26. MINIMÁLIS KÖLTSÉGŰ UTAK MINDEN CSÚCSPÁRRA

Az előző két fejezetben tárgyalt feladat általánosításaként a gráfban található *összes csúcspárra* szeretnénk meghatározni a legkisebb költségű utat. A probléma gyakorlati előfordulására példa lehet az autós térképekben előforduló táblázat, amely a városok egymástól való legkisebb távolságait tartalmazza. Ez egy négyzetes táblázat, ahol a sorok és az oszlopok címkeként egyaránt a városok neveit viselik. A táblázat x címkéjű sorának és y címkéjű oszlopának a metszéspontjában található az y városnak az x várostól való legkisebb távolsága.

Modellezzük az autós térképet egy gráffal (irányított vagy irányítatlan, attól függően, hogy vannak-e egyirányú utak). A csúcsokat megfeleltetjük a városoknak, az élek pedig a városokat összekötő közvetlen utaknak. Az utak hossza legyen az élek költsége, tehát a gráf legyen élsúlyozott. Célunk a gráf alapján egy ilyen táblázat előállítása.

A csúcspárok közötti legkisebb költségű utakat megkereshetnénk az *előző* feladatnál látott megoldó *módszerek* segítségével. Minden csúcsot forrásként tekintve futtassuk le a „legrövidebb utak egy forrásból" algoritmusok egyikét.

1. Amennyiben az élsúlyok *nem-negatívak*, akkor alkalmazhatjuk a *Dijkstra-algoritmust*. Ekkor a műveletigény:
   a) Elsőbbségi sorként *rendezetlen tömböt* használva az utak költségértékeinek tárolására: $T(n) = n \cdot O(n^2) = O(n^3)$
   b) Az utak költségeinek elsőbbségi sorát *kupaccal* reprezentálva: $T(n) = n\, O\big((n+e)\log n\big) = O(n^2 \cdot \log n + n \cdot e \cdot \log n)$, ami ritka (illetve nem-sűrű) gráfokra $T(n) = O(n^2 \cdot \log n)$ futási időt eredményez.
2. Ha negatív élsúlyokat is megengedünk, akkor a *Bellman-Ford algoritmust* használhatjuk, amellyel a műveletigény $T(n) = n \cdot \Theta\big((n-1)\cdot e\big) = \Theta(n^2 \cdot e)$. Ez ritka gráfokra $T(n) = O(n^3)$, sűrű gráfokra $T(n) = \Theta(n^4)$ nagyságrendű lépésszámot jelent.

Ebben a fejezetben az *összes legrövidebb út* meghatározására – a megszorítások nélküli élsúlyok esetére – hatékonyabb eljárást adunk. Vizsgáljuk ennek az algoritmus speciális változatát *gráfok tranzitív lezártjának* a kiszámítására.

26.1. A Folyd algoritmus

Feladat. Adott egy $G = (V, E)$ élsúlyozott, irányított vagy irányítás nélküli, negatív összköltségű irányított kört nem tartalmazó véges gráf. Határozzuk meg $\forall\, u, v \in V$ csúcspárra az u-ból v-be vezető legkisebb költségű utat.

A fejezet további részében az utak hosszán az út mentén szereplő élek költségeinek az összegét értjük, a csúcspárok távolságán pedig a csúcspár közötti egyik legrövidebb út hosszát értjük. Tegyük fel, hogy $V = \{1, 2, \ldots, n\}$, és hogy a G gráf az C szomszédsági mátrixával adott. A csúcspárok távolságának a kiszámítására egy szintén $n \times n$-es D mátrixot fogunk használni.

Vezessünk be a következő fogalmat. Legyen egy $p = \langle v_1, \ldots, v_k \rangle$ egyszerű út *belső csúcsa* p minden v_1-től és v_k-tól különböző csúcsa, azaz $\langle v_2, \ldots, v_{k-1} \rangle$ halmaz elemei.

Az algoritmus elve az, hogy a megoldáshoz vezető n-lépéses iteráció során folyamatosan fenntartjuk a $D^{(k)}$ mátrixunkra a következő *invariáns tulajdonságot*.

Állítás. A k-adik iteráció lefutása után $\forall\ (i,j) \in V \times V$ csúcspárra $D^{(k)}[i,j]$ azon $i \rightsquigarrow j$ utak legrövidebbjeinek a hosszát tartalmazza, amelyek közbülső csúcsai k-nál nem nagyobb sorszámúak. Tehát $k = n$ esetén $\forall\ (i,j)$ csúcspárra $D^{(n)}[i,j]$ az $i \rightsquigarrow j$ utak legrövidebbjeinek a hosszát, azaz a feladat megoldását tartalmazza.

Bizonyítás. Az állítást k szerinti teljes indukcióval látjuk be.

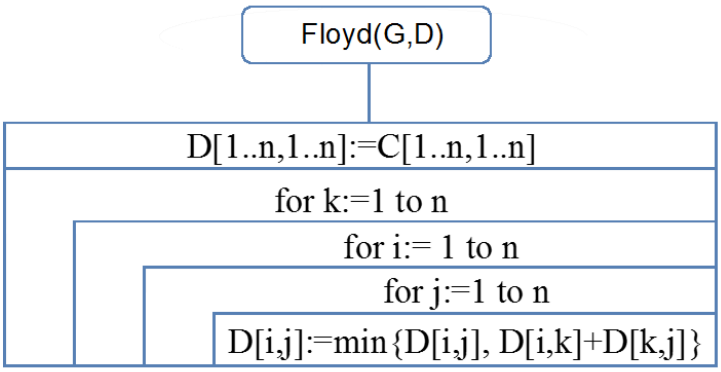
- $k = 0$: $\forall\ (i,j)$ csúcspárra $D^{(0)}[i,j]$ tartalmazza azon $i \rightsquigarrow j$ utak közül a legkisebb költségű utak hosszát, amely belső csúcsainak sorszáma kisebb, mint 1, azaz nem tartalmaznak belső csúcsot. Ami nem más, mint a C szomszédsági mátrixban szereplő érték. Tehát $D^{(0)}$ mátrix értéke legyen C szomszédsági mátrix.
- $k - 1 \rightarrow k$: a $D^{(k)}[i,j]$ értéket szeretnénk kiszámítani a $D^{(k-1)}$ mátrix értékeinek a felhasználásával. Két esetet különböztetünk meg aszerint, hogy $p^{(k)} = i \rightsquigarrow j$ (i-ből j-be vezető, belső csúcsként nem nagyobb, mint k sorszámú csúcsokat tartalmazó) egyik legrövidebb útnak, k belső csúcsa vagy sem. ($p^{(k)}$ út legyen egyszerű út, mert ha tartalmazna kört, és nem lehet negatív összköltségű a kör, akkor a kört "kivágva" a kapott út költsége nem nő, tehát a legrövidebb utak között vannak egyszerűek.)

  1. Ha k nem belső csúcsa $p^{(k)}$ útnak, akkor $p^{(k)}$ minden belső csúcsának sorszáma legfeljebb $k - 1$, azaz $p^{(k)}$ hossza azonos a legfeljebb $k - 1$ belső csúcsokat tartalmazó $i \rightsquigarrow j$ legrövidebb út hosszával $D^{(k-1)}[i,j]$-vel.
  2. Ha k belső csúcs a $p^{(k)}$ úton, akkor felbonthatjuk ${p_1}^{(k)} = i \rightsquigarrow k$ és ${p_2}^{(k)} = k \rightsquigarrow j$ legfeljebb $k - 1$ sorszámú belső csúcsokat tartalmazó i-ből k-ba ill. k-ból j-be vezető legrövidebb egyszerű utakra (legrövidebb út részútja is legrövidebb út). Tehát $D^{(k)}[i,j] = D^{(k-1)}[i,k] +\ D^{(k-1)}[k,j]$.

  Tehát a két eset közül az adja a rövidebb utat, ahol kisebb a számított érték, azaz a kérdéses legrövidebb út hossza megkapható az alábbi képlettel:

  $$D^{(k)}[i,j] = \min\{D^{(k-1)}[i,j]\,, D^{(k)}[i,j] = D^{(k-1)}[i,k] +\ D^{(k-1)}[k,j]\}$$

A fenti bizonyítás közvetlenül megadja az algoritmus lényegi lépését, $D^{(k)}$ előállítását. Mivel $D^{(k)}[i,k] = D^{(k-1)}[i,k]$ és $D^{(k)}[k,j] = D^{(k-1)}[k,j]$, így elegendő egyetlen D mátrix az algoritmus végrehajtásához. Az algoritmus a 26.1. ábrán látható.



26.1. ábra. A Floyd algoritmus

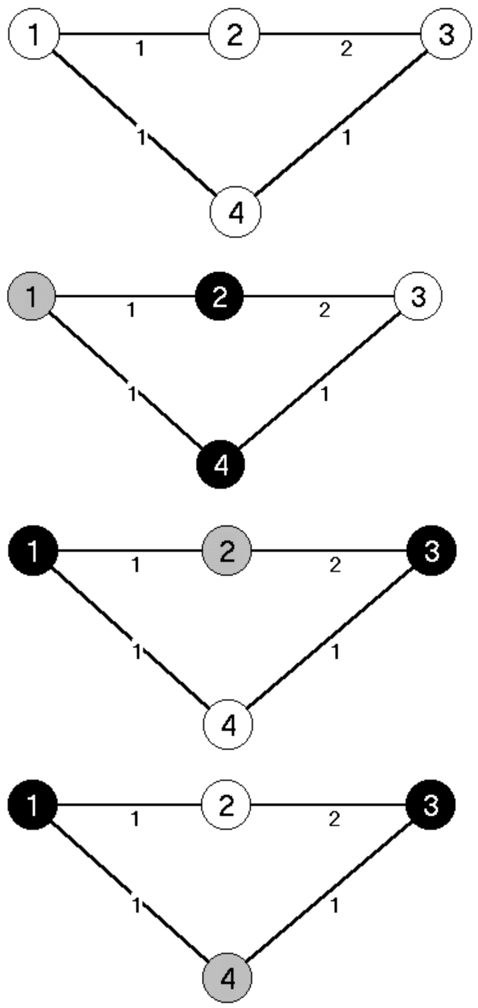
A Floyd algoritmust az ábrázolás szintjén adtuk meg mátrixos ábrázolás mellett, mint ahogy a szakirodalomban szokás.

26.2. Az algoritmus szemléltetése

A 26.2. ábra szemlélteti az algoritmus működését ADS szinten. Minden iteráció után megadjuk a gráf és a költségmátrix aktuális állapotát.

A kezdeti inicializáló lépés után D mátrix megegyezik a gráf csúcsmátrixának értékével.

Az első iteráció során, az 1-es csúcson átmenő utakkal próbáljuk javítani a mátrix értékeit. Amikor a 2-es csúcsból a 4-es csúcsba menő utakat vizsgáljuk, találunk az 1-esen átmenő javító utat, $D[2,4]$ értékét 2-re javítjuk. Mivel a gráf irányítatlan, így a szimmetrikus esetben is történik javítás ($D[4,2]$).



$$D^{(0)} = \begin{pmatrix} 0 & 1 & \infty & 1 \\ 1 & 0 & 2 & \infty \\ \infty & 2 & 0 & 1 \\ 1 & \infty & 1 & 0 \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 1 & \infty & 1 \\ 1 & 0 & 2 & 2 \\ \infty & 2 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 1 & 3 & 1 \\ 1 & 0 & 2 & 2 \\ 3 & 2 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 2 & 2 \\ 2 & 2 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

26.2. ábra. A Floyd algoritmus lépésenkénti végrehajtása

A második iterációs lépésben már olyan javító utakat keresünk, amelyek belső csúcsainak a sorszáma legfeljebb 2. Vizsgáljuk az legfeljebb 1-es sorszámú belső csúcsokat tartalmazó utakat (ill. a még nem létező utakat), és megpróbáljuk közbülső csúcsnak beilleszteni a 2-es csúcsot. Az 1-ből a 3-ba, ill. 3-ból az 1-be találtunk javító utat (az eddig nem létező úthoz, a ∞ hosszú úthoz képest) a 2-esen át.

A harmadik iterációban nem találunk a 3-as csúcson átmenő javító utakat, így a mátrix nem változik.

A negyedik iterációs lépésben már olyan javító utakat keresünk, amelyek belső csúcsainak a sorszáma legfeljebb 4. Vizsgáljuk a legfeljebb 3-as sorszámú belső csúcsokat tartalmazó utakat (ill. a még nem létező utakat), és megpróbálunk a 4-es csúcson átmenő, kisebb költségű "elkerülő" utat találni. Találunk a 4-es csúcson átmenő javító utat.

Eddig az 1-esből a 3-mas csúcsba vezető, legfeljebb 3-as sorszámú belső csúcsokat tartalmazó legrövidebb út hossza 3 volt. Ez az út az $\langle 1,2,3\rangle$. Most megengedjük, hogy belső pont sorszáma lehet 4 is, így megvizsgálva az $\langle 1,4\rangle$ ill. $\langle 4,3\rangle$ részutak hosszának összegét, az kevesebb mint, az $\langle 1,2,3\rangle$ út hossza, tehát találtunk egy kisebb költségű elkerülő utat. Természetesen ez a szimmetrikus esetre is igaz. A 4. lépés után megkapjuk a végeredményt.

Az algoritmus n iterációs lépésben, az n^2-es mátrix minden elemére konstans számú műveletet végez, így $T(n) = \Theta(n^3)$. Ez egy *stabil algoritmus*, mivel legjobb, legrosszabb és átlagos esetben is azonos a műveletigénye.

A műveletigényünk tehát nagyságrendileg ugyanannyi, mintha a Dijkstra algoritmust csúcsmátrixos ábrázolású gráfon, prioritásos sorként rendezetlen tömböt használva, minden csúcsra, mint forrásra lefuttatnánk. A Dijkstra algoritmusnál már láttuk, hogy ezt a megvalósítást sűrű gráfok esetén célszerű alkalmazni. Ritka gráfok esetén éllistás ábrázolást használhatunk, prioritásos sorként pedig kupacot, így a műveletigény $T(n) = O(n^2 \cdot \log n)$, ami jobb, mint a Floyd algoritmus műveletigénye. Úgy tűnhet, hogy felesleges a Floyd algoritmust használni, mivel a Dijkstra algoritmus jobb eredményt ad, vegyük azonban figyelembe, hogy a Dijkstra algoritmust csak nem negatív költségű élek esetén használhatjuk.

Amennyiben előfordulhat a gráfban negatív súlyú él (de negatív összköltségű kör nem), a Bellman-Ford algoritmus használatával ritka gráfokra $T(n) = O(n^3)$, sűrű gráfokra $T(n) = O(n^3)$ műveletigénnyel tudjuk megoldani a feladatot. Látható, hogy sűrű gráfok esetén a Floyd algoritmus hatékonyabb.

Amennyiben a csúcspárok közötti legrövidebb utakra is kíváncsiak vagyunk (és nem csak azok hosszára), a korábban már látott módon eltárolhatjuk a megelőző (vagy közbülső k címkéjű) csúcsot. Mivel most csúcspárok közötti utakról van szó minden lehetséges csúcspárra ($n \cdot n$ csúcspár), így érdemes mátrixot használni.

26.3. Tranzitív lezárt

Feladat. Adott egy $G = (V, E)$ irányított vagy irányítás nélküli, súlyozatlan, véges gráf. Határozzuk meg $\forall\, u, v \in V$ csúcsra, hogy létezik-e u-ból v-be vezető út a gráfban.

A fejezet további részében a gráfunk legyen véges, súlyozatlan, irányított vagy irányítatlan, az utak hosszán pedig az út mentén található élek számát értjük.

Állítás. Legyen a G gráf szomszédsági mátrixa $C[1 \ldots n, 1 \ldots n]$. Ekkor $C^k[i,j]$ $(1 \le i,j \le n, k \in \mathbb{N})$ az i-ből a j-be vezető k hosszúságú utak számát adja meg.

Bizonyítás. k szerinti teljes indukcióval.

- $k = 1$: Az 1 hosszú út egy élnek felel meg. Tehát $C^1[i,j]$ az i-ből a j-be menő élek számát adja meg, (ami megállapodás szerint legfeljebb 1 lehet), ez pedig pontosan a szomszédsági mátrix definíciója.
- $k-1 \to k$: Tegyük fel, hogy $k-1$-ig teljesül az állítás. Kérdés, hányféleképpen juthatok el k hosszú úton i-ből j-be? Az $i \rightsquigarrow j$ k hosszúságú utat a következő módon tudjuk felbontani: $i \rightsquigarrow h$ $k-1$ hosszú út, majd $h \to j$ él. Kérdés az, hogy hányféleképpen juthatok el k hosszú úton i-ből j-be úgy, hogy a j-t megelőző pont a gráfban a h csúcs.

Az indukciós feltevés szerint, $C^{k-1}[i,h]$ féle módon juthatunkk el $k-1$ hosszú úton i-ből h-ba. Továbbá, ha létezik $h \to j$ él a gráfban ($C[h,j]=1$), akkor azon már csak egyféleképpen tudunk tovább menni j-be, azaz $C^{k-1}[i,h] \cdot C[h,j]$ megadja az i-ből a j-be menő k hosszú utak számát, ahol j előtti megelőző csúcs a h. Amennyiben nem létezik $h \to j$ él, akkor a szorzat értéke 0, ami kifejezi, hogy nincs ilyen út a gráfban.

Az előzőekben h-n átmenő utakat számláltunk, de s tetszőleges csúcsa lehet a gráfnak, így ezeket minden $\forall h \in V$ csúcsra összegezzük:

$$C^k[i,j] = \sum_{i=1}^{n} \left(C^{k-1}[i,h] \cdot C[h,j]\right)$$

ahol $i,j \in [1 \dots n]$, ami nem más, mint egy mátrix szorzat: $C^k = C^{k-1} \cdot C$.

Következmény. Legyen $S^k = C + C^2 + C^3 + \cdots + C^k$ mátrix. Ekkor $S^k[i,j]$ eleme az i-ből j-be vezető legfeljebb k hosszúságú utak számát adja meg.

Definíció. A $G=(V,E)$ véges gráf *útmátrixa*, vagy *elérhetőségi mátrixa*:

$$U[i,j] = \begin{cases} 1 & \text{ha } \exists\ i \rightsquigarrow j \text{ út a gráfban} \\ 0 & \text{különben} \end{cases}$$

ahol $i,j \in [1 \dots n]$, $n = |V|$.

Állítás. Ha létezik i-ből j-be vezető út a gráfban, akkor létezik i-ből j-be vezető egyszerű út is, továbbá minden egyszerű út legfeljebb $n-1$ hosszú, egy egyszerű kör pedig legfeljebb n hosszú. Tehát az *útmátrix előállításához* számoljuk ki $C + C^2 + C^3 + \cdots + C^k$ összeget, és az eredménymátrixban a nullánál nagyobb elemeket írjuk át 1-esre.

Definíció. Egy $G=(V,E)$ gráf *tranzitív lezárása* $G'=(V',E')$ gráf, ahol $V'=V$ és $(u,v) \in E' \Leftrightarrow \exists\ u \rightsquigarrow v$ út a gráfban.

Állítás. G útmátrixa G' szomszédsági mátrixa.

Állítás. G erősen összefüggő $\Leftrightarrow$ U-ban nincs nulla elem $\Leftrightarrow$ G' teljes gráf.

26.4. A Warshall algoritmus

Az előző fejezetben egy gráf tranzitív lezártját elő tudtuk állítani a szomszédsági mátrix hatványainak az összegeként, amelynek hatékonysága $T(n) = \Theta(n^4)$. Most lássunk egy nagyságrenddel hatékonyabb módszert.

A tranzitív lezárt meghatározására használhatnánk a Floyd algoritmust is, hiszen az algoritmus lefutása után, ha $D[i,j]$ véges, akkor létezik $i \rightsquigarrow j$ út, ha végtelen, akkor pedig nincs i-ből j-be vezető út. Azonban a Floyd algoritmus elvét felhasználva, szép algoritmus adható az ábrázolás szintjén a problémára (bár S. Warshall nevéhez fűződő algoritmus megelőzte Floydét).

Adott a $G=(V,E)$ véges, súlyozatlan, irányított vagy irányítatlan gráf. A Floyd algoritmushoz képest az alábbi változtatások után megkapjuk a Warshall algoritmust:

1. A W logikai értékekből álló mátrix (a Floyd-nál D-vel jelölt mátrix) kezdeti értéke legyen

$$U[i,j] = \begin{cases} \uparrow & \text{ha } i=j, \text{vagy (i,j)} \in \text{E} \\ \downarrow & \text{különben} \end{cases}$$

ahol $i,j \in [1 \dots n]$, $n = |V|$.

2. A ciklusban végzett művelet pedig legyen a következő:

$$W[i,j] := W[i,j] \vee (W[i,k] \wedge W[k,j])$$

Az algoritmus helyességének a belátása hasonlóan történhet, mint a Floyd algoritmusnál.

Természetesen a *műveletigény* is aszimptotikusan megegyezik a Floyd algoritmus műveletigényével, azzal a különbséggel, hogy a Floyd algoritmusnál, a ciklus belsejében konstans műveletnek tekintett összeadás és minimumválasztás helyett, most logikai műveleteket végzünk, ami hatékonyabb lehet.