Projektovanje algoritama

L08. Hash tabele

Direktno adresiranje

- Koristi se kod statičkih nizova.
- Operacije INSERT, DELETE, GET zahtevaju O(1) vreme.

- Ograničenja:
  - Ključevi su samo nenegativni celi brojevi
  - Veliki broj različitih ključeva dovodi do velikog memorijskog zauzeća

- Prednost:
  - Sve operacije pristupa su O(1) u najgorem slučaju (worst-case)

Hash tabele

- Prostor za skladištenje nije **ključ** nego **funkcija ključa**.

$$h\colon\ U \to \{0, 1, 2, \ldots, m-1\}$$

- *h(k)* – **hash vrednost** ključa *k*.

- Problem:
  - **Kolizija** – više od jednog ključa se mapira na istu lokaciju.

Hash tabele – rešavanje kolizije

- Više elemenata koji se mapiraju na isti ključ se smeštaju u listu.

- SEARCH $\boldsymbol{T(n) = \theta(l)}$
- INSERT $\boldsymbol{T(n) = \theta(1)}$
- DELETE $\boldsymbol{T(n) = \theta(1)}$ (ako je lista dvosmerna)

Hash tabele – uniformno mapiranje

- Sve lokacije imaju jednaku verovatnoću da budu mapirane.

- Ukupan broj elemenata: n
- Ukupan broj lokacija: m

- Faktor ispune (*load factor*)*:* $\alpha = \frac{n}{m}$

Hash tabele – uniformno mapiranje

- Dva slučaja pretrage:
  - *Neuspešna pretraga:* $\boldsymbol{T(n) = \theta(1 + \alpha)}$
  - *Uspešna pretraga:* $\boldsymbol{T(n) = \theta(1 + \alpha)}$
- Cilj: broj lokacija treba da bude proporcionalan broju elemenata!

Hash funkcije

- Dobra hash funkcija prati uniformnu raspodelu.
- Međutim:
  - Raspodela ključeva gotovo uvek **nije** uniformna.
  - Raspodela ključeva gotovo uvek **nije** predvidiva.
  - Teško je definisati funkciju koja „pegla“ neuniformnost raspodele ključeva i formira uniformnu raspodelu po lokacijama hash tabele.
- Hints:
  - Bliski ključevi treba da budu što udaljeniji u hash lokacijama.
  - Hash vrednost ne treba da zavisi od šablona u podacima.
  - Interpretirati ključeve kao prirodne brojeve, čak i ako to nisu.

Hash funkcije – metod ostatka pri deljenju

$$h(k) = k \bmod m$$

- Šta izabrati za ukupan broj lokacija (m)?

- Loš izbor: broj koji je stepen dvojke (ostatak odgovara samo delu bita)
- Dobar izbor: prost broj koji nije blizu stepena dvojke

Hash funkcije – metod množenja

$$h(k) = \lfloor m(kA \bmod 1) \rfloor \qquad 0 < A < 1$$

- Šta izabrati za ukupan broj lokacija (m)? Nije bitno.
- Šta izabrati za A?

$$A \approx \frac{\sqrt{5} - 1}{2} \qquad [Knuth]$$

Otvoreno adresiranje

- Ne formiramo listu na svakoj lokaciji
  - **Jedna lokacija = jedan element**

- Definišemo **redosled lokacija** koje pretražujemo za svaki element
  - **Sve lokacije se konačno moraju pretražiti**.
  - **Ubrzanje dobijamo time što redosled definišemo takav da se element najverovatnije nalazi na lokacijama koje su na početku redosleda.**

Otvoreno adresiranje – dodavanje elementa

```
HASH-INSERT(T,k)
  i = 0
  repeat
    j = h(k,i)
    if T[j] == NIL
      T[j] = k
      return j
    else
      i = i + 1
  until i == m
  error "hash table overflow"
```

$T(n) = ?$

Otvoreno adresiranje – pretraga elementa

```
HASH-SEARCH(T,k)
  i = 0
  repeat
    j = h(k,i)
    if T[j] == k
      return j
    i = i + 1
  until T[j] == NIL or i == m
  return NIL
```

$T(n) = ?$

Otvoreno adresiranje – uklanjanje elementa

- Ukoliko uklonimo element postavljanjem na NIL, prekidamo lanac pretrage (redosled) i elemente nakog tog elementa ne možemo dostići.

- Jedno rešenje: uvesti stanje DELETED za uklonjene elemente.
- Problem: elementi ostaju u kasnijim delovima lanca pretrage i posle dužeg vremena gubimo na brzini pretrage.
  - **Ukoliko nam treba često uklanjanje elemenata, koristiti prethodnu ideju sa listom na svakoj lokaciji.**

Otvoreno adresiranje – hash funkcije

- Kako definisati dobru hash funkciju koja formira redosled pretrage po lokacijama?

- **Cilj: biti što bliže uniformnoj raspodeli redosleda, odn. tome da hash funkcija proizvodi svaki redosled od *m!* mogućih permutacija lokacija sa jednakom verovatnoćom.**

Otvoreno adresiranje – hash funkcije
Linearni redosled (*linear probing*)

$h'(k) \quad - \; neka\ hash\ funkcija$

$$\boldsymbol{h(k,i) = (h'(k) + i) \bmod m}$$

Otvoreno adresiranje – hash funkcije
Kvadratni redosled (*quadratic probing*)

$$h'(k) \quad - \; neka\ hash\ funkcija$$

$$\boldsymbol{h(k,i) = \left(h'(k) + c_1 * i + c_2 * i^2\right) \bmod m}$$

Otvoreno adresiranje – hash funkcije
Dvostruki hash (*double hashing*)

$h_1'(k), \quad h_2'(k) \; - \; neke\ dve\ hash\ funkcije$

$$\boldsymbol{h(k,i) = \left(h_1'(k) + i * h_2'(k)\right) \bmod m}$$

Primer:

$$h_1(k) = k \bmod m$$

$$h_2(k) = 1 + (k \bmod m'), \qquad m' < m \qquad [npr.\ m = 701, m' = 700]$$

Koliko je ovo stvarno brzo?

Neuspešna pretraga:

$$T(n) = O\left(\frac{1}{1-\alpha}\right)$$

Uspešna pretraga:

$$T(n) = O\left(\frac{1}{\alpha}\ln\frac{1}{1-\alpha}\right)$$