24. MINIMÁLIS KÖLTSÉGŰ UTAK I.

Az *útvonaltervezés* az egyik leggyakrabban végrehajtott eljárása a gráfok alkalmazásai körében. A feladat például a közlekedésben jelentkezik. A gráfot itt az a térkép jelenti, amely tartalmazza a kiindulási pontot és a célállomást, amellett a szóba jövő utak sem futnak le róla.

Az egyes útszakaszok megtételéhez (például üzemanyag és idő) ráfordítás szükséges, ezért olyan utat szeretnénk ezekből a szakaszokból összeállítani, amelynek a *költsége* minél kisebb, esetleg *minimális*. A gráf éleihez tehát *nemnegatív költségértékeket* rendelünk. Az útszakaszok lehetnek mindkét irányban járhatók, ha például a navigációs probléma az ország térképén két város között merül fel, illetve lehetnek egyirányúak, ha egy városon belül keresünk legrövidebb utat két pont között. Ennek megfelelően a gráf egyaránt lehet *irányítás nélküli*, illetve *irányított*.

A gráfon megfogalmazott, minimális költségű útvonal keresése során „alapesetben" nem támaszkodhatunk más ismeretre, mint az élsúlyok értékére. Nincs olyan háttértudásunk, amely eleve kizárná bármelyik csúcsot az útvonalból. A feladatot ezért az általános esetben úgy fogalmazzuk meg, hogy egy kiinduló csúcsból a gráf *minden* pontjához keressük az oda vezető minimális költségű utat.

A kitűzött feladat megoldására *Dijkstra* nevezetes algoritmusát ismertetjük. Ez egy *mohó* eljárás, amely a *startcsúcsból* kiindulva, minden lépésben az *addigi legkisebb költséggel* elérhető csúcsot választja ki, és annak szomszédjaiba új költségeket számol. (Könnyen felismerhetjük majd, hogy a *szélességi bejárásnál* – egyszerűbb esetben – találkoztunk már ennek a megoldásnak az elvével.)

Az eljárás ismertetésénél újra hasznát vesszük annak, hogy megkülönböztetjük az *absztrakt adatszerkezet*, illetve az adatok *reprezentálásnak* a szintjét. Az ADS szinten megadott algoritmus nem csak szemléletes, hanem a megengedhető mértékig nem-determinisztikus, valamint az eltérő reprezentációk közös kiindulópontját képezi. Adatábrázolás szintjén *mindkét gráf-reprezentációra*, valamint az algoritmusban szereplő *elsőbbségi sor két megvalósítására* meggondoljuk a *műveletigényét*.

24.1. A minimális költségű utak problémája

A bevezetőben körvonalazott problémát pontosabban is megfogalmazzuk, mint gráfon értelmezett feladatot.

Feladat. Adott egy $G = (V, E)$ élsúlyozott, véges gráf és egy $s \in V$ csúcs, a kezdőcsúcs. A gráf élei lehetnek irányítatlanok, és rendelkezhetnek irányítással. Szeretnénk meghatározni minden $v \in V$ csúcsra az s-ből v-be vezető legkisebb költségű utat, a költségértékével együtt.

Elterjedt az a szóhasználat, amely a *kezdőcsúcsot startcsúcsnak* vagy *forrásnak* nevezi, a *minimális költségű* utat pedig *optimális* útnak, vagy (kissé megtévesztően) *„legrövidebb"* útnak mondja, amit mindjárt meg is indokolunk.

Módosítsuk a szélességi bejárásnál bevezetett *úthossz* fogalmunkat. Legyen $G = (V, E)$ élsúlyozott, irányított vagy irányítás nélküli gráf $c: V \times V \to \mathbb{R}$ *súlyfüggvénnyel*. A $v_0 \rightsquigarrow v_k = \langle v_0, v_1, \ldots, v_k \rangle$ út *költsége* (*összsúlya*, *hossza*) az utat alkotó élek súlyainak az *összege*, azaz

$$d(p) = \begin{cases} 0 & \text{ha } k = 0 \\ \sum_{i=1}^{k} c(v_{i-1}, v_i) & \text{egyébként} \end{cases}$$

Vezessük be a *legkisebb költségű* (minimális költségű, optimális vagy legrövidebb) út fogalmát. Az u-ból a v-be $(u, v \in V)$ vezető minimális költségű út *súlya* (összköltsége, hossza) legyen

$$\delta(u,v) = \begin{cases} \min\{d(u \rightsquigarrow v)\} & \text{ha } \exists u \rightsquigarrow v \text{ út a gráfban} \\ \infty & \text{különben} \end{cases}$$

Az u csúcsból a v-be vezető legrövidebb úton a $\delta(u,v)$ súlyú utak egyikét értjük.

A *szélességi keresésnél* már találkoztunk hasonló feladattal, az ottani legrövidebb (legkisebb élszámú) utak nyilvántartásával. Most is ugyanúgy járunk el, egy $P[1 \ldots n]$ tömbben tartjuk nyilván minden csúcsnak a *megelőzőjét*, az algoritmus által talált (egyik) legrövidebb úton.

A szélességi fához hasonlóan definiálhatjuk a *legrövidebb utak feszítőfáját* is. Az $F(V', E')$ gráf a $G = (V, E)$ gráfban a legrövidebb utak fája, ha V' elemei az s-ből elérhető csúcsok, $E' \subseteq E$ és minden $v \in V'$csúcsra pontosan egy egyszerű út vezet s-ből v-be, és ez az út egyike az s-ből v-be vezető legrövidebb G-beli utaknak.

24.2. A megoldás módszere

A *megoldás elve* a következő. Minden lépésben tartsuk nyilván az összes csúcsra, a forrástól az illető csúcsba vezető, eddig talált legkisebb összsúlyú utat. A már megismert módon a $d[1..n]$ tömbben a távolságot, és $P[1..n]$ tömbben a megelőző csúcsot tároljuk.

Azt mondhatjuk, hogy egy v ∈ V csúcs már KÉSZ, ha ismert a hozzá vezető legrövidebb út. Kezdetben egyetlen csúcs sem KÉSZ, és azt szeretnénk elérni, hogy az eljárás először magát az s kezdőcsúcsot válassza ki, hiszen innen indul a keresés és formálisan nézve az önmagából hozzá vezető legrövidebb út nulla költsége eleve ismert.

1. Kezdetben a forrástól vett távolság legyen a kezdőcsúcsra 0, a többi csúcsra ∞.
2. Minden lépésben a nem KÉSZ csúcsok közül tekintsük az egyik legkisebb eddigi költségű *v* csúcsot. A v-t *terjesszük ki*, azaz a szomszédjaira számítsuk ki a v-be vezető, és onnan egy kimenő éllel meghosszabbított út költségét. Amennyiben ez jobb (kisebb), mint az illető szomszédba eddig talált legrövidebb út összsúlya, akkor innen kezdve ezt az utat tekintsük az adott szomszédba vezető, eddig talált legrövidebb útnak. Ezt az eljárást szokás *közelítés*nek is nevezni.

24.3. Az eljárás helyessége

Az algoritmus helyességét több lépésben igazoljuk.

1. *Állítás*. A $v_0 \in V$ csúcsból a $v_k \in V$ csúcsba vezető, bármely legrövidebb $\langle v_0, \ldots, v_{k-1}, v_k \rangle$ út olyan, hogy a $\langle v_0, \ldots, v_{k-1} \rangle$ út is egyike a v_0-ból a v_{k-1}-be vezető legrövidebb utaknak.

   Bizonyítás. Az úthossz definíciójából tudjuk, hogy $d(\langle v_0, \ldots, v_{k-1}, v_k \rangle) = d(\langle v_0, \ldots, v_{k-1} \rangle) + c(v_{k-1}, v_k)$. Indirekt tegyük fel, hogy létezik $q = \langle v_0, \ldots, v_{k-1} \rangle$ útnál rövidebb p út v_{k-1}-be. Ekkor ezen az úton eljutva v_{k-1}-be az úthossz: $d(p) + c(v_{k-1}, v_k) < d(q) + c(v_{k-1}, v_k)$, mivel $d(p) < d(q)$. Tehát találtunk a legrövidebb útnál rövidebb utat, ami ellentmondás.
2. Az előző állításból következik, elegendő a legrövidebb úton csak a megelőző csúcsot eltárolni. Az állítás könnyen általánosítható "a legrövidebb út részútja is legrövidebb út" állításra.

3. *Állítás*. $\forall\langle v_0, \ldots, v_k\rangle$ úton, a $d(v_0 \rightsquigarrow v_0)$, $d(v_0 \rightsquigarrow v_1)$, …, $d(v_0 \rightsquigarrow v_k)$ részutak költségei monoton növő sorozatot alkotnak.

   Bizonyítás. Mivel nincsenek negatív élsúlyok, ezért $d(v_0 \rightsquigarrow v_i)$, $i = 0 \ldots k$ nem negatív tagú sorozatnak tekinthető.

4. *Állítás*. Az egyes lépések után, $\forall\, u \in V\backslash$KÉSZ csúcs esetén, ha $\exists\, s \rightsquigarrow u$ út a gráfban, akkor az u csúcshoz vezető legrövidebb úton $\exists\, v \in$ KÉSZ csúcs.

   Bizonyítás. Az $s \in$ KÉSZ biztosan teljesül.

5. *Állítás*. Minden lépésben, $\forall\, u \in V\backslash$KÉSZ csúcsra teljesül $d_{min} \leq \delta(s,u)$, azaz s-ből u-ba vezető utak távolsága nem csökkenhet a jelenlegi minimum alá.

   Bizonyítás. Az első lépésben triviálisan teljesül az állítás, mivel $d_{min} = 0$ és nincs negatív élsúly. Indirekt tegyük fel, hogy $\exists\, u \in V\backslash$KÉSZ, amelyre az eddig talált legrövidebb p út $d(p) \geq d_{min}$, de a legrövidebb $p^* = s \rightsquigarrow u$ útra $d_{min} > d(p^*)$. Tudjuk, hogy van KÉSZ csúcsa a p^* útnak (2. lépéstől kezdve). Legyenek $v \in$ KÉSZ és $w \in$ Szomszéd$(v)\backslash$KÉSZ egymást követő csúcsai p^*-nak (biztos létezik w, mivel $u \in V\backslash$KÉSZ, legfeljebb $w = u$). Mivel $v \in$ KÉSZ, így már ismert v-be vezető egyik legrövidebb út, ami része az u-ba menő egyik legrövidebb útnak. Legyen u-ba menő egyik legrövidebb útnak w-ig tartó részútja ${p^*}_w = s \rightsquigarrow v \rightarrow w$, aminek a hossza már ismert, mivel v-t már KÉSZ-nek választottuk, ezért ${p^*}_v$ ismert, továbbá v-t kiterjesztettük, így ${p^*}_w$ is ismert. Azt is tudjuk, hogy $d_{min} \leq d({p^*}_w)$, mivel $w \notin$ KÉSZ. A monotonitás tulajdonság miatt $d({p^*}_w) \leq d(p^*) \implies d_{min} \leq d({p^*}_w) \leq d(p^*)$, ami ellentmond az indirekt feltevésnek.

6. *Állítás*. Az egyes lépésekben az algoritmus által KÉSZ-nek kiválasztott $u \in V$ csúcsra valóban ismert az egyik legrövidebb $s \rightsquigarrow u$ út.

   Bizonyítás. Legyen $p = s \rightsquigarrow u$ a jelenleg ismert legrövidebb út (ami lehet ∞ távolságú is), amelyre tudjuk $d(p) = d_{min}$, és indirekt tegyük fel, hogy $\exists\, p^* = s \rightsquigarrow u$ út, amelyre $d(p^*) < d(p)$. Tudjuk, hogy $\exists\, v \in V\backslash$KÉSZ és $w \in$ Szomszéd$(v)\backslash$KÉSZ csúcs a p^* úton. A korábbiakban látott módon levezethető, hogy $d(p) = d_{min} \leq d(p^*_w) \leq d(p^*)$ ami ellentmond az indirekt feltevésnek. Tehát rövidebb $s \rightsquigarrow v$ utat a későbbiekben sem találhatunk.

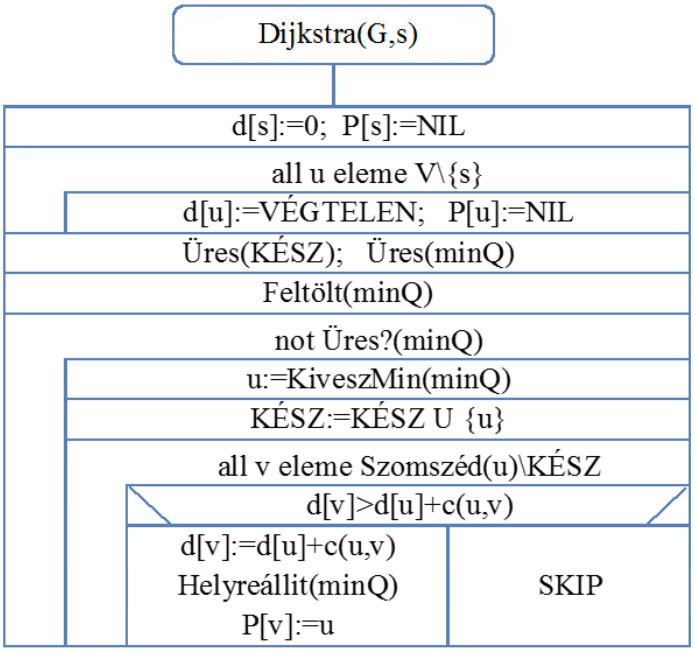
7. *Állítás*. A fenti algoritmus, negatív élsúlyokat nem tartalmazó $G = (V, E)$ véges gráf esetén, $s \in V$ forrás (kezdőcsúcs) és $\forall v \in V$ csúcsra, meghatározza s-ből v-be vezető legrövidebb utat és annak hosszát.

   Bizonyítás. Az algoritmus minden lépésben KÉSZ-nek választ egy csúcsot. Mivel véges sok csúcsa van a gráfnak, az algoritmus véges időn belül terminál, és $\forall v \in V$ csúcs KÉSZ-en van, azaz ismert a legrövidebb $s \rightsquigarrow v$ út.

24.4. Dijkstra algoritmusa

A $d[1..n]$ és $P[1..n]$ tömböket, a korábban ismertetett módon, a távolság és a megelőző csúcs nyilvántartására használjuk. A KÉSZ halmazba rakjuk azokat a csúcsokat, amelyekhez már ismerjük az egyik legrövidebb utat. Ezen kívül, használunk egy *minimumválasztó elsőbbségi (prioritásos) sort* ($minQ$), amelyben a csúcsokat tároljuk a már felfedezett, legrövidebb $d(s \rightsquigarrow u)$ távolsággal, mint kulcs értékkel.

A *Dijkstra algoritmusa* a 24.1. ábrán látható.



Műveletigény

rendezettlen tömb	kupac
1	
"(n-1)-szer"	
n-1	
1	
0	n
"n-szer"	
n*n	n*logn
n*1	
"e-szer"	
e*1	e*logn

24.1. ábra. A Dijkstra algoritmus és műveletigénye

24.5. Az algoritmus megvalósítása reprezentációs szinten

Vizsgáljuk meg a prioritásos sor ($minQ$) megvalósításának két, természetes módon adódó lehetőségét.

1. A prioritásos sort valósítsuk meg rendezetlen tömbbel, azaz a prioritásos sor legyen maga a $d[1..n]$ tömb. Ekkor a minimum kiválasztására egy *feltételes minimum keresés*t kell alkalmazni, amelynek a műveletigénye $\Theta(n)$. A Feltölt($minQ$) és a Helyreállít($minQ$) absztrakt műveletek megvalósítása pedig egy SKIP-pel történik.
2. *Kupac adatszerkezet* használatával is reprezentálhatjuk a prioritásos sort. Ekkor a Feltölt($minQ$) eljárás, egy kezdeti kupacot épít, amelynek a műveletigénye *lineáris* (lásd: 16. fejezet). Azonban most a d[1..n] tömb változása esetén a kupacot is karban kell tartani, mivel a kulcs érték változik. Ezt a Helyreállít($minQ$) eljárás teszi meg, amely a csúcsot a gyökér felé "szivárogtatja" fel, ha szükséges (mivel a kulcs értékek csak csökkenhetnek). Ennek a műveletigénye $\Theta(\log n)$-es.

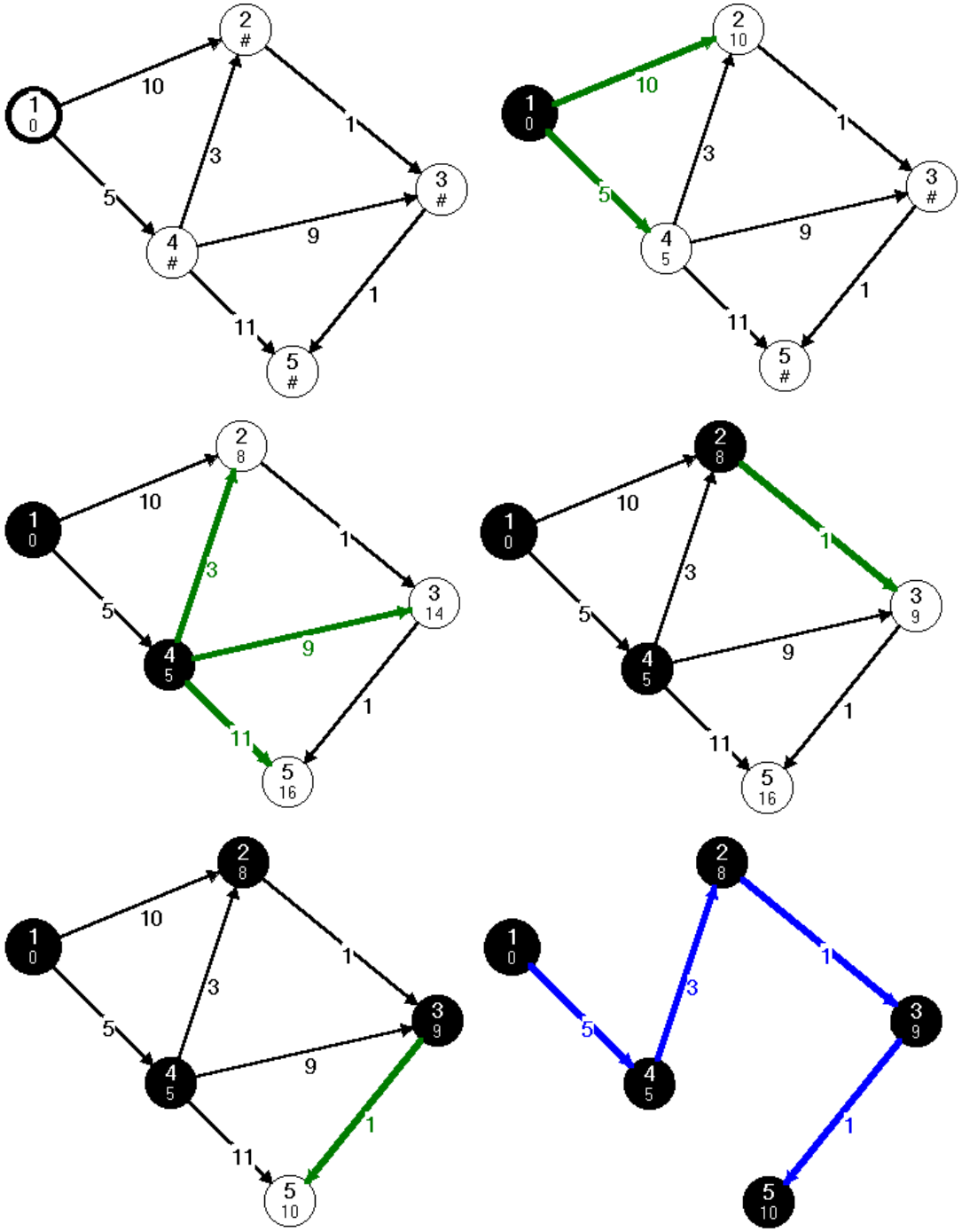
Megjegyzés. Nem szükséges kezdeti kupacot építeni, felesleges a kupacba rakni a végtelen távolságú elemeket. Kezdetben csak a kezdőcsúcs legyen a kupacban, majd amikor először elérünk egy csúcsot és a távolsága már nem végtelen, elég akkor berakni a kupacba.

24.6. Az algoritmus szemléltetése

A 24.2. ábrán megfigyelhető Dijkstra algoritmusának működése lépésenként. A KÉSZ halmazhoz való tartozást színezéssel valósítjuk meg. Legyenek a nem KÉSZ csúcsok fehérek, a KÉSZ csúcsok pedig fekete színűek. A csúcsokra a címkén kívül, felírtuk az eddig talált legrövidebb út hosszát is (d tömbbeli értékeket). A végtelen nagy távolságot jelöljük '#' jellel. A forrás legyen az 1-es címkéjű csúcs.

Az inicializáló lépés után a kezdőcsúcs 0, a többi csúcs végtelen súllyal szerepel az elsőbbségi sorban. Az első lépésben kivesszük a prioritásos sorból az 1-es csúcsot (mivel az ő prioritása a legkisebb). Az 1-es csúcshoz már ki van számítva a legrövidebb út, tehát ez a csúcs már elkészült, színezzük feketére. Kiterjesztjük az 1-est, azaz a szomszédjaira

kiszámítjuk az 1-esből kimenő éllel meghosszabbított utat. Ha ez javító él, azaz az 1-esen átmenő út rövidebb, mint az adott szomszédba eddig talált legrövidebb út, akkor a szomszédban ezt feljegyezzük (d és P tömbbe). Az ábrán kiemeltük a javító éleket. Megfigyelhető, hogy a 2-es csúcsba már korábban is találtunk 10 hosszú utat $\langle 1, 2 \rangle$, de a második lépésben, a 4-es csúcs kiterjesztésekor, találunk, a 4-es csúcson átmenő rövidebb 8 hosszú utat. A 2-es csúcs kiterjesztésekor a 3-as csúcsba találtunk egy rövidebb utat. A negyedik lépésben még találunk rövidebb utat az 5-ös csúcsba, majd az utolsó lépésben kivesszük a prioritásos sorból az 5-ös címkéjű csúcsot is. Az utolsó lépésben berajzoltuk a legrövidebb utak fáját alkotó éleket.



24.2. ábra. A Dijkstra algoritmus lépésenkénti végrehajtása

24.7. Műveletigény

A prioritásos sor fenti két megvalósítása esetén, a következőképpen alakul az algoritmus műveletigénye. A 24.1-es ábrán feltüntettük az egyes műveletek költségét a két ábrázolás estén. A belső ciklust célszerű globálisan kezelni, ekkor mondható, hogy összesen legfeljebb annyiszor fut le, ahány éle van a gráfnak.

1. Rendezetlen tömb esetén:
$$T(n) = O(1 + n - 1 + 1 + 0 + n^2 + n + e) = O(n^2 + e) = O(n^2)$$
2. Kupac esetén:
$$T(n) = O(1 + n - 1 + 1 + n + n \cdot \log n + n + e \cdot \log n) = O\big((n + e) \cdot \log n\big)$$

Rendezetlen tömbbel való ábrázolás műveletigénye csak a csúcsok számától függ, míg a kupacos ábrázolás műveletigénye, az élek számának is a függvénye. Sűrű gráfnak nevezzük az olyan gráfokat, amelyre $e \approx n^2$, ritka gráfoknak pedig, amelyre $e \approx n$ (vagy "kevesebb"). Tehát a kupacos ábrázolás műveletigénye ritka gráf esetén $O(n \cdot \log n)$, míg sűrű gráf esetén $O(n^2 \cdot \log n)$. Az az érdekes helyzet adódott, hogy a gráf sűrűsége befolyásolja milyen ábrázolást érdemes választani. A kupac, csak a ritka gráfok esetén hatékonyabb, míg sűrű gráfok esetén a rendezetlen tömbbel való reprezentáció az olcsóbb.

Tehát a reprezentáció szintjén sűrű gráf esetén *csúcsmátrix és rendezetlen tömb*, titka gráf esetén *szomszédsági lista és kupac* javasolt.

24.8. Megjegyzések

A *mohó algoritmus* mindig az adott lépésben optimálisnak látszó döntést hozza, vagyis a lokális optimumot választja abban a reményben, hogy ez globális optimumhoz fog majd vezetni. Dijkstra algoritmusa is mohó stratégiát követ, amikor minden lépésben KÉSZ-nek választ egy csúcsot. Mivel a legrövidebb út részútja is legrövidebb út, így a lokális optimumok választásával elérhetjük a globális optimumot.

Most vizsgáljuk meg a szélességi keresés és a Dijkstra algoritmus kapcsolatát. Mindkét algoritmusnál, egy kezdőcsúcsból kiinduló legrövidebb utakat állítunk elő, csak a Dijkstra algoritmusnál az utak hosszának fogalmát általánosítjuk. Legyen minden él súlya egységnyi, ekkor a Dijkstra algoritmus egy szélességi keresést hajt végre. Tehát mondhatjuk, hogy a szélességi keresés speciális esete a Dijkstra algoritmusnak, ahol a prioritásos sor helyett, egy egyszerű sort használunk, amellyel javítunk a műveletigényen. Azt kell belátni, hogy a sor használatával is mindig a legkisebb távolságú csúcsot választjuk KÉSZ-nek, de ez következik a szélességi keresésnél megvizsgált invariáns tulajdonságból.