Algel IX. gyakorlat

Hash!

2009. április 7/9.

2. **Mutassuk meg, hogy nyitott címzéses hashelés és lineáris próba esetén már két kulcshoz tartozó hash-függvényérték megegyezése is okozhat tetszőlegesen nagy méretű csomósodást!**
Konstruktívan: legyen $h(x_1) = h(x_2)$, ezután $h(x_i) = h(x_{i-1}) - 1$. Így csak x_1-hez és x_2-höz tartozó érték egyezik meg, mégis látható, hogy tetszőleges nagy n-re n hosszú csomónk lesz (és minden beszúrás ütközéses volt).

3. **Mi a baja a $h(k) = k^2 \pmod 7$ hash-függvénynek, ha a tábla 7 méretű?**
Nem az összes lehetséges értéket veszi fel. Ki kell számolni mind a 7 lehetőséget, és látszik.

4. **A $T[0 : M - 1]$ táblában rekordokat tárolunk nyitott címzésű hashelt szervezéssel. Az ütközések feloldására lineáris próbát alkalmazunk. Tegyük fel, hogy a tábla használata során egy hibás törlés történt: egy cellából kitöröltünk egy rekordot a törlés-bit beállítása nélkül.**

   (a) **Igaz-e, hogy a hibás törlés helye mindig megtalálható?**
   Nem, pl egy elem volt a táblában, és azt töröltük.

   (b) **Adjunk hatékony (lineáris) algoritmust a tábla megjavítására! (Módosítsuk a táblát úgy, hogy megszűnjenek a hibás törlés negatív következményei!)**
   Akkor okozhat ez problémát, ha az utolsó üres hely mondjuk i, de egy i-nél kisebb pozíciójú x elemre $h(x) \geq i$. Ha ilyen helyzet nincs, akkor jó a tábla. Arra kell gondolni, hogy a táblán végighaladva mindig az utolsó üres hely érdekel minket. Induljunk el a végéről, és minden elemre számoljuk ki $h(x)$-et, és nézzük meg, hogy a nyilvántartott utolsó üresről vagy arról túlról származik-e. Ha igen, akkor máshol nem lehetett hibás törlés, hiszen ha mégsem ott lett volna, akkor a törlés előtt is rossz lett volna a tábla. Bökjük be az utolsó üresen a törlés-bitet! A tábla jó lesz, hiszen helyes törlés után is így nézne ki a tábla. Egy végigolvasásból megvoltunk, tehát lineáris az algoritmus.

8. **[Vizsga: 2008. június 3.] Az 1 és 91 közötti összes 3-mal osztható egész számot valamilyen sorrendben egy M méretű hash-táblába raktuk a $h(x) = x \pmod M$ hash-függvény segítségével, lineáris próbával. Ennek során hány ütközés fordulhatott elő, ha $M = 35$, illetve ha $M = 36$?**
$M = 35$ esetén a $3 \ldots 33$ számok a saját helyükre kerülnek, $0 \pmod 3$ helyekre, $36 \ldots 69$ számok $1 \pmod 3$ helyekre, $72 \ldots 90$ számok pedig $2 \pmod 3$ helyekre, és mindegyik elfér, tehát nincs ütközés. $M = 36$-nál csak $3 \ldots 33$ fér be ütközés nélkül, $36 \ldots 69$ pont ugyanezekre a helyekre menne, $72 \ldots 90$ szintén. Innen már ki lehet számolni.

9. **[Vizsga: 2005. május 26.] A kezdetben üres M méretű hash-táblába sorban beraktuk a $k_1, k_2, \ldots, k_n$ kulcsokat a $h(x) \equiv x \pmod M$ hash-függvénnyel, lineáris próbával. Jelölje t_1 a keletkezett táblában az egymás melletti foglalt mezők maximális számát. Amikor ugyanezt a $k_1, k_2, \ldots, k_n$ sorozatot ugyanabban a sorrendben egy üres $2M$ méretű táblába rakjuk be a $h(x) \equiv x \pmod{2M}$ hash-függvénnyel, lineáris próbával, akkor a kapott táblálban legyen t_2 az egymás melletti foglalt mezők maximális száma.**

   (a) **Igazolja, hogy $t_2 \leq t_1$**
   Tfh $t_2 > t_1$. Ekkor vegyünk egy t_2 hosszú sorozatot a $2M$ méretű táblában, és nézzük meg, hogy ezek hogyan helyezkednének el az eredeti táblában (a beszúrások sorrendje megegyezik!). Vegyük észre, hogy az eredetibe beszúráskor, ha egyéb ütközés nem volt,

ugyanezt a sorozatot kapjuk (ciklikus értelemben, modulo aritmetika miatt), vagy esetleg hosszabbat is (lineáris próba miatt). Ez ellentmond a feltételnek, tehát $t_2 \leq t_1$.

(b) **Igaz-e, hogy $t_1 \leq 2t_2$?**
Ellenpélda: $|0|4|2|$ (mod 3), de $|0|\ |2|\ |4|\ |$ (mod 6), $t_1 = 3$, $t_2 = 1$, és $3 \not\leq 2 \cdot 1$.

---

10. **[Vizsga: 2003. március 31.] Tervezzen adatstruktúrát a következő feltételekkel. Természetes számokat kell tárolni, egy szám többször is szerepelhet. A szükséges műveletek:**
BESZÚR(i): i egy újabb példányát tároljuk
TÖRÖL(i): i egy példányát töröljük
MINDTÖRÖL(i): i összes példányát töröljük
DARAB(i): visszaadja, hogy hány példány van i-ből
ELEM(K): megmondja, a nagyság szerinti rendezésben a K-adik elem értékét.
Az adatstruktúra legyen olyan, hogy ha m-*féle* elemet tárolunk, akkor mindegyik művelet lépésigénye $O(\log m)$.
(Például ha a tárolt elemek $1, 1, 3, 3, 3, 8$, akkor DARAB$(1) = 2$, ELEM$(4) = 3$ és $m = 3$.)
Legyen mondjuk egy 2-3 fa, csak minden pointer mellé tároljuk el azt is, hogy az adott részfában hány elemet tárolunk, a levelekben pedig az szerepel, hogy az adott kulcsból hány darab van. Beszúrás annyiban változik, hogy ha már bent van az elem, csak a számlálóját növeljük, és ezt felfele a gyökérig megtesszük (hiszen az adott részfában eggyel nő az elemek száma), amúgy rendes beszúrás (részfák elemszámára figyelve); törlés hasonlóan, minden törlés értelemszerűen; darabszám kereséssel, és számláló megmondja. A K-adik elem megtalálása lényegében egy keresés, ahol mindig abba a részfába megyünk, amivel még nem lépjük túl K-t (hiszen az összegek megvannak). Mivel m levelű 2-3 fánk van, és az egyes lépések a standard műveletek, csúcsonként konstans adminisztrációval, részfaméretek nyilvántartása miatt esetleg $O(szint)$ extra munkával, ezért $O(\log m)$-ből mindegyik megy.

11. **[PZH: 2008. május 9.] Vázolja a 2-3 fának (és műveleteinek) egy olyan módosítását, amiben továbbra is van KERES, BESZÚR, TÖRÖL, MIN, MAX művelet, és ezeken kívül van még RANG és K-ADIK művelet is, ahol RANG(x) azt adja vissza, hogy a tárolt elemek között az x a rendezés szerint hányadik elem, a K-ADIK(i) pedig, hogy a rendezés szerint a tárolt elemek közül melyik az i-edik. A módosítás során a felsorolt szokásos műveletek lépésszámának nagyságrendje ne változzon, és mindkét új művelet lépésszáma legyen $O(\log n)$, ahol n a tárolt elemek száma.**
Fentebb mintha hasonlót csináltunk volna, sőt, az még többet is tud.