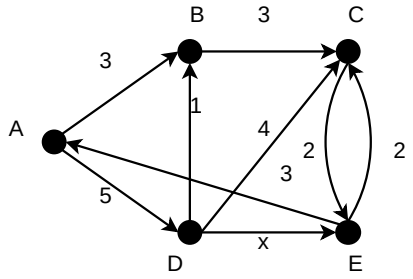
Algel IV. gyakorlat

2012. február 29.

2. **[ZH: 2009. április 24.] Dijsktra-algoritmussal határozza meg az alábbi gráfban az A pontból az összes többi pontba menő legrövidebb utak hosszát az x pozitív valós paraméter függvényében. Minden lépésnél írja fel a távolságokat tartalmazó D tömb állapotát és a KÉSZ halmaz elemeit.**



A	**B**	**C**	**D**	**E**	**KÉSZ**
$\underline{0}$	**3**	∞	5	∞	A
$\underline{0}$	$\underline{3}$	6	**5**	∞	A,B
$\underline{0}$	$\underline{3}$	6	$\underline{5}$	$5+x$	A,B,D

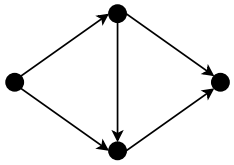
Ha $0 < x \leq 1$:

$\underline{0}$	$\underline{3}$	**6**	$\underline{5}$	$\underline{5+x}$	A,B,D,E
$\underline{0}$	$\underline{3}$	$\underline{6}$	$\underline{5}$	$\underline{5+x}$	A,B,C,D,E

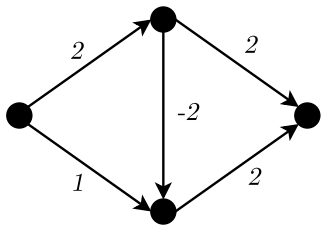
Ha $1 < x$:

$\underline{0}$	$\underline{3}$	$\underline{6}$	$\underline{5}$	$\min(8, 5+x)$	A,B,C,D
$\underline{0}$	$\underline{3}$	$\underline{6}$	$\underline{5}$	$\underline{\min(8, 5+x)}$	A,B,C,D,E

3. **Határozzuk meg az alábbi gráfban az élsúlyokat úgy, hogy a Dijkstra algoritmus rossz eredményt adjon!**



Attól, hogy rakunk bele negatív élsúlyt, még akár működhetne! Úgy kell negatív élsúlyt belerakni, hogy romoljon el, pl:



A bal oldali csúcsból futtatva az alsó csúcsra az alogritmus 1-et mond, pedig a legrövidebb út oda 0 lenne.

4. **Adjuk meg az összes olyan minimális élszámú irányított gráfot (élsúlyokkal együtt), amely(ek)re az alábbi táblázat a Dijkstra algoritmusban szereplő $D[\,]$ tömb változásait mutathatja. Adjuk meg a legrövidebb utakat tartalmazó $P[\,]$ tömb állapotait is!**

v_1	v_2	v_3	v_4	v_5	v_6
0	2	6	∞	∞	7
0	2	5	9	∞	6
0	2	5	6	9	6
0	2	5	6	8	6
0	2	5	6	7	6

A táblázat alapján vizsgáljuk az algoritmus futását. Mivel a legkisebb élszámú gráf kell, ezért csak oda húzunk be élet, ahol az egyes lépésekben javítottunk (ide pedig kötelező). Látszik, hogy v_1-ből indulunk. Az első sor miatt innen v_2-be, v_3-ba és v_6-ba vezet él, a megfelelő súlyokkal. A KÉSZ-be v_2-t választjuk, és a javítások miatt rájövünk, hogy kell lennie egy v_2v_3, v_2v_4 és egy v_2v_6 élnek, 3, 7 és 4 súlyokkal. A továbbiakban minden ugyanígy (a gráfot nem rajzolom fel). Csak az utolsó előtti lépésben lesz választási lehetőség, hogy v_4-et vagy v_6-ot vegyük be a KÉSZ-be. Ettől függően lesz egy 2 súlyú v_4v_5 és egy 1 súlyú v_6v_5 élünk, vagy pont fordítva. A P tömb változásai (csak az első esetben):

v_1	v_2	v_3	v_4	v_5	v_6
–	v_1	v_1	v_1	v_1	v_1
–	v_1	v_2	v_2	v_1	v_2
–	v_1	v_2	v_3	v_3	v_2
–	v_1	v_2	v_3	v_4	v_2
–	v_1	v_2	v_3	v_6	v_2

5. **Egy G gráfban pontosan egy él súlya negatív, és nincs a gráfban negatív összsúlyú irányított kör. Adjunk $O(n^2)$ lépésszámú algoritmust az $s \in V(G)$ pontból az összes többi pontba vezető legrövidebb utak meghatározására!**
Egy legrövidebb út kétféle lehet: vagy használja a negatív élet, vagy nem. Először futtassunk egy Dijsktrát (a szépség kedvéért hagyjuk ki a negatív élet), az i csúcsba vezető legrövidebb út hosszát jelölje d_i^+. Jelölje u azt a csúcsot, amiből a negatív él indul. Világos, hogy d_u^+ helyes, hiszen a negatív élen nem mehetünk át az ő eléréséhez az eredeti gráfban. Belőle is indíthatunk egy Dijsktrát (az eredeti gráfon), hiszen ekkor először a negatív él másik végpontját veszi be a KÉSZ halmazba, ami a definíciónak megfelel. Ezeket a legrövidebb értékeket jelölje d_i^-. A legrövidebb utak tehát $\min(d_i^+, d_u^+ + d_i^-)$ képlettel számolhatók. Lépésszám egy keresés és két Dijsktra, azaz $O(e+n^2+n^2) = O(n^2)$ (az ugye mindenkinek világos, hogy egy egyszerű gráfban $e = O(n^2)$).

11. **[Vizsga: 2008. május 27.] Egy kupacba beraktunk egy új x elemet, majd végrehajtottunk egy MINTÖR műveletet. Mikor fordul elő, hogy végül az eredeti kupacot kapjuk vissza?**
Pontosan akkor, ha az eddig tárolt elemeknél kisebb x. Ez a feltétel szükséges, hiszen a MINTÖR a legkisebbet fogja visszaadni, és ha az új nem az lenne, akkor mindenképpen bennemaradni. Az elégségesség kicsit macerább. Amit tudunk: x a gyökérig felment, és legalulról indult. MINTÖRNÉL a legalsó kerül x helyébe (nevezzük y-nak), az fog lefele menni. Azt akarjuk bizonyítani, hogy pont az eredeti helyére kerül vissza (és mindenki más is, aki az úton volt). Indukcióval az aktuális szintre: a gyökérben most y van, egyik gyereke az eddigi legkisebb, és pont azon az oldalon van, ahonnan x eredetileg „feljött". Mindenképpen ő fog a gyökérbe kerülni, ezért y vele fog helyet cserélni, azaz ugyanazon az úton indul lefele, ahonnan x jött. Ez az érvelés pontosan ugyanígy megy tetszőleges közbenső szinten is, ha y aktuális pozícióját vesszük a gyökérnek.

12. **Adjunk hatékony (*hehe :)*) algoritmust egy kupac tizedik legkisebb elemének meghatározására!**
Konstans (azaz $O(1)$), hiszen az i-edik legkisebb elem a gyökértől legfeljebb i távolságra van,

így 2^{10} elem közül kell a tizedik legkisebbet megtalálni, ami konstans lépés (a legbutább módon is).

13. **Adott egy n elemet tartalmazó kupac és egy k kulcs. Keressük meg a kupac k-nál kisebb elemeit! Ha m ilyen elem van, akkor az algoritmus $O(m)$ elemi lépést használhat.**
Lényegében egy bejárás, elindulunk a gyökérből, és amíg k-nál kisebb elemet találunk, addig kiírjuk, ha $\geq k$-t, akkor az adott ágat hanyagoljuk, hiszen a kupac-tulajdonság miatt arrafele csak még nagyobbak lehetnek. Legfeljebb a nekünk jó elemek mindkét gyerekét vizsgáljuk meg feleslegesen, vagyis $O(2m) = O(m)$ vizsgálatot végzünk.

14. **Adjunk konstans szorzó erejéig optimális költségű algoritmust az alábbi problémára!**
INPUT: Egy $A[1:n]$ tömb, amely eredetileg az $1, \ldots, n$ számokat tartalmazta kupacba rendezve, de öt elem megsérült, és a helyére $*$ került.
FELADAT: Találjuk meg a tömb összes olyan kitöltését, ami lehetett az eredeti!
$O(n)$ lépésből ki lehet találni, hogy melyik 5 szám hiányzik. Ezeknek 5! sorrendje lehet, így mindegyiket végig tudjuk próbálni. Minden egyes elemnek csak a szülőjét és két gyerekét kell megvizsgálni, hogy jó-e velük a kupactulajdonság (permutációnként 15 vizsgálat). Ha az adott permutáció jó, akkor van egy lehetséges kitöltésünk. Minden lehetőséget megvizsgáltunk, így az algoritmus biztos helyes. Lépésszám: $O(n + 5! \cdot 15) = O(n)$. Ez azért optimális, mert pusztán az 5 sérült elem megtalálása is $\Omega(n)$ lépés.

15. **Tervezzünk olyan adatszerkezetet, ami egy rendezett halmaz elemeinek tárolására szolgál. A megvalósítandó műveletek: *Felépít*(n): n elemből felépíti a struktúrát; *Mintör*, *Maxtör*: a min. illetve max. elem törlése; *Beszúr*(x): az x elemet a struktúrába illeszti. Az egyes műveletek költsége ne legyen több, mint *Felépít*: $O(n)$; *Mintör, Maxtör, Beszúr*: $O(\log n)$, ahol n a tárolt elemek száma.**
Használjunk egy min-kupacot és egy max-kupacot, valamint a tárolt elemeket a két kupacban kössük össz pointerekkel (hogy az egyik kupacban lévő, a kezünkben lévő elemről keresés nélkül tudhassuk, hogy hol található a másik kupacban).
Felépít: pointerek inicializálása + két kupac építése: $O(n + n + n) = O(n)$.
Beszúr: pointer inicializálása + beszúrás egyikbe és másikba: $O(1 + \log n + \log n) = O(\log n)$.
Mintör: min-kupacból mintör, max-kupacban pedig a megfelelő hely törlése (annyi különbséggel az eredeti algoritmushoz képest, hogy nem a gyökérből, hanem valahonnan középről indítjuk a törlést, de az algoritmus ugyanúgy fog működni). $O(\log n + \log n) = O(\log n)$.
Maxtör: ugyanaz mint előbb, csak szerepcserével.

16. **Igazoljuk, hogy egy n elemből álló kupac felépítése $\Omega(n)$ összehasonlítást igényel!**
Fejet asztalbaverően egyszerű bizonyítás: mivel a gyökérben a legkisebb elem van, ezért nem lehet hamarabb felépíteni, mint a legkisebb elemet megtalálni, ami pedig önmagában is $\Omega(n)$.