Aszalós László

Informatikai algoritmusok

mobiDIÁK könyvtár

Aszalós László

Informatikai algoritmusok

mobiDIÁK könyvtár

SOROZATSZERKESZTŐ

Fazekas István

Aszalós László

Informatikai algoritmusok

Szakmai segédanyag
Programtervező matematikusok részére
első kiadás

mobiDIÁK könyvtár
Debreceni Egyetem
Informatikai Kar

mobiDIÁK könyvtár
Debreceni Egyetem
Informatikai Kar
4010 Debrecen, Pf. 12
http://mobidiak.unideb.hu

A mű a *A mobiDIÁK önszervező mobil portál* (IKTA, OMFB-00373/2003) és a *GNU Iterátor, a legújabb generációs portál szoftver* (ITEM, 50/2003) projektek keretében készült.

Tartalomjegyzék

I. fejezet
Binomiális kupacok

A korábban már megismert kupacok esetén a minimális elem megkeresése konstans, a kupac valamely elemének a törlése logaritmikus bonyolultságú volt. A kupacba szúrás (a tömb megnyújtása egy elemmel) szintén megoldható logaritmikus bonyolultsággal, csak az új elem és ősei esetén kell a beszúró rendezésben használt ciklusmagot egyszer végrehajtani. A kupacok összefűzése viszont lényegében egy új kupac felépítését jelenti, tehát lineáris bonyultságú.

1. Binomiális fák

Egy új adatszerkezet, a *binomiális kupac*ok esetén a kupacok egyesítése már egyszerűbb, logaritmikus bonyolultságú.

A binomiális kupac binomiális fák halmaza lesz. Lássuk először ennek a definícióját!

A B_0 binomiális fa egyetlen (gyökér) csúcsból áll. A B_k binomiális fa két összekapcsolt B_{k-1} binomiális fából áll: az egyik gyökércsúcsa a másik gyökércsúcsának legbaloldalibb gyereke. A B_k binomiális fára fennállnak a következő tulajdonságok:

- 2^k csúcs van.
- a magassága k.
- az i-dik mélységben pontosan $\binom{k}{i}$ csúcs van.
- a gyökércsúcs fokszáma k, ami nagyobb, mint bármely más csúcs fokszáma; másrészt, ha a gyökércsúcs gyerekeit balról jobbra haladva megszámozzuk $k-1$, $k-2$, $\ldots 0$-val, akkor az i gyerek egy B_i fa gyökércsúcsa.

2. Binomiális kupac felépítése

Egy H binomiális kupac binomiális fák olyan halmaza, amely kielégíti a következő *binomiális kupac tulajdonság*okat:

- H minden binomiális fája kupac rendezett, azaz egy csúcs kulcsa nagyobb, vagy egyenlő mint a szülője kulcsa.
- H-ben legfeljebb egy olyan binomiális fa van, melyben a gyökércsúcsnak egy meghatározott fokszáma van.

A második feltétel alapján az n csúcsot tartalmazó kupac szétbomlik 2^i csúcsot tartalmazó, különböző fákra, amelyekből így csak $\ln n + 1$ lehet.

Az alábbi programokban a következő jelöléseket használjuk: az `x` csúcs kulcs mezőjére `x.kulcs`, az `x` csúcs szülőjére az `x.apa`, az `x` csúcs legbaloldalibb (legidősebb) gyerekére `x.gyerek`, az `x` csúcstól jobbra következő csúcsra az `x.testver`, az `x` csúcs gyerekeinek számára az `x.fokszam` jelöléssel hivatkozunk. Gyökércsúcsok esetén a `x.testver` a következő gyökérre mutat, vagy `Nil` értéket kap. A gyökércsúcsok ezen láncolt listájára a `H.fej`-jel hivatkozunk.

3. Binomiális kupacok műveletei

3.1. Új kupac készítése

Egy üres binomiális kupac készítése (`BINOMIÁLIS-KUPACOT-LÉTREHOZ`) csak abból áll, hogy a `H.fej`-et `Nil`-re állítjuk. Ezért a bonyolultság konstans.

3.2. Minimális kulcs megkeresése

Mivel a kupacot alkotó binomiális fák kupacrendezettek, így a gyökérelemek között találjuk meg a minimális kulcsú elemet. Ezért a függvény bonyolultsága $O(\ln n)$.

Function `BINOMIÁLIS-KUPACBAN-MIN(`H`)`

Input: `H` binomiális kupac
Output: `H` minimális kulcsú elemének a címe

```
1  y ← NIL
2  x ← H.fej
3  min ← ∞
4  while x ≠ NIL do
5  |  if x.kulcs < min then
6  |  |  min ← x.kulcs
7  |  |  y ← x
8  |  endif
9  |  x ← x.testver
10 endw
11 return x
```

3.3. Két binomiális kupac egyesítése

Mivel a binomiális kupac definíciója alapján a listában nem lehet két ugyanolyan fokszámú binomiális fa, így ha két ilyen B_k fával találkozunk, akkor kettőjük

összefűzésével alkotunk egy B_{k+1} fát. Az összefűzést a BINÁRIS-ÖSSZEKAPCSOLÁS rutin valósítja meg. Mivel itt csak négy értékadás történik, konstans a bonyolultság.

Procedure `BINOMIÁLIS-ÖSSZEKAPCSOLÁS(`*y,z*`)`

Input: `y, z` binomiális fák gyökérelemeinek címei
Eredmény: `y`-t `z` alá fűzi

1 $y.apa \leftarrow z$
2 $y.testver \leftarrow z.gyerek$
3 $z.gyerek \leftarrow y$
4 $z.fokszam \leftarrow z.fokszam + 1$

A BINOMIÁLIS-KUPACOKAT-EGYESÍT függvény felhasznál egy összefésülő rutint, mely egy olyan listát készít, melyben a binomiális fák méret szerint növekvő sorrendben találhatóak. Ha az eredeti fák is ilyen alakban voltak, akkor ez az összefésülés a két kupacban szereplő binomiális fák számával lineáris bonyolultságú. A rutin végigfut az így kapott lista elemein, és szükség szerint az azonos méretű fákat összekapcsolja. Ennek megfelelően a teljes bonyolultság a csúcsok számának logaritmusa.

3.4. Csúcs beszúrása a kupacba

Ez három lépésből áll. Elsőként létre kell hozni egy üres kupacot, majd második lépésben ebbe beszúrni az egyetlen csúcsunkat. Ezután nem marad más, mint ezt az új kupacot egyesíteni a régivel.

3.5. Minimális kulcsú csúcs kivágása

A minimális kulcsú csúcsra a gyökérelemek között találunk rá. Ezt a láncolt listát kell végignézni a minimális elemért. Ezután egy új, üres kupacot kell készíteni, és verem módjára ki kell bővíteni a minimális elem fája gyökerének fiaival. Mivel a magasabb fokszámú gyerekek vannak korábban, így ezek kerülnek hátra az új listában. A minimális elemhez tartozó fát törölni kell az eredeti listából, és ennek eredményét egyesíteni kell az gyerekekből származó kupaccal.

3.6. Kulcs csökkentése

Ha a kulcsot egy valóban kisebb értékre kívánjuk csökkenteni, akkor az adott csúcs szülein kell végigfutni, és a beszúró rendezéshez hasonlóan kell előretolni (felbuborékoztatni) a csökkentett kulcsú elemet.

Function BINOMIÁLIS-KUPACOKAT-EGYESÍT(H_1,H_2)

Input: H_1 és H_2 binomiális kupac
Eredmény: A két kupacot egyesíti binomiális kupaccá

```
1  H ← BINOMIÁLIS-KUPACOT-LÉTREHOZ()
2  H.fej ← BINOMIÁLIS-KUPACOKAT-ÖSSZEFÉSÜL(H1,H2)
3  if H.fej=NIL then return H
4  w ← NIL
5  x ← H.fej
6  y ← x.testvér
7  while y ≠ NIL do
8      if x.fokszám ≠ y.fokszám or
       (y.testvér ≠ NIL and y.testvér.fokszám = x.fokszám) then
9          w ← x
10         x ← y
11     else
12         if x.kulcs≤ y.kulcs then
13             x.testvér← y.testvér
14             BINOMIÁLIS-ÖSSZEKAPCSOLÁS(y, x)
15         else
16             if w = nil then
17                 H.fej← y
18             else
19                 w.testvér ← y
20             endif
21             BINOMIÁLIS-ÖSSZEKAPCSOLÁS(x, y)
22             x ← y
23         endif
24     endif
25     y ← x.testvér
26 endw
27 return H
```

3.7. Csúcs törlése

Ez nem áll másból, mint a csúcs kulcsának lehető legkisebbre csökkentéséből, s ehhez kapcsolódóan az előző rutin meghívásából, majd a minimális kulcsú elem törléséből.

II. fejezet
Fibonacci-kupacok

Bizonyos gyakorlati feladatokban nincs gyakran szükség elemek törlésére. Ekkor a korábban logaritmikus bonyolultságú rutinok helyett közel konstans bonyolultságúakat is használhatunk.

1. Fibonacci-kupac felépítése

A korábbihoz hasonlóan a Fibonacci-kupac is kupac-rendezett fák gyűjteménye. Viszont a kupac-rendezett fák nem feltétlenül binomiális fák.

A binomiális kupacokban a fák a méret szerint rendezve voltak. Itt ilyen rendezés nincs. Azért, hogy könnyebben mozoghassunk a gyökerek illetve a testvérek között, a korábbi egyirányban láncolt listákat duplán láncolt ciklikus listák váltják fel. Természetesen két ilyen duplán láncolt ciklikus lista összefűzése konstans bonyolultságú. A csúcsoknál feljegyezzük a gyerekek számát. A binomiális kupacnál a fej a gyökérlista elejére, a legkisebb fára mutatott. Itt a `H.min` a minimális kulcsot tartalmazó fa gyökerére, azaz a minimális kulcsú elemre mutat.

2. Fibonacci-kupac műveletei

2.1. Minimális kulcs megkeresése

A kupac `min` mutatója pont ezt adja meg, nem kell keresni.

2.2. Új Fibonacci-kupac létrehozása

Itt is egy üres szerkezetet kell készíteni, ami konstans bonyolultságú.

2.3. Egy csúcs beszúrása egy kupacba

A csúcshoz tartozó adatokat kell aktualizálni, beírni a kulcsértéket, beállítani, hogy nincs gyereke, majd a kupac gyökérlistájába kell beszúrni ezt a csúcsot. Végül ha a csúcs kulcsértéke kisebb, mint a kupachoz tartozó minimum, akkor erre a csúcsra kell irányítani a mutatót. A binomiális kupacokra jellemző összevonás itt elmarad.

2.4. Kupacok egyesítése

A két kupac gyökérlistáját kell egyesíteni, s meg kell vizsgálni, hogy melyik minimális elem kisebb, mert ez lesz az új minimális elem.

2.5. Minimumcsúcs kivágása

A binomiális kupachoz hasonlóan itt is gyökérlistába gyűjtjük a minimális elem gyerekeit, amire most használhatjuk az eredeti listát is. Viszont az elhalasztott összevonásokat is most hajtjuk végre. Ehhez felhasználunk egy tömböt, mely az i-fokszámú csúcsok helyét tartalmazza. A gyökérlistán végiglépkedve ha egy olyan fokszámú csúcsot találunk, mely még nem szerepelt, akkor azt a tömb megfelelő helyén feljegyezzük. Ha olyan fokszámra akadunk, amelyet már feljegyeztünk, akkor az aktuális és feljegyzett fákat egyesítjük a gyökérelemek kulcsai alapján. Természetesen az új fát is össze kell vetni a táblázattal. Miután a gyökérlistát bejártuk, a feljegyzett címekből kell az új gyökérlistát összeállítani egy ciklusban, és a ciklusmagban figyelni a minimális elemet.

2.6. Kulcs csökkentése

Ha a kulcs csökkentése után az adott elem és apja között a kupac tulajdonság már nem áll fenn, akkor az adott elemet le kell választani a szülőről és a gyökérlistába beszúrni, majd össze kell hasonlítani a minimális elemmel, s ha szükséges azt aktualizálni.

2.7. Csúcs törlése

A csúcs kulcsának minimálisra (minusz végtelen) csökkentése után a minimális csúcsot kell törölni.

III. fejezet
Maximális folyamok

Egy súlyozott gráfot nem csupán úgy lehet elképzelni, mint egy térképet, ahol az egyes csúcsok a városokat, az élek a köztük meglévő utakat jelölik, a súlyok pedig az adott út megtételével járó költséget (kilométer, forint, stb). Felfoghatjuk úgy is, mintha az élek egy hálózat elemei lennének, és a súly pedig kapacitást jelölne, azaz vízvezeték esetén megadná, hogy egy csövön mennyi víz folyhat át egységnyi idő alatt; villamos hálózat esetén pedig megadhatná, hogy egy vezeték milyen erősségű áramot bír el; közúthálózat esetén megadhatná, hogy az egyes buszmegállók között napjában hány utast képesek a buszok szállítani.

Ezekben az esetekben a gráf egy csúcsát termelőnek (forrásnak) tekintjük, ahonnan a gráf egy másik csúcsába a fogyasztóba (nyelőbe) áramlik a víz, áram, illetve a lakosság. Feltesszük, hogy a gráf ugyanolyan iramban termeli a terméket, mint ahogy azt a fogyasztó felhasználja, és a gráf többi csúcsában nem halmozódnak fel a termékek. Természetesen merül fel az az alapvető kérdés, hogy egy adott gráf esetén mennyi lehet az maximális mennyiség, ami eljuthat a termelőtől a fogyasztóig, illetve ekkor melyik élen mennyi mennyiséget kell szállítani.

1. Ford-Fulkerson algoritmus

A $G = (E, V)$ irányított gráfot, ha minden $(u, v) \in E$ élének adott egy nemnegatív $c(u, v)$ kapacitása, hálózatnak nevezzük. (Ha $(u, v) \notin E$, akkor $c(u, v)$ legyen 0.) A termelőt s-sel, a fogyasztót t-vel jelöljük. Feltesszük, hogy a gráf minden csúcsa rajta van valamely s-ből t-be vezető úton.

Hálózati folyamnak nevezzük azt a $f : V \times V \to R$ függvényt, melyre a következők teljesülnek:

- $f(u, v) \leq c(u, v)$ minden $u, v \in V$ esetén,
- $f(u, v) = -f(v, u)$ minden $u, v \in V$ esetén,
- $\sum_{v \in V} f(u, v) = 0$ minden $u \in V - \{s, t\}$ esetén.

Az $f(u, v)$ értéket az u-ból v-be vezető élen a folyam értékének nevezzük. A folyam negysága a $\|f\| = \sum_{v \in V} f(s, v)$. Feladatunk a maximális folyam meghatározása.

Legyen adott egy f folyam. Egy (u, v) él reziduális kapacitása legyen a következő: $c_f(u, v) = c(u, v) - f(u, v)$, magyarul a kihasználatlan kapacitás. Adott f folyam esetén definiálható az f-hez tartozó $G_f = (V, E_f)$ reziduális hálózat, ahol

$$E_f = \{(u, v) \in V \times V \| c_f(u, v) > 0\}.$$

Előfordulhat hogy $(u,v) \notin E$, ám $(u,v) \notin E_f$. Mindenesetre egy él csak akkor szerepelhet a reziduális hálózatban, ha az él, vagy a fordítottja szerepel az eredeti gráfban. Ezzel a reziduális gráf éleinek száma felülről korlátozható az eredeti gráf éleinek kétszeresével.

A reziduális hálózatban egy s-ből t-be vezető utat *javítóút*nak nevezünk. A p javítóút kapacitása $c_f(p) = \min\{c_f(u,v)|(u,v)\ p\text{-beli él}\}$.

Procedure FORD-FULKERSON(*G,s,t*)

Input: G gráf, s forrás, t nyelő
Eredmény: f maximális folyam

```
1  forall the (u,v) ∈ E do
2  |  f[u,v] ← 0
3  |  f[v,u] ← 0
4  endfall
5  while Van s-ből t-be p út G_f-ben do
6  |  c_f(p) ← min{c_f(u,v)|(u,v) ∈ p}
7  |  forall the (u,v) ∈ p do
8  |  |  f[u,v] ← f[u,v] + c_f(p)
9  |  |  f[v,u] ← −f[u,v]
10 |  endfall
11 endw
```

A kapacitások rendszerint egész számokként vannak megadva. (Racionális számok esetén a közös osztóval beszorozva újra egész számokhoz jutunk.) Ekkor a maximális folyam értéke is egész. Ennek $|E|$-szorosa adja a bonyolultságot, ha az utakat a reziduális gráfban szélességi kereséssel határozzuk meg. Ha a kezdetben a legrövidebb utakkal dolgozunk, majd sorra haladunk a hosszabbak felé, akkor a bonyolultság $O(VE^2)$ lesz.

2. Előfolyam algoritmusok

Ha a hálózati folyam definíciójában a harmadik pontot az alábbira cseréljük, akkor az *előfolyam* definícióját kapjuk: $\sum_{v\in V} f(u,v) \geq 0$ minden $u \in V - \{s\}$ esetén. (Magyarul megengedjük, hogy az egyes csúcsokban felhalmozódás legyen). A feleslegfolyamon az alábbi értéket értjük: $e(u) = \sum_{v\in V} f(u,v)$.

Az algoritmus során kezdetben a forrásból kivezető minden élet maximális kapacitással üzemeltetünk. A gráf csúcsai bizonyos magasságban helyezkednek el, amelyek az algoritmus futása során növekedhetnek. A magasabban fekvő csúcsokból a felesleget az élek mentén a lentebb elhelyezkedő csúcsokba „pumpálhatjuk át”. Ha egy csúcsban felesleg van és innen indul telítetlen él, de nem pumpálhatunk belőle, akkor ezt a pontot valamely telítetlen élének másik csúcsa fölé emelhetjük (megemelés).

Procedure PUMPÁLÁS(*u,v*)

Input: u, v csúcsok
Eredmény: A túlfolyásból átadunk v-nek

1 $d_f(u,v) \leftarrow \min(e[u], c_f(u,v))$
2 $f[u,v] \leftarrow f[u,v] + d_f(u,v)$
3 $f[v,u] \leftarrow -f[u,v]$
4 $e[u] \leftarrow e[u] - d_f(u,v)$
5 $e[v] \leftarrow e[v] + d_f(u,v)$

Procedure MEGEMELÉS(*u*)

Input: u csúcs
Eredmény: u magassága növekszik

1 $h[u] \leftarrow 1 + \min\{h[v] | (u,v) \in E_f\}$

Procedure INDULÓ-ELŐFOLYAM(*G,s*)

Input: G gráf, s forrás
Eredmény: induló előfolyamot készít

1 **forall the** $u \in V$ **do**
2 $\quad h[u] \leftarrow 0$
3 $\quad e[u] \leftarrow 0$
4 **endfall**
5 **forall the** $(u,v) \in E$ **do**
6 $\quad f[u,v] \leftarrow 0$
7 $\quad f[v,u] \leftarrow 0$
8 **endfall**
9 $h[s] \leftarrow |V|$
10 **forall the** $u \in \{(u,v) \in E\}$ **do**
11 $\quad f[s,u] \leftarrow c(s,u)$
12 $\quad f[u,s] \leftarrow -c(s,u)$
13 $\quad e[u] \leftarrow c(s,u)$
14 **endfall**

Az általános előfolyam algoritmus kezdetben meghívja az INDULÓ-ELŐFOLYAM rutint, majd egy ciklus addig fut, amíg a PUMPÁLÁS vagy a MEGEMELÉS feltételei teljesülnek. Ekkor a ciklusmagban a megfelelő művelet hajtódik végre.

Bizonyítás nélkül megjegyezzük, hogy ez a rutin minden esetben megadja a maximális folyamot.