17. tétel

Adatszerkezetek és algoritmusok vizsga

Frissült: 2013. január 28.

Hasító táblák

A gyakorlatban sokszor csak a BESZÚR, KERES, TÖRÖL műveleteket megvalósító dinamikus szerkezetre van szükségünk – hogyan lehetne ezt hatékonyan tenni?

Az eddigi kereső szerkezeteink a kulcsok összehasonlításán alapultak. Végrehajtási idő $O(n)$ vagy $O(\log n)$ volt - lassú. Szeretnénk az előzőnél jobbat elérni.

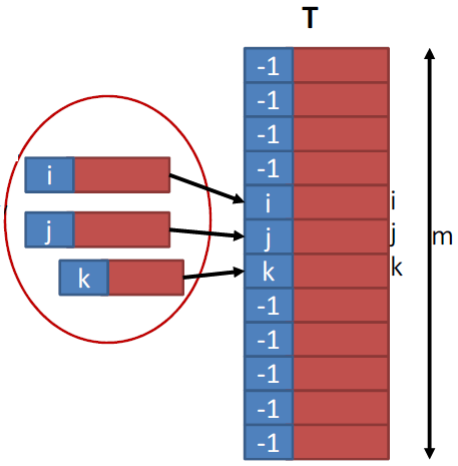
Azt szeretnénk kihasználni, hogy ha vektorban indexelünk, az csak O(1)!

A hash-elés:

- Adott a K kulcsok halmaza. A K elemeivel azonosított rekordok száma azonban várhatóan jóval kisebb, mint K számossága.
- Ekkor K-t egy alkalmas h függvénnyel leképezzük az ábrázolás alapját képező kisebb tartományra. Legyen ez a [0 .. M-1] intervallum.
- A
$$h : K \to [0..M-1]$$
függvényt hash-függvénynek nevezzük.
- Mivel M < |K|, sőt általában M $\ll$ |K|, ezért h nem lehet injektív, hanem szükségképpen fellép a kulcsütközés: van olyan k $\neq$ k', amelyre h(k) = h(k').

Közvetlen hozzáférésű (címzésű) táblák. Tegyük fel, hogy az elemek kulcsai különböző egész értékek a [0 , m-1] intervallumból, és m nem túl nagy. Használjuk magukat a kulcsértékeket, hogy i kiválasszunk egy helyet a T közvetlen hozzáférésű táblában, melyben az elemeket tároljuk

Egy k kulcsú elem **keresése**: nézzük meg a k indexű elemet. ha van itt egy érték, megtaláltuk, ha a jelző itt pl. -1, akkor nincs benne.



Keresés, beszúrás, törlés (tömör) pszeudokódja.

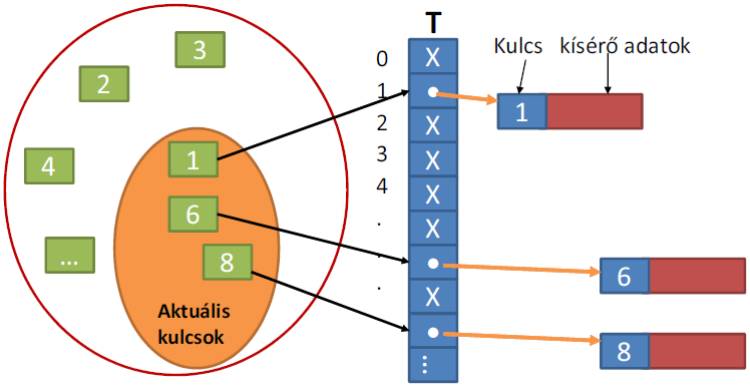
```
Közvetlen_címzésű_keresés (T, k)
  return T[k]

Közvetlen_címzésű_beszúrás (T, x)
  T[kulcs[x]] ← x

Közvetlen_címzésű_törlés (T, x)
  T[kulcs[x]] ←-1
```

Mindegyik O(1) idejű!

Megszorítások. A kulcsok

- egészek
- egyediek
- kis intervallumból valóak legyenek
- sűrűn legyenek az intervallumban. Ha ritkásan vannak - sok üres hely van az értékek között - túl sok helyet használunk el, hogy a sebességet megnyerjük

Másik lehetőség a tárolásra. T-indexei a kulcsok, T-ben mutatók, csak akkor tárolom az egészet, ha kell.



Ilyenkor a műveletek.

```
T inicializálása
     minden elemét null-ra állítjuk

Közvetlen_címzésű_keresés (T, k)
    return „a T[k] által mutatott objektum"

Közvetlen_címzésű_beszúrás (T, x)
    helyet foglalunk x-nek,
    T[kulcs[x]] mutatóját erre állítjuk

Közvetlen_címzésű_törlés (T, x)
    felszabadítjuk a T[kulcs[x]] által mutatott objektumot
    (T[kulcs[x]] is null lesz!)
```

Itt is, mindegyik O(1) idejű

Megszorítások gyengítése és a kulcsütközések

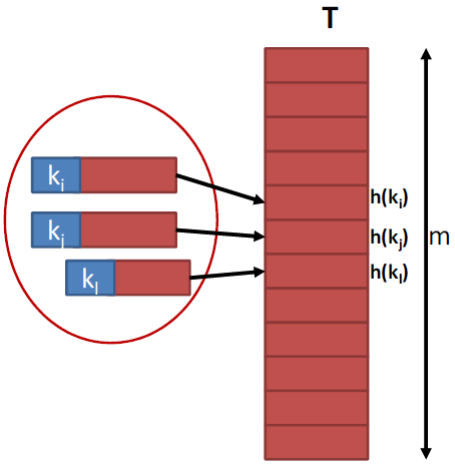
A fenti megszorítások túl erősek. Elég lesz egy gyengébb is.

A kulcsok egészek legyenek. Kell egy hash függvény

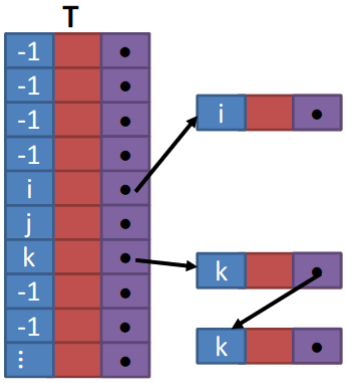
$$h(kulcs) \rightarrow egész$$

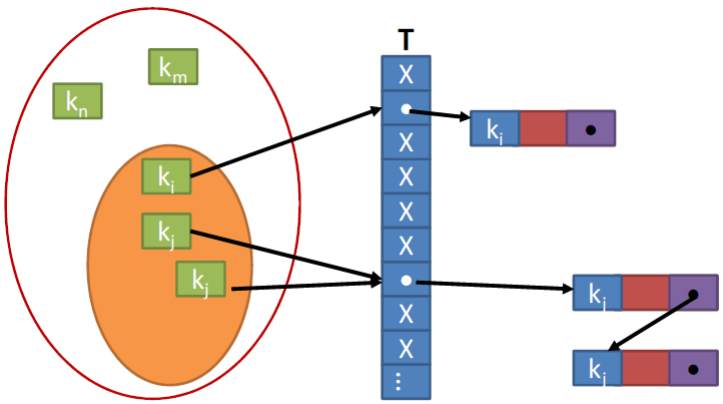
Ezt a függvényt a kulcsra alkalmazva egy indexet kapunk

Ha h minden kulcsot egy egyedi egész értékre képez le a $0...m-1$ intervallumban, akkor a keresés O(1)



A kulcsok egyediek legyenek. Hozzuk létre a **duplikátumok** láncolt listáját, és ezt kapcsoljuk a táblához. ha egy keresésnek elég akármelyik k kulcsú elem, a végrehajtás még mindig O(1)





Hash függvény

Tetszőleges függvény, ami a 0..m-1 -ben generál értékeket egy megfelelő (nem túl nagy) m–re jó lesz! Amíg ő O(1).

Ez a fv. 0...255 egy értékét adja vissza:

```
int hash( char *s, int n )
{
   int sum = 0;
   while( n-- ) sum = sum + *s++;
   return sum % 256;
}
```

Vagy: *xor* függvény is jó:

```
sum = sum ^ *s++;
```

Hash függvények egyszerű egyenletestől univerzálisig

Egyszerű egyenletes hasítás

$P(k)$ legyen annak a valószínűsége hogy a k kulcs előfordul, a hasító táblánkban m hely van, egy egyenletes has függvény, $h(k)$ biztosítani fogja:

$$\sum_{k|h(k)=0} P(k) = \sum_{k|h(k)=1} P(k) = ... \sum_{k|h(k)=m-1} P(k) = \frac{1}{m}$$

(szumma minden k-ra, melyre $h(k) = 0$).
A kulcsok száma minden helyre azonos.
Ha a kulcsok a $[0, r)$-en véletlenszerűen szétszórt egészek, akkor

$$h(k) = \left\lfloor \frac{mk}{r} \right\rfloor$$

egy egyenletes hash függvény. A kulcsokat egészek egy intervallumára képeztük le: $0 \leq k < r$. Most csökkentsük ezt az intervallumot $[0, m)$-re, ahol m a hash tábla egy elfogadható mérete.

Hogyan?

- Osztás (mod) -
$$h(k) = k \mod m$$
m mi legyen: pl. 2^n
- Szorzó módszer: konsanssal szorzunk (0 < A < 1), utána kivesszük a tört részt belőle:
$$(kA - \lfloor kA \rfloor)$$
Utána ezt még felszorozzuk m-mel:
$$h(k) = \lfloor m * (kA - \lfloor kA \rfloor) \rfloor$$
Itt jó a 2 hatvány.
- Univerzális hashelés

Univerzális hashelés

A hash függvényt véletlenül, az aktuálisan tárolandó kulcsoktól függetlenül választjuk meg.

A hasító függvényt egy függvényosztályból futás közben véletlenül választjuk ki.

Legyen H a hasító függvények véges halmaza, ezek egy K kulcs univerzumot a $[0, m)$ tartományba képeznek le.

H **univerzális**, ha $\forall x, y \in K, x \neq y$ kulcspárra azoknak a $h \in H$ hasító függvényeknek a száma, melyre $h(x) = h(y)$, pontosan $\frac{|H|}{m}$. Ilyenkor a kulcsütközés valószínűsége pontosan $\frac{1}{m}$

Megválasztásuk. Vegyük egy olyan p prím számot, mely elég nagy hogy minden kulcs benne legyen a $[0, ..., p-1]$-ben $(p > m)$.

Jelölés:

$$Z_p = \{0, 1, ...p-1\}, Z_p^* = \{1, 2, ..p-1\}$$

Definiáljuk: $\forall a \in Z_p^*, \forall b \in Z_p$:

$$h_{a,b}(k) = ((a * k + b) \mod p) \mod m$$

Az ilyen függvények osztálya:

$$H_{p,m} = \{h_{a,b} : a \in Z_p^*, b \in Z_p\}$$

Tétel. A hasító függvények fenti egyenlőségekkel definiált $H_{p,m}$ osztálya univerzális.

Kulcsütközés

Pl. a fenti függvényre:

```
hash("AB", 2) és hash("BA", 2)
```

ugyanazt az értéket adják!

Ezt hívjuk kulcsütközésnek: a hash függvény két különböző kulcshoz rendeli ugyanazt a címet. Számos technikát használnak a feloldására.

A táblázat fel kell ismerje és fel kell oldja ezt.

Felismerés. Tároljuk az aktuális kulcsot az elemmel a hash táblában: Számítsuk ki a címét k = h(kulcs) Ellenőrizzük a találatot:

```
if (table[k].key == kulcs) then találat
else próbáld a következőt
```

Kulcsütközésekre megoldások

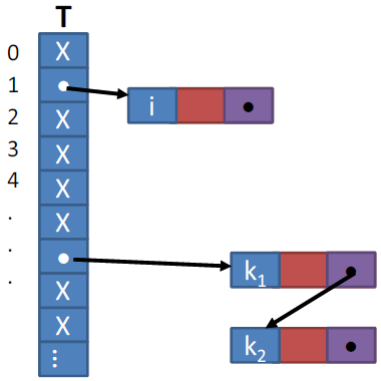
Láncolt lista

Minden tábla elemhez egy láncolt listát rendelünk Keressünk i kulcsú elemet: X.

Láncolt hasító keresés(T, i): kiszámítjuk $h(i)$-t, keressük az i kulcsú elemet a $T[h(i)]$ listában. Ha NULL-t találunk, a kulcs nincs a táblában

Beszúrunk x-t: *Láncolt hasító beszúrás*(T, x): kiszámítjuk h(x.kulcs)-t, beszúrunk a $T[h(x.kulcs)]$ lista elejére.

Törlés – hasonlóan a $T[h(x.kulcs)]$ listából.



Túlcsordulási terület

A „láncolt” listát a tábla egy speciális területén hozzuk létre – ez a túlcsordulási terület

Tfh. h(k) == h(j) és a k lett először tárolva

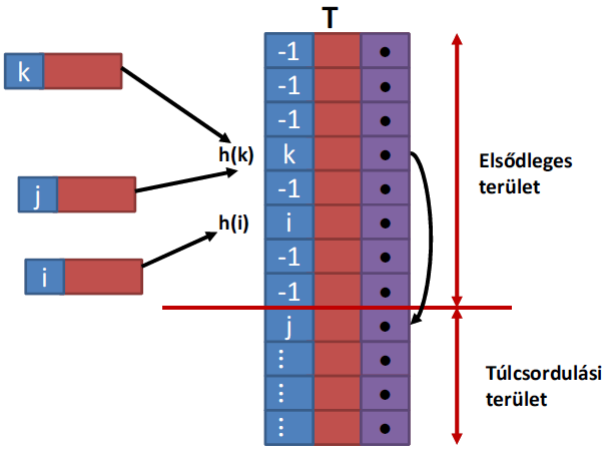
Pl. Hozzáadjuk j-t:

- Kiszámítjuk h(j)-t
- Megtaláljuk k-t :-(
- Megkeressük az első szabad helyet a túlcsordulási területen
- betesszük j-t
- k „pointere” erre mutat (a mutató itt a táblabeli index)

Többféle lehetőség a – lista elejére, végére is szúrhatunk be

Keresés – ugyanaz, mint a láncolt lista

Törlésnél – lehet a lista utolsó elemét idetenni, de lehet új állapotot bevezetni a töröltre, ez majd jön.



Nyílt címzés

Az elemeket a táblában tároljuk.

Egy elem keresésénél végigmegyünk a táblázaton, amíg meg nem találjuk, vagy el tudjuk dönteni, hogy nincs benne a táblában.

Elem beszúrásánál kipróbáljuk az összes helyet, amíg üreset nem találunk – valamilyen stratégia szerint – ez a beszúrandó kulcs függvénye, nem a 0 .. m-1 felsorolás!

Kiterjesztjük a hash függvény értelmezési tartományát az ún. kipróbálási számmal:

- $h : K \times \{0, 1, ..., m-1\} \to \{0, 1, ...m-1\}$
- Megköveteljük, hogy minden k kulcsra a

$$(h(k, 0), h(k, 1), ..., h(k, m-1))$$

kipróbálási sorozat a $\{0, 1, \ldots, m-1\}$ egy permutációja legyen.

Így előbb-utóbb minden hely szóba jön.

Algoritmusok:

Keresés. Keresésnél aszerint keressük, ahogy a beszúrás betehette.

```
Hasító_keres(T, k)
i ← 0
repeat j ← h(k,i)
if T[j] = k
  then return j
  i ← i + 1
until T[j] = NIL vagy i = m
return NIL
```

Törlés. Bonyolultabb, ui. nem elég csak NIL-t írni, hiszen akkor egy következő keresés nem találná meg azokat, amelyek később vannak a táblában ⇒ vezessünk be a NIL-en kívül még egy szimbólumot, a TÖRÖLT-et

```
Hasító_töröl(T, k)
i ← 0
repeat j ← h(k,i)
  if T[j] = k
     then T[j] ← TÖRÖLT
       return
  i ← i + 1
until T[j] = NIL vagy i = m
```

Beszúrás. Tfh. a T hasító táblázatban csak kulcsok vannak vagy NIL

```
Hasító_beszúr(T,k)
i ← 0
repeat j ← h(k,i)
  if T[j] = NIL vagy T[j] = TÖRÖLT
    then T[j] ← k
            return
    else i ← i + 1
until i = m
error „hasító táblázat túlcsordulás"
```

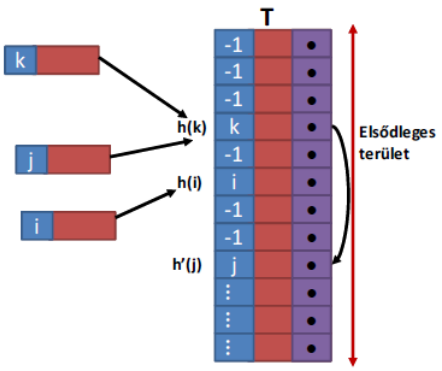
Kettős hashelés

Használjunk egy második hash függvényt – h'. Sok variáció lehet. Általános név: re-hashing – kettős hashelés

Feltessszük, hogy h(k) == h(j), és k lett először tárolva. Hozzáadjuk j-t, kiszámítjuk h(j)-t, így megtaláljuk k-t. Ismételjük, míg találunk üres helyet: kiszámítjuk h'(j)-t, betesszük j-t.

Keresés – használd h(x)-t, aztán h'(x)-t

Egy mezőnek 3-féle státusza lehet: üres – foglalt – törölt (hogy a keresés folytatódhasson)



A második hash fv-nél lehet pl.

- Négyzetes próba
- Dupla hasítás