Algel IV. gyakorlat

Egy kupac kupac

2011. február 28.

6. **[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.

7. **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).

8. **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.

9. **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.

10. **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.

11. **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 $\Omega(n)$.