Algoritmy a dátové štruktúry

Hash

23.11.2010

Abstraktné dátové typy

- Všeobecne štruktúra, ktorá zvláda uložiť istý typ prvkov a robiť nad nimi požadované operácie

- Príklad Stack
  - ukladali sme (aj opakujúce sa) čísla (dokonca ľubovoľné prvky)
  - operácie Push(i), Pop(), Top(), Empty()
- Implementácia ADT môže byť rôzna
  - v tomto prípade pole, spj. zoznam

ADT Slovník

- Ukladáme prvky s kľúčmi – t.j. dvojice (k,O) dokonca môžu existovať (k,O_1) a (k,O_2)
- Operácie
  - Search(k): vyhľadávanie podľa kľúča
  - Insert(k,O)
  - Delete(k,O)
- Príklady: tel. zoznam, mapovanie host names na IP adresy

Implementácia slovníka - Log

- Ukladáme si netriedenú postupnosť
  - napríklad ako dvojito-spájaný zoznam
- Insert – jednoducho niekam pridáme O(1)
- Find, Delete – musíme v najhoršom prejsť celý Logfile [napríklad akk tam prvok nie je] O(n)

- Použitie: buď relatívne malé slovníky alebo situácie, kedy sa hlavne vkladá (napr. Log)

Implementácia slovníka - Hash

- Hashovacia tabuľka **pevnej** veľkosti N (0..N-1) priradí pomocou nejakej funkcie h() každému prvku s kľúčom nejaké miesto v tabuľke
- Prvok (k,O) umiestnime na mieste i=h(k)
- Potom operácie sú nasledovné
  - Insert: Tab[h(k)]← (k,O)
  - Find: return Tab[h(k)]
  - Delete: if Tab[h(k)]==(k,O) then remove

Príklad - Hash

- Chceme ukladať dvojice (ISBN, NázovKnihy)
- Naša hashovacia tabuľka bude veľkosti 1000 (čiže bude ukladať do 0..999)
- Hashovacia funkcia bude brať posledné 3 cifry ISBN tj. h(ISBN)=ISBN mod 1000

- zrejme sa môže stať, že vkladáme druhú knihu na už zaplnené miesto – **KOLÍZIA**

Riešenie kolízii - reťazenie

- Každé miesto tabuľky je vlastne zreťazený zoznam
- Insert: vloženie prvku (k,O) do zoznamu T[h(k)]
- Search(k): prezeranie zoznamu T[h(k)]
- Delete(k,O): vymazanie prvku (k,O) zo zoznamu T[h(k)]

Riešenie kolízii – Lin. probing

- Keď zistím, že vznikla kolízia, tak prvok umiestnim na ďalšiu pozíciu
- Pr.: h(x)=x mod13,Ins(18,41,22,44,59,32,31,73)
- Všimnite si – máločo je na svojom “mieste”

  18_5, 41_2, 22_9, 44_5, 59_7, 32_6, 31_5, 73_8

- Problém: ak teraz vymažeme 18 a budeme hľadať 44?
- Vytvoríme špeciálnu hodnotu “available”

Riešenie kolízii – Lin. Probing (2)

```
Find(k)
  i←h(k); p←0;
  repeat
    c=A[i];
    if c=Ø return false
    else if c.key=k return c
    else i←i+1 mod N; p++;
  until p=N;
```

```
Insert(k,O)
  i←h(k); p←0;
  repeat
    c=A[i];
    if (c=Ø or c=”av”) c←(k,O)
    else i←i+1 mod N; p++;
  until p=N;
  return ERR;
```

```
Delete(k,O)
  i←h(k); p←0;
  repeat
    c=A[i];
    if c=Ø return ERR
    else if c=(k,O) return c; c←”av”
    else i←i+1 mod N; p++;
  until p=N;
```

Riešenie kolízii – Double Hash

- Keď zistím, že vznikla kolízia, tak prvok umiestnim o nejakú hodnotu d(k) ďalej
- Pr.: h(k)=k mod13, d(k)=7-k mod 7
  Ins(18,41,22,44,59,32,31,73)

k	h	d	Probes
18	5	3	5
41	2	1	2
22	9	6	9
44	5	5	5,10
59	7	4	7
32	6	3	6
31	5	4	5,9,0
73	8	4	8

-
- Aká má byť funkcia d?
  - rozhodne by nemala dosahovať 0
  - ak prime(N) možeme pridaním d prejsť celé pole
  - bežne d(k)=q – k mod q, kde q je prvočislo < N

Hashovacie funkcie

- Výsledná hashovacia funkcia je obvykle

$$h(k)=h_2(h_1(k))$$

- h_1: kľúče → integer
  - hash code map
- h_2: integer → 0..N-1
  - compression map

Hashovacie funkcie (2)

- Hash code map $h_1(k)$
  - Pamäťová adresa
  - Integer cast
  - Component sum: $k=a_0..a_{n-1}$ a $h(k)=\sum a_i$
  - Polynóm: $k=a_0..a_{n-1}$ a $h(k)=\sum a_i.z^i$

Hashovacie funkcie (3)

- Compression map: $h_2(x)$
- Najbežnejšia je $h_2(x) = x \bmod N$ (N je prvočíslo)
- Iný príklad $h_2(x) = (a.x + b) \bmod N$
  - a,b>0
  - a mod N ≠ 0 (ináč všetko na b)

Vlastnosti hashovania

- Najhorší prípad pre všetky operácie O(n)
  - s tým sa nič nedá urobiť – prečo?
- Zoberieme teda naplnenie tabuľky $\alpha = n/N$
  - dá sa ukázať, že pri náhodných číslach je počet pokusov v double hashingu $1 / (1 - \alpha)$
- Z toho vyplýva, že ak chceme operácie v O(1)
  - Netreba príliš napĺňať tabuľku
  - Treba “náhodné” čísla

Hashovanie v iných oblastiach

- My sme potrebovali hash na určenie miesta
  - Chceli sme jedine “dobre rozdelenie”
  - Využitie v DB, programovanie – asoc. pole
- Kontrola integrity dát
  - Potrebné, aby zmena v dátach zmenila hash
- Krypto – napr. kontrola hesiel
  - Okrem toho chceme aby nebola invertovateľná
  - Ťažko zistiť kolízie
  - ...

Cvičenia

- Príklady hashovacích funkcii
- Univerzálne hashovanie?