (és annak súlya) már rendelkezésre állt. Az algoritmusok ezt a lépést (a következő optimális út `lru` értékének kiszámítását) az illető út utolsó élével való közelítés révén valósították meg. Úgy is mondhatnánk, hogy a dinamikus programozásos stratégia implementálása nem jelent mást, mint megfelelő sorrendben közelíteni az optimális utak utolsó éleivel. A Bellman-Ford algoritmus bemutatásánál ezt a sorrendet a legkisebb súlyú utak fáját alkotó *élek* topologikus sorrendjeként azonosítottuk. A leghatékonyabb az lenne, ha csak ezzel az él-sorozattal közelítenénk (a többi éllel való közelítések szükségtelenek). Ennek lehetősége azért kizárt, mert feltételezi, hogy előre ismerjük az optimális utakat. Mivel a fölösleges közelítések, vagy közelítési próbálkozások, „nem ártanak", ezért a megoldást a gráf összes élének az átfésülése adja. Az első két algoritmus egyszer járja végig a gráf éleit. Mindkettőnek sikerül olyan él-sorrendben végezni a közelítéseket, hogy e közelítések részsorozataként megvalósuljon a dinamikus programozás szerinti építkezés. Arra a helyzetre amikor a gráf tartalmaz kört és negatív élet nem ismert olyan algoritmus, amelyik generálni tudna egyetlen megfelelő él-sorrendet. Ezért a Bellman-Ford algoritmus a gráf összes élének többszöri átfésülése által tudja csak biztosítani a dinamikus programozás igényelte közelítés sorozatot. Mindhárom algoritmus a *d* tömböt használja a már meghatározott optimális utak súlyainak eltárolására.

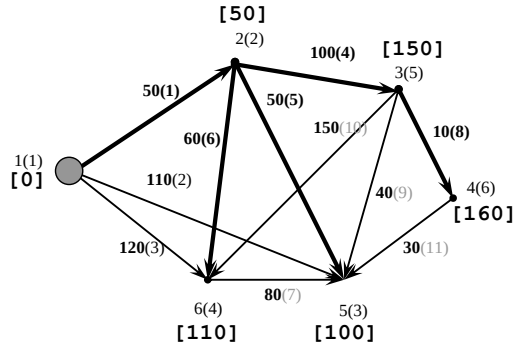
A mellékelt ábrák ugyanazon a gráfon (körmentes és nem tartalmaz negatív élet) mutatják be, hogy miként biztosítják a megvizsgált algoritmusok a dinamikus programozás előírta közelítés sorozatot. Az élek súlyai mellet az általuk végzett közelítési sorrend látható. Az első algoritmus esetében ez a sorrend a kezdőpontok topologikus sorrendjén alapszik. Dijkstra algoritmusa a kezdőpontok *lru* értéke szerinti sorrendet használja. Szürkével jelöltük az átugrott. A Bellman-Ford algoritmus bemutatásánál egy tetszőleges él-sorrendet választottunk. A dinamikus programozást megvalósító közelítésekhez tartozó élek esetén félkövér karakterekkel írtuk az illető művelet sorszámát. A Bellman-Ford algoritmus kapcsán azt is megjelöltük, hogy

hányadik menetben kerül sor az illető közelítésre. A csomópontok mellett zárójelben az látható, hogy milyen sorrendben kerülnek meghatározásra az illető pontokhoz vezető legrövidebb utak. Mindenik ábra alá leírtuk az ábrázolt közelítés sorozatot (kiemelve az optimális utakat felépítő közelítéseket). Az ábrák jól érzékeltetik, hogy

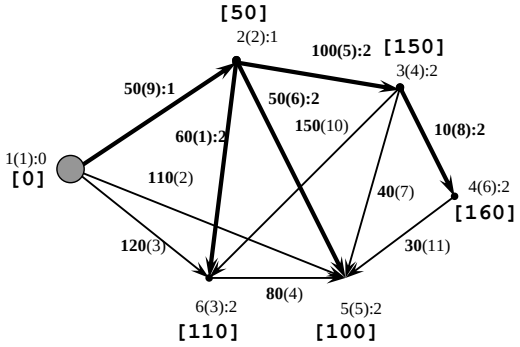
- az első két algoritmus generálta élsorozatok különbözhetnek.
- algoritmusokként különbözhet az optimális utakat kiépítő közelítések topologikus él-sorrendje (és ezzel együtt az optimális utak meghatározásának sorrendje).



(1,2), (1,5), (1,6), **(2,3)**, **(2,5)**, **(2,6)**, **(3,4)**, (3,5), (3,6), (4,5), (6,5)



(1,2), (1,5), (1,6), **(2,3)**, **(2,5)**, **(2,6)**, (6,5), **(3,4)**, (3,5), (3,6), (4,5)



1: (2,6), (1,5), (1,6), (6,5), (2,3), (2,5), (3,5), (3,4), **(1,2)**, (3,6), (4,5)

2: **(2,6)**, (1,5), (1,6), (6,5), **(2,3)**, **(2,5)**, (3,5), **(3,4)**, (1,2), (3,6), (4,5)

a.) Topologikus sorrenden alapuló optimális út algoritmus,

b) Dijkstra algoritmusa, c) Bellman-Ford algoritmus.

7. Következtetések helyett

A módszer másik didaktikai erőssége (a comeniusi alapelven túl), hogy az algoritmusok kulcs elemeit egymáshoz kapcsolva mutatja be, ami tartósabb mentális reprezentációt eredményez. Ez összhangban van azzal, ahogy a modern tudomány az emberi memóriát látja: „*A memóriát ne úgy képzeljük el, mint egy tartályt, ami fokozatosan megtelik. Inkább egy olyan fához lehetne hasonlítani, amelyen kampók nőnek. Ezekre a kampókra kerülnek az információk. Minden elraktározott információ újabb kampókat jelent, melyekre még több információ kapcsolódhat.*" [6]. Bár Russel ezen szavai elsősorban a memória növekedéséről szólnak, a szemléltetés azt is mutatja, milyen fontos az elsajátított dolgokat összekapcsolni az elmében. E módszer éppen ezt teszi lehetővé. A diákok elméjében a bemutatott három algoritmusról kialakult kép nemcsak átfogó, tiszta és mély lesz, hanem egyben tartósan eltárolt is.

Irodalomjegyzék

1. Comenius, Orbis sensualium pictus, 1653

2. T. H. Cormen, C. E. Leirserson, R. L. Rives, Introduction to Algorithms, by The Massachusetts Institute of Technology, 1990, 266-270, 287-289.

3. Kátai Z., Algoritmustervezés felülnézetből, Sapientia Kiadó, Kolozsvár, 2006

4. Kátai Z., "Upperview" algorithm design in teaching computer science in high schools, Teaching Mathematics and Computer Science, 3 (2005) 2

5. Kátai Z., Dynamic programming strategies on the decision tree hidden behind the optimizing problems, Informatics in Education, Institute of Mathematics and Informatics, Lithuania, 2006 (elfogadva)

6. P. Russel, The Brain Bokk, Dutton, New York, 1979