Heširanje

Idealno pretrazivanje je kada na osnovu zadatog kljuca mozemo da direktno pristupimo podatku bez uporedjivanja sa drugim kljucevima. Ovo bi bilo moguce ukoliko bi obezbedili tabelu koja ima onoliko mesta koliko ima razlicitih mogucih vrednosti kljuceva. Glavni problem je ovde kada je broj mogucih kljuceva daleko veci od broja potrebnih kljuceva pa je onda iskoriscenje tabele vrlo malo.

Za resenje ovog problema potrebno je pronaci funkciju koja transformise kljuc u ceo broj koji pripada opsegu indeksa u tabeli. Upravo je **hesiranje** tehnika kojom se vrsi preslikavanje skupa kljuceva na tabelu znacajno manjih dimenzija. Idealno bi bilo da funkcija za svaki kljuc daje jedinstvenu poziciju. Ta funkcija naziva se **hes funkcija,** a tabela koja se koristi u tom postupku zove se **hes tabela.** Ponekad se ovaj metod hesiranja naziva i tehnikom **rasutog adresiranja ili tehnikom direktnog adresiranja.** Ulaz hes tabele moze pored kljuceva da sadrzi i same zapise ili pokazivace na zapise. Kapacitet ulaza tabele je jedan kljuc ili jedan zapis koji kljuc identifikuje. Ako je velicina hes tabele ***n***, onda su indeksi od ***0*** do ***n-1.***

Neka je $T[i],\ 0 \leq i < n-1$, hes tabela sa n ulaza i neka kljucevi pripadaju nekom skupu mogucih vrednosti *S*. Broj kljuceva je tipicno mnogo veci od broja ulaza u tabeli. Neka je *h* hes funkcija koja vrsi mapiranje kljuceva u adresni prostor tabele. Tada funkcija primenjena na neki kljuc K daje **maticnu adresu *i* kljuca K, $i = h(K)$.**

//

42.a) (03.septembar 2008.)

Objasniti i analiticki definisati pojam kolizije.

odgovor:

Moguce je da hes funkcija za dva razlicita kljuca da istu vrednost:

$$h(K_i) = h(K_j),\ \ K_i \neq K_j$$

Posto dva kljuca ne mogu biti smestena na isto mesto u tabeli, ovaj slucaj predstavlja problem koji se naziva **kolizija.** Kljucevi Ki i Kj se nazivaju **sinonimi**, a skup sinonima je **klasa ekvivalencije.** Kolizije je prakticno nemoguce izbeci ukoliko kljucevi nisu unapred poznati.

//

Pozeljne osobine za efikasnost pristupa su:

- **uniformnost** – za hes funkciju se kaze da je uniformna na skupu kljuceva S kada za slucajni kljuc K ∈ S postoji jednaka verovatnoca da se on mapira u bilo koju maticnu adresu:

$$P(i = h(K)) = 1/n, \quad 0 \leq i \leq n - 1$$

- **odrzavanje poretka** – sekvencijalan pristup u poretku kljuceva bi se mogao ostvariti hes funkcijama koje odrzavaju poredak i zadovoljavaju uslov:

$$h(K_i) > h(K_j) \text{ za } K_i > K_j$$

Pri primeni tehnike hesiranja treba napraviti dve skoro nezavisne odluke:

- izabrati efikasnu hes funkciju
- izabrati metod za razresavanje problema kolizije

Heš funkcija

Kriterijum za izbor hes funkcije:

- **jednostavnost** – radi brzeg izracunavanja
- **uniformnost** – u cilju smanjenja verovatnoce kolizije

U praksi kljucevi mogu biti numericki, alfanumericki, alfabetski. Vecina hes funkcija trazi da kljuc bude iskljucivo numericki zbog aritmetickih operacija nad njim, pa alfabetski i alfanumericki kljucevi treba da se predstave pre primene hes funkcije u numerickom obliku.

Metod ekstrakcije

Ovo je izuzetno jednostavan metod koji se svodi na izvlacenje (ekstrakciju) odredjenog potrebnog broja cifara sa unapred definisanih pozicija.

Ako imamo kljuc koji se na primer sastoji od 7 cifara i imamo tabelu od na primer 1000 (0 do 999) ulaza. Nama su potrebne tri cifre da adresiramo ovu tabelu. Mi sada uzmemo cifre sa nekoh definisanih pozicija. Neka su to pozicije 2, 4 i 5. Potom slepimo, formiramo indeks i udjemo u tabelu i to nam je maticna adresa. Ovom metodu je najbolja osobina jednostavnost, a sve ostalo je diskutabilno jer ce sinonimi biti svi oni kljucevi koji imaju iste cifre na ove tri pozicije. Pozeljno je da hes funkcija zavisi od svih znakova kljuca, ali i od njihovih pozicija jer bi se u suprotnom kljucevi koji se mogu dobiti prostom permutacijom znakova mapirali na iste adrese.

Kako biramo hes funkcije?!

Hes funkcije mogu biti:

- **nezavisne od raspodele kljuceva** – ovo je cesca varijanta gde nam nisu poznate verovatnoce pojavljivanja kljuceva
- **zavisne od raspodele kljuceva** – ovo je povoljniji slucaj jer mi tacno znamo koji ce nam kljucevi dolaziti. Onda mozemo da odaberemo uniformniju hes funkciju.

Heš funkcije nezavisne od raspodele ključeva

Metod deljenja

Rezultat hes funkcije je ostatak pri deljenju celobrojnog kljuca nekim brojem n koji je manji ili jednak velicini hes tabele:

$$h(K) = K \bmod n$$

Performanse ovog metoda u smislu smanjenja verovatnce kolizije veoma zavise od pravilnog izbora delioca *n*.

//

45. (februar 2012.)

Objasniti kako ne treba odnosno kako treba birati delilac kod metoda deljenja pri hesiranju.

odgovor:

Ne preporucuje se da *n* bude:

- **parno** – jer se tada parni kljucevi mapiraju u parne ulaze, a neparni u neparne ulaze. Ako su kljucevi pretezno parni ili neparni, samo polovina ulaza ce biti opterecena.
- **stepen broja 2 ili 10** – delilac koji je stepen broja 2, iako omogucava lako izracunavanje ostatka izdvajanjem donjih *i* bitova kljuca, upravo zbog toga nije preporucljiv jer ne zavisi od svih bitova kljuca. Delilac kao stepen broja 10 nije pogodan ako su kljucevi decimalni brojevi.
- **neprosto sa modulom kongruencije –** u slucaju da su kljucevi kongruentni po modulu d treba da se izbegava vrednost *n* koja nije uzajamno prosta sa d

Preporucuje se da *n* bude:

- **prost broj ne previse blizu stepena broja 2**

//

Metod množenja

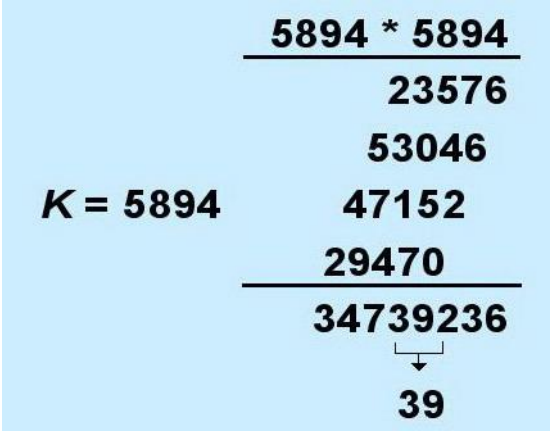
Metod je zasnovan na mnozenju kljuca pogodno odabranom realnom konstantom sa vrednoscu izmedju 0 i 1. Zatim se od proizvoda uzme samo deo iza decimalne tacke, sto bi trebalo da predstavlja slucajan broj koji zavisi od svih cifara kljuca. Na kraju se dobijeni broj pomnozi sa velicinom tabele i dobije pozicija ulaza u tabelu.

$$h(K) = n(cK \bmod 1),\ \ 0 < c < 1$$

Ovde izbor delioca *n* nije kritican. Ipak, za performanse je bitan izbor vrednosti konstante *c*. Neke analize su pokazale da je povoljno birati konstantu kao ***c = 0.61803***.

Metod sredine kvadrata

Numericka reprezentacija kljuca se pomnozi sama sa sobom, pa se sa fiksnih pozicija iz sredine kvadrata uzme onoliko cifara koliko je potrebno za adresiranje tabele.

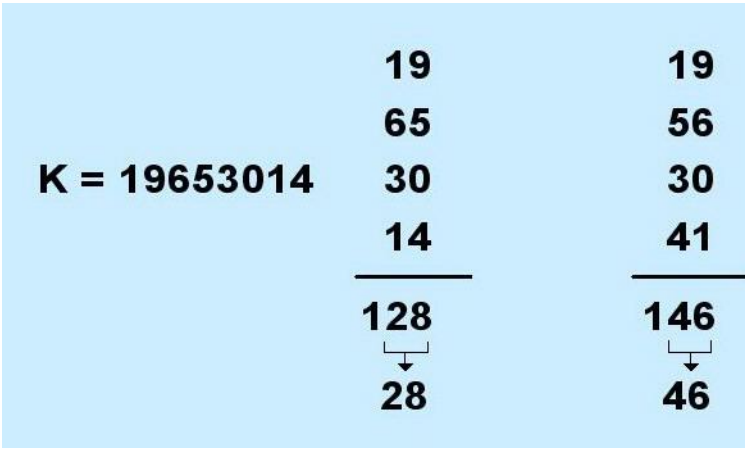


Metod sklapanja

Metod se zasniva na deobi kljuca na delove iste duzine. Zatim se ovi delovi saberu, a prenos se ignosrise.

U ovoj metodi hes funkcija zavisi od svih cifara, ali ne i od njihovih pozicija, pa je u mnogim slucajevima efikasnost pod znakom pitanja. Da bi se malo poboljsala uniformnost, koristi se druga varijanta ove metode.

U drugoj koloni je demonstrirana varijanta ovog metoda sa obrtanjem kod koje se redosled cifara u svakom drugom delu invertuje pre primene operacije sabiranja.



Metod konverzije osnove

Ako je kljuc dat u brojnom sistemu sa osnovom p (obicno 2 ili 10), ovaj metod tretira kljuc kao broj u osnovi q (q>p).

Ako kljuc ima vrednost 6154 u dekadnom brojnom sistemu (p = 10), kada se prebaci u sistem q=13, dobije se:

$$6 * 13^3 + 1 * 13^2 + 5 * 13^1 + 4 * 13^0 = 13420$$

pa se izdvoji potreban broj cifara. Da bi se smanjila verovatnoca kolizije, ***q*** se odabira tako da bude uzajamno prosto sa ***p***.

Savrsena hes funkcija

Hes funkcija koja obezbedjuje da nema kolizije u nekom skupu kljuceva se naziva savrsena hes funkcija. Ona ima osobinu:

$$h(K_i) \neq h(K_j), \text{ za svako } i \neq j \text{ gde } K_i, K_j \in S$$

Savrsenu hes funkciju nije lako pronaci, pogotovo za dinamicki skup. Sto je hes tabela veca u odnosu na broj kljuceva, savrsenu hes funkciju je lakes naci. Idealno bi bilo imati savrsenu hes funkciju za *n* kljuceva u tabeli sa *n* ulaza. Takva savrsena hes funkcija se naziva **minimalnom.**

Heš funkcije zavisne od raspodele ključeva

Kada je skup kljuceva unapred poznat, onda je moguce naci efikasnu hes funkciju koja ce smanjiti broj kolizija.

//

40. (februar 2007.)

Objasniti metodu hesiranja koja koristi ***analizu cifara.***

odgovor:

Metod analize cifara

Neka su kljucevi poznati unapred. Mi sada zelimo da primenimo prost metod ekstrakcije, tj. da hes formiramo od cifara sa unapred izabranih pozicija, ali treba pravilno izabrati pozicije. Prvo se analizira skup numerickih vrednosti svih kljuceva, pa se napravi tabela koja za svaku poziciju kljuca daje broj prijavljivanja pojedinih cifara na toj poziciji u svim kljucevima. Zatim se selektuje onoliko pozicija koliko je cifara potrebno da se adresira hes tabela tako sto se odaberu kolone tabele koje pokazuju najmanje varijacije pojavljivanja razlicitih cifara.

//

Razresavanje kolizije

Kolizija predstavlja slucaj kada hes funkcija mapira dva ili vise razlicitih kljuceva na istu maticnu adresu. Broj kolizija se najjednostavnije moze smanjiti povecanjem velicine tabele. Tako se poboljsavaju performanse hesiranja, ali po cenu dodatnog prostora i njegovog slabijeg iskoriscenja.

Pri kvantitativnom odredjivanju performansi najcesce se koristi parameter koji se naziva ***popunjenost tabele α (load factor).*** Ovaj parameter predstavlja odnos broja zauzetih ulaza u tabeli i velicine tabele.

Za razresavanje kolizije tradicionalno se koriste dva nacina:

- **otvoreno adresiranje** – kolizije se razresavaju u okviru tabele
- **ulancavanje** – kolizije se razresavaju van tabele

Otvoreno adresiranje

Otvoreno adresiranje se svodi na pronalazenje neke druge lokacije u hes tabeli razlicite od maticne adrese kada se javi kolizija. Osnovna ideja je formulisati pravilo koje za svaki kljuc odredjuje ispitni niz kao niz adresa u tabeli koje se proveravaju pri umetanju novog kljuca. Ako treba da se umetne novi kljuc, a maticna adresa je zauzeta, onda se pokusava sa sledecom adresom u ispitnom nizu, sve dok se ne nadje slobodna lokacija.

Generisanje ispitnog niza za dati kljuc naziva se ***ponovnim hesiranjem*** jer se na osnovu zauzete adrese izracunava neka druga adresa sve dok se ne dodje do slaganja kljuca pri pretrazivanju ili do prve slobodne lokacije pri umetanju.

Pozeljno je da ispitni niz bude neka permutacija od skupa adresa (0…*n-1*) u tabeli sto omogucava da se u najgorem slucaju kolizije razrese posle *n* koraka.

Varijante otvorenog adresiranja su:

- linearno pretrazivanje
- slucajno pretrazivanje
- kvadratno pretrazivanje
- dvostruko hesiranje

//

44. (februar 2007.)

Dati pseudokod i objasniti opste algoritme umetanja i pretrazivanja kljuca u hes tabeli kod **tehnika otvorenog adresiranja**. Kakav se problem javlja kod brisanja i zasto?

odgovor:

```
HASH-INSERT(T, K)
i = 0
repeat
    j = h(K)
    if (T[j] = empty) then
        T[j] = K
        return j
    else
        i = i + 1
    end_if
until i = n
ERROR(Tabela puna)
```

```
HASH-SEARCH(T, K)
i = 0
repeat
    j = h(K)
    if (T[j] = K) then
        return j
    else
        i = i + 1
    end_if
until (T[j] = empty) or (i = n)
return empty
```

Umetanje kljuca K u tabelu T se realizuje funkcijom ***HASH–INSERT (T, K).*** Inicijalno je broj pokusaja pri umetanju jednak nula (***i=0***). Imamo jednu petlju koja generise ispitni niz.

U prvom prolazu pravimo maticnu adresu ***(j = h(K)).*** Potom proveravamo da li je lokacija prazna (***T[j] = empty***).

Ako je lokacija prazna, smestamo kljuc u istu (***T[j] = K***). Funkcija vraca adresu u tabeli na kojoj je kljuc smesten (***return j***).Ako lokacija nije prazna, idemo u sledeci pokusaj (***i = i + 1***).

Ukoliko se iscrpi citav niz (***until i = n***), a ne nadje se slobodna lokacija, javlja se poruka da je tabela puna (***ERROR (Tabela puna***)).

Pretrazivanje kljuca K u tabeli T se realizuje funkcijom ***HASH–SEARCH (T, K).*** Postupak je slican prethodnom. Hesiramo kljuc i dodjemo do adrese do koje bi dosli i da umecemo. Proverimo da li je kljuc tamo (***T[j] = K***), pa ako jeste, funkcija vraca adresu na kojoj je kljuc pronadjen (***return j***).

Ako nismo pronasli kljuc, idemo u rihesiranje (***i = i + 1***). Bitno je istaci da se generisanje ispitnog niza obavlja na isti nacin prilikom umetanja i prilikom pretrazivanja.

Neuspesno pretrazivanje se proglasava ako u ispitnom nizu dodjemo do prazne lokacije ili ako obidjemo sve ulaze tabele (***T[j] = empty or (i = n)***), a kljuc nije pronadjen. Tada funkcija vraca ***return empty.***

Problem brisanja se svodi na cinjenicu da se prilikom brisanja kljuc ne moze jednostavno ukloniti iz tabele i taj ulaz proglasiti slobodnim, jer se moze prekinuti ispitni niz do nekih drugih kljuceva.

Ovaj problem se resave tako sto se na mestu obrisanog kljuca ostavi poseban *kod (na primer, deleted)* koji je razlicit od vrednosti svih kljuceva, ali i od vrednosti *empty*. Ovo je takozvani metod “poluslobodne” lokacije. Sada se pri pretrazivanju ispitni niz ne prekida kada se dodje do lokacije *deleted*, vec se preskace kao da je zauzeta i ispitni niz dalje generise. Pri umetanju se ovakva lokacija tretira kao slobodna, tako da ona moze da se iskoristi za smestanje novog kljuca.

Mana ovog postupka je ta sto jednom zauzete lokacije vise nikada ne postaju slobodne, sto rezultira losim performansama pretrazivanja. Zato se ovaj metod koristi kada su brisanja u tabeli redja.

Postoji i **metod selektivnog pomeranja zapisa** kod kojeg se oslobodjena pozicija popuni. To je u principu slozena operacija.

//

Linearno pretrazivanje

//

42.d) (03.septembar 2008.)

Objasniti i analiticki definisati pojam linearnog pretrazivanja.

odgovor:

Ovaj metod pri koliziji generise ispitni niz sukcesivnih lokacija, a kad dodje do kraja tabele, vraca se na pocetak. Postupak ponovnog hesiranja se moze predstaviti na jedan od dva nacina:

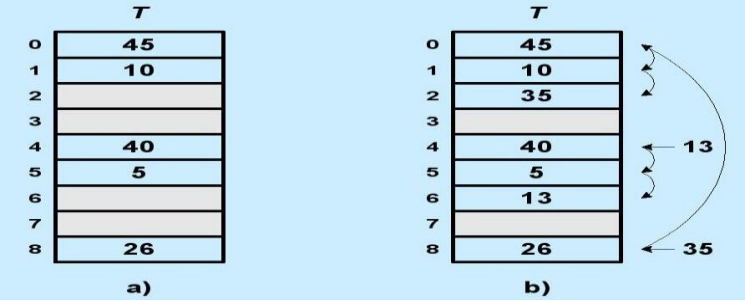
$$h_i(K) = (h_0(K) + i) \bmod n, \quad i = 1, 2, \ldots, n-1$$

$$h_i(K) = (h_{i-1}(K) + 1) \bmod n$$

Linearno pretrazivanje u slucaju kolizije pocinje sa sekvencijalnim trazenjem od maticne adrese i tezi da postavi kljuc sto blize svojoj maticnoj adresi.

//

Primer:



Neka je data prazna tabela sa 9 ulaza i hes funkcija h(K) = K mod 9. Kljucevi 5, 10, 26, 40, 45 se tada umecu bez kolizije na svoje maticne adrese (*slika a)*).

Medjutim kada dodje kljuc 13, njegova maticna adresa 4 je zauzeta, pa se pokusava na prvu sledecu, ali je i ona zauzeta jer se tu nalazi kljuc 5. Proba se sa sledecom adresom, ona je slobodna , tako da kljuc 13 popunjava ulaz 6.

Prilikom umetanja kljuca 35, njegova maticna adresa 8 je zauzeta, pa se pokusava na adrese 0, 1 i tek je lokacija 2 slobodna.

Selektivno pomeranje pri brisanju kod linearnog pretrazivanja:

Kod linearnog pretrazivanja je moguce premestanje kljuceva posle brisanja sa ciljem da se popuni obrisano mesto, a da se ne prekine ispitni niz ka kljucevima na narednim adresama.

Ako je brisanje izvrseno na lokaciji ***i***, tada se trazi prva sledeca prazna lokacija ***j*** po pravilu linearnog pretrazivanja. Zatim se pocevsi od adrese ***i+1,*** izracunavaju maticne adrese za kljuceve koji se nalaze izmedju lokacija ***i*** i **j**.

Neka se na nekoj takvoj lokaciji ***l*** nalazi kljuc K. Ako je ***r = h(K)*** maticna adresa tog kljuca i ako se ona ne nalazi izmedju ***i*** i ***j*** (***r < i*** ili ***r > j***), onda je moguce izvrsiti premestanje kljuca sa tekuce lokacije na lokaciju ***i*** i time osloboditi tekucu poziciju.

Ako se ***r*** nalazi izmedju ***i*** i ***j,*** onda se kljuc ne moze premestiti jer bi se time on postavio ispred svoje maticne adrese sa koje pocinje pretrazivanje na njega.

Posto je sada prazna lokacija presla na adresu ***l,*** postupak se zatim sikcesivno ponavlja sa lokacijama izmedju ***l*** i ***j***, sa ciljem da ova slobodna pozicija "izadje" na adresu ***j-1***.

Ovakvim premestanjem se popravljaju performanse pretrazivanja.

Primarno grupisanje

//

42.b) (03.septembar 2008.)

Objasniti i analiticki definisati pojam ***primarnog grupisanja*** (i kako se ono izbegava).

odgovor:

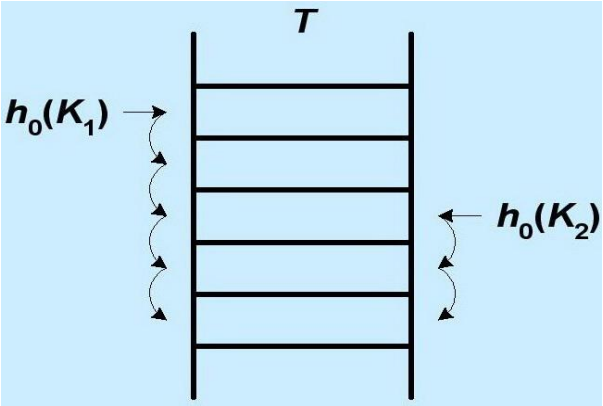
Glavna prednost linearnog pretrazivanja je jednostavnost, ali je glavni nedostatak pojava ***primarno grupisanje*** koja se obicno javlja kod vece popunjenosti tabele.

Analiticki se definise kao:

$$h_{i+r}(K_1) = h_r(K_2) \quad \text{za } r = 0, 1, 2, \ldots$$

Ako imamo ispitni niz od nekog kljuca K_1 pa u nultom pokusaju dodjemo do maticne adrese, u prvom pokusaju dodjemo do neke sledece i tako redom. Ako se sada desi da maticna

adresa nekog drugog kljuca K_2 upadne na neko mesto u ispitni niz prvog kljuca i od tog trenutka se oni poklapaju, onda se ta pojava naziva ***primarno grupisanje***.



Ova pojava nije dobra jer ce sinonimi od K_1 i K_2 zavrsavati na istim mestima i onda ce se ispitni niz produzavati pa ce se stvarati grupe zauzetih lokacija i nece se kljucevi razbacivati po samoj tabeli.

Kako resiti problem primarnog grupisanja?!

Jednostavan nacin da se ovo izbegne je da hes funkcija bude zavisna i od broja pokusaja:

$$h_i(K) = (h_{i-1}(K) + i) \bmod n,\ i = 1, 2, \ldots$$

//

42.e) (03.septembar 2008.)

Objasniti i analiticki definisati pojam ***kvadratnog pretrazivanja*** i objasniti njegove prednosti.

odgovor:

Jos bolji nacin da se izbegne primarno grupisanje je primena tehnike ***kvadratnog pretrazivanja*** kod koje se ispitni niz generise preko kvadrata broja neuspesnih pokusaja:

$$h_i(K) = (h_0(K) + i^2) \bmod n,\ \ i = 1, 2, \ldots, n-1$$

Ovaj niz se moze generisati i bez dodatnih operacija kvadratiranja ako se ima u vidu da je:

$$i^2 = d_i = d_{i-1} + 2i + 1$$

Osnovni problem kod kvadratnog pretrazivanja je sto ispitni niz u opstem slucaju ne sadrzi svaku adresu u tabeli, pa moze da se desi slucaj pri umetanju da postoji slobodna lokacija, a da na ovakav nacin njoj ne moze da se pridje.

Ako je ***n*** prost broj, garantuje se da ispitni niz sadrzi bar ***n/2*** adresa. Ako je ***n*** prost broj oblika ***4j + 3***, garantuje se da ce ispitni niz da sadrzi svaku adresu u tabeli.

//

Sekundarno grupisanje

//

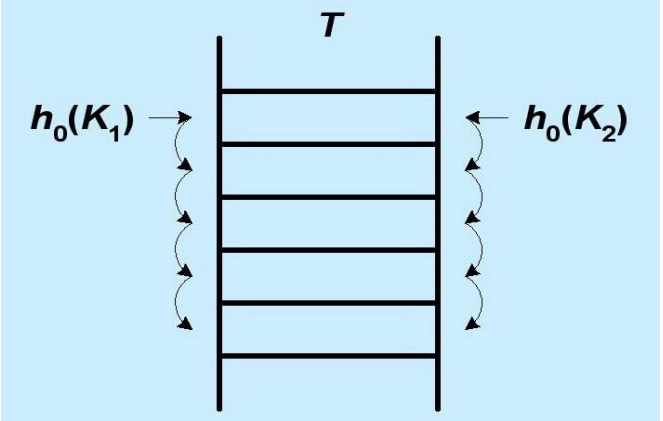
42.c) (03.septembar 2008.)

Objasniti i analiticki definisati pojam ***sekundarnog grupisanja*** i kako se ovaj problem resava.

odgovor:

Ovaj fenomen se javlja kada dva kljuca imaju istu maticnu adresu, pa prethodni metodi generisu i potpuno iste ispitne nizove.

$$h_i(K_1) = h_i(K_2) \quad \text{za } i = 0, 1, 2, \ldots$$



Problem sekundarnog grupisanja se drasticno smanjuje metodom dvostrukog hesiranja.

//

Dvostruko heširanje

//

42.f) (03.septembar 2008.)

Objasniti i analiticki definisati pojam ***dvostrukog heširanja.***

41. (14.februar 2007.)

Objasniti da li se primarno i sekundarno grupisanje izbegava kod dvostrukog hesiranja.

odgovori:

Ovaj metod se zasniva na primeni dve nezavisne hes funkcije: primarne ***h(K)*** i sekundarne ***g(K)*:**

$$h_i(K) = (h_{i-1}(K) + g(K)) \bmod n$$

ili

$$h_i(K) = (h_0(K) + i\,g(K)) \bmod n, \quad i = 1, 2, \ldots, n-1$$

Na zadati kljuc se prvo primenjuje primarna hes funkcija, pa ako je doslo do kolizije, na vrednost kljuca se primenjuje i sekundarna hes funkcija koja onda izracunava pomeraj u ispitnom nizu.

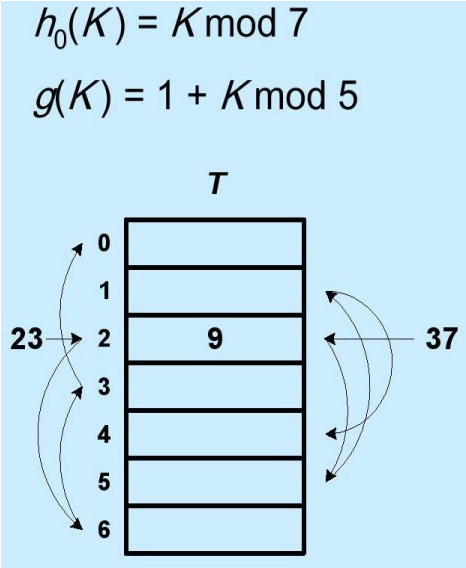
Dvostruko hesiranje veoma smanjuje verovatnocu sekundarnog grupisanja za nezavisne funkcije h_0 i g (ne otklanja skroz).

Sekundarna hes funkcija mora da bude pazljivo izabrana da bi ispitni niz sadrzao sve adrese. Zato ***g(K)*** mora da daje rezultat u opsegu ***1*** do ***n-1*** i da bude uzajamno prost sa ***n***.

Knuth predlaže
- ✓ $h_0(K) = K \bmod n$
- ✓ $g(K) = 1 + K \bmod (n-2)$
- ✓ n i $n-2$ prosti brojevi

//

Primer dvostrukog heširanja:



Kljucevi 23 i 37 su sinonimi sa istom maticnom adresom 2 jer je:

- 23 mod 7 = 2
- 37 mod 7 = 2

Kada je ona zauzeta (u ovom primeru kljucem 9), dvostruko hesiranje generise dva razlicita ispitna niza. To se radi po principu:

(“dobijen ulaz koji nije slobodan” + *g(K)*) mod *n* (gledas *n* iz primarne heš funkcije, u nasem primeru je to 7) :

Ispitni niz za kljuc 23 (pored lokacije 2) cine:

- (2 + 1 + 23 mod 5) mod 7 = 6 mod 7 = 6
- (6 + 1 + 23 mod 5) mod 7 = 10 mod 7 = 3
- (3 + 1 + 23 mod 5) mod 7 = 7 mod 7 = 0 itd.

Ispitni niz za kljuc 37 (pored lokacije 2) cine:

- (2 + 1 + 37 mod 5) mod 7 = 5 mod 7 = 5
- (5 + 1 + 37 mod 5) mod 7 = 8 mod 7 = 1
- (1 + 1 + 37 mod 5) mod 7 = 4 mod 7 = 4 itd.

Poboljsanja metoda otvorenog adresiranja

Kada je zauzetost tabele velika broj poredjenja pri pretrazivanju raste, sto znatno usporava ovu operaciju. Zato uvodimo dve metode poboljsanja gde ***metoda uredjene tabele*** smanjuje poredjenja pri neuspesnom pretrazivanju, a ***Brent-ov metod*** smanjuje broj poredjenja pri uspesnom pretrazivanju.

Metod uredjene tabele

Ovaj metod pri umetanju odrzava skup kljuceva koji se mapiraju na istu maticnu adresu uredjenim po opadajucim vrednostima kljuceva. Zbog toga, kada se u operaciji umetanja novog kljuca K pri trazenju praznog mesta u ispitnom nizu dodje do prve lokacije na kojoj se nalazi kljuc K' < K, kljuc K se umece na njegovo mesto, a zatim se nastavlja sa trazenjem mesta za K na isti nacin.

Kada se koristi?!

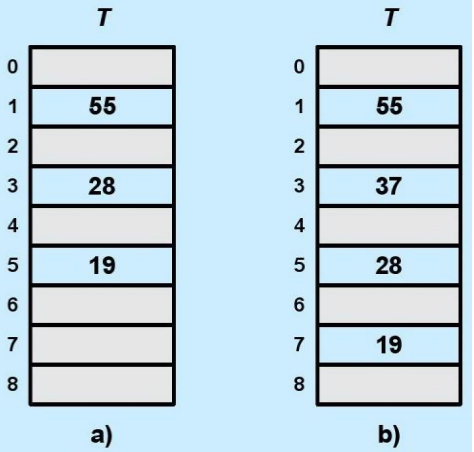
Moze se primeniti na sve tehnike kod kojih se generisanje sledece adrese u ispitnom nizu vrsi samo na osnovu prethodne adrese i vrednosti kljuca, ali ne i na osnovu prethodnog broja pokusaja.

Performanse

Znacaj rezultata ovog metoda je sto izjednacava broj poredjenja pri uspesnom i neuspesnom pretrazivanju, ali kvari performanse umetanja, pa je bolji za staticke tabele koje cesto pretrazuju.

Primer

$h_i(K) = (K + 2i) \bmod n$



Na slici a) je prikazano stanje nakon umetanja kljuceva 55, 19 i 28 tim redom. Svi imaju istu maticnu adresu 1. Kada se umece kljuc 37 (*slika b)*), koji je sinonim ovih kljuceva, on se ne stavlja na kraj ispitnog niza (1, 3, 5, 7…) vec se umece ispred kljuca 28. Na taj nacin se odrzava skup kljuceva koji se mapiraju na istu maticnu adresu po opadajucim vrednostima kljuceva. Ako bi nakon ovoga, pretrazivali tabelu na kljuc 46, ne moramo proveriti sve lokacije vec odmah posle provere na adresi 3 (37 < 46) utvrdjuje se da ovaj kljuc nije u tabeli. Dakle, dobitak je taj sto ne moramo da idemo do kraja ispitnog niza.

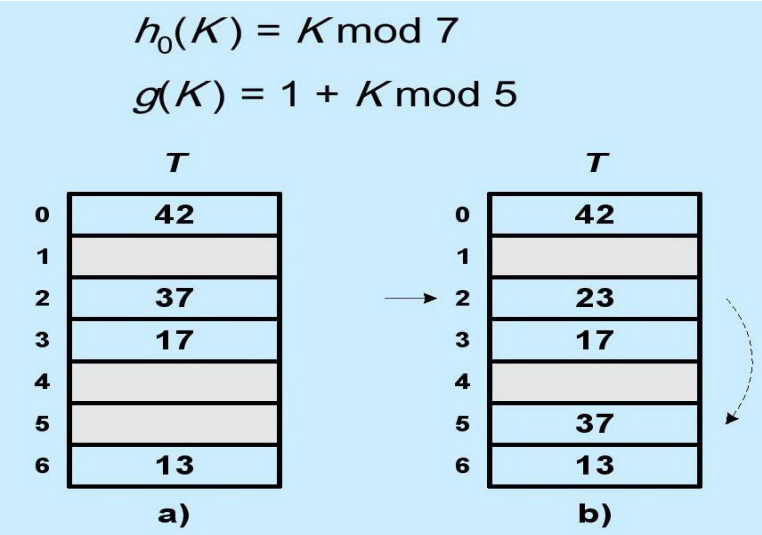
Brent-ov metod

Ovaj metod smanjuje broj poredjenja pri uspesnom pretrazivanju, a nema uticaja na neuspesno pretrazivanje. Zasniva se na preuredjenju hes tabele i koristi se pri dvostrukom hesiranju. ***Osnovna ideja je da se premestanjem nekog vec umetnutog kljuca oslobodi mesto za novi kljuc, a da ukupan prosecan broj poredjenja pri trazenju jednog i drugog kljuca bude smanjen.***

Kada se koristi?!

Ovaj metod je pogodan za staticke tabele gde nema mnogo umetanja.

Primer



Pocetno stanje tabele dato je na slici a). Mi sada zelimo da umetnemo kljuc 23. Njegov ispitni niz cine lokacije 2, 6, 3, 0 i sve su one popunjene. Kljuc 23 bi mogao da se postavi tek na prvu slobodnu lokaciju, a to je adresa 4. U ovoj varijanti bi za uspesno pretrazivanje kljuca 23 bilo potebno 5 poredjenja (2, 6, 3, 0, 4), a za kljuc 37 samo jedno predjenje, a u proseku 3 poredjenja po kljucu.

Ako bi primenili ***Brent-ov*** metod pa kljuc 23 postavili na maticnu adresu 2 (23 mod 7 = 2), a kljuc 37 pomerili na prvu sledecu lokaciju koja je slobodna iz njegovog ispitnog niza (2, 5, 1…), a to je adresa 5, tada bi pri uspesnom pretrazivanju ova dva kljuca bilo potrebno ukupno 3 poredjenja (jedno za kljuc 23 i dva za kljuc 37), a u proseku 1.5 poredjenja.

Ulancavanje

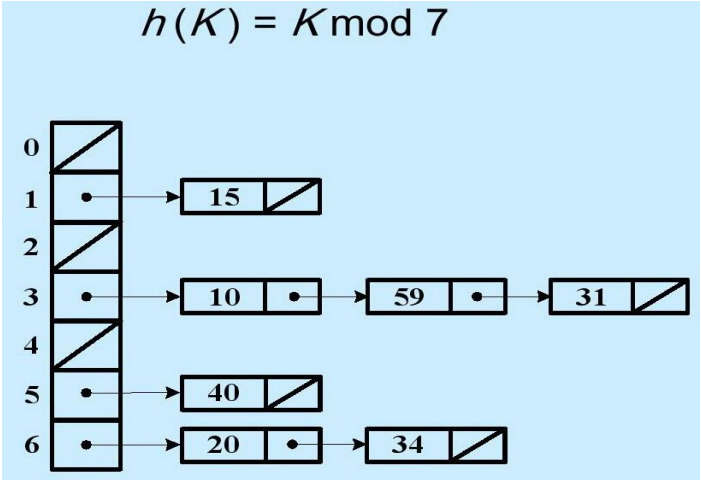
Tehnike otvorenog adresiranja imaju dva nedostatka. Prvi problem nastaje u primenama gde ima dosta brisanja, jer se zauzete lokacije ne oslobadjaju, a premestanje moze biti vrlo slozeno, ako je uopste moguce.

Drugi problem je neefikasno pretrazivanje kod tabele sa vecom popunjenoscu, jer se u ispitnom nizu mogu naci i adrese zauzete kljucevima koji nisu sinonimi trazenog kljuca. Pomenuti problemi se efikasno resavaju drugom tehnikom koja se zove ***ulancavanje.***

Odvojeno ulancavanje

Sustina ove tehnike je da se sinonimi ulancaju u liste cija se zaglavlja nalaze na maticnim adresama. Zauzeti ulaz hes tabele sadrzi samo pokazivac na listu sinonima koji odgovaraju toj maticnoj adresi. Posto svaka lista odgovara samo jednoj klasi ekvivalencije, ova varijanta ulancavanja se naziva odvojeno ulancavanje.

Primer



Za kljuc 15 maticna adresa je 1, a za kljuceve 10, 59 i 31 maticna adresa je 3 pa su oni ulancani. Za kljuc 40 maticna adresa je 5, a za kljuceve 20 i 34 maticna adresa je 6 pa su oni ulancani.

//

43. (24.septembar 2008.)

Koje su prednosti, a koji nedostaci odvojenog ulancavanja u odnosu na otvoreno adresiranje?

odgovor:

Prednosti:

Operacije sa kljucevima kod tehnike ulancava su jednostavnije nego kod otvorenog adresiranja.

Pretrazivanje na zadati kljuc se svodi na prolazak kroz ulancanu listu cije se zaglavlje nalazi na maticnoj adresi. Ako se citava lista prodje, a kljuc se ne nadje, pretrazivanje je neuspesno.

Umetanje novog kljuca moze da se vrsi na pocetak liste, ako se zna da kljuc nije u tabeli, ili na kraj liste, ako treba prvo proveriti da li je kljuc unesen, jer se tu zavrsava neuspesno pretrazivanje. Ako se lista sinonima drzi uredjena po vrednosti kljuca, tada je pretrazivanje efikasnije, ali je umetanje manje efikasno jer se kljuc mora postaviti na mesto koje mu po poretku odgovara.

Brisanje je veoma jednostavno jer se svodi na prevezivanje liste. Zbog brisanja je pogodno da lista bude dvostruko ulancana.

Moze se ocekivati da prosecne duzine lista budu mala, pa su operacije sa hes tabelom brze. Kada bi liste bile dugacke, onda bi bilo efikasnije da se umesto liste koristi binarno stablo pretrazivanja.

Dodatna prednost je i to sto ***tehnika ne ogranicava broj kljuceva*** kao kod otvorenog adresiranja gde je broj kljuceva ogranicen velicinom kontinualne tabele.

Nedostaci:

Sve ove prednosti se placaju dodatnim prostorom za pokazivace i vecim brojem praznih ulaza u hes tabeli.

//

Objedinjeno ulancavanje

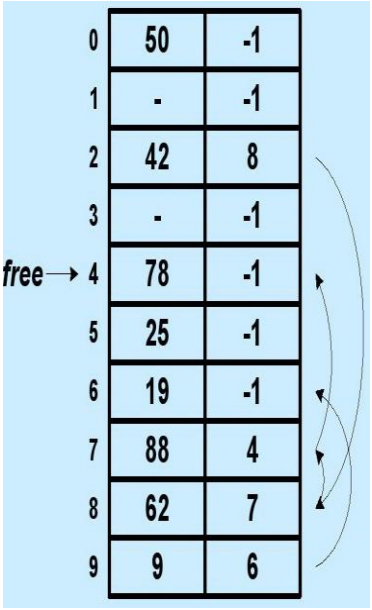
Sto je veca hes tabela, liste su u proseku krace, a performanse bolje. Sa povecanjem hes tabele, raste i broj nezauzetih ulaza u hes tabeli. Zato se javlja ideja da se taj prostor iskoristi za smestaj elemenata liste, sto poboljsava efikasnost koriscenja prostora sa dva aspekta: koristi nezauzete ulaze hes memorije i izbegava dodatni prostor za smestaj clanova liste. Ovakva varijanta ulancavanja se naziva ***standardno objedinjeno hesiranje.***

Ovaj metod predstavlja hibridno resenje izmedju otvorenog adresiranja i odvojenog ulancavanja, jer kljuceve smesta samo u osnovnoj tabeli, kao otvoreno adresiranje, ali ne koristi ponovno hesiranje u slucaju kolizije vec direktne pokazivace na sledecu adresu u ispitnom nizu.

Svaki ulaz hes tabele treba da pored kljuca key sadrzi i polje next koje sluzi za cuvanje adrese ulaza u tabeli gde se nalazi eventualni naredni clan liste sinonima. U pocetnom stanju u svim ulazima polje kljuca ima pocetno stanje empty, a polje next je -1, sto je analogno praznom pokazivacu.

Objedinjeno ulancavanje bolje koristi prostor u odnosu na odvojeno ulancavanje, ali su performanse losije. Nepovoljan efekat ove metode je sto dopusta objedinjavanje lista za vise klasa ekvivalencije. U primeru koji sledi kljucevi 42, 62, 78 i 88 se nalaze u jednoj listi, a to su kljucevi iz dve klase ekvivalencije za maticne adrese 2 i 8. Spajanje se desilo kada je kljuc 62 postavljen na lokaciju 8 koja predstavlja maticnu adresu za kljuceve 88 i 78, jer je njegova maticna adresa 2 vec bila popunjena kljucem 42.

Primer



Hes tabela ima 10 ulaza, a hes funkcija je oblika ***h(K) =K mod 10.***

U tabelu se redom umecu kljucevi : **42, 9, 25, 62, 88, 50, 19, 78.**

- **h(42)** = 42 mod 10 = 2; **ulaz 2: 42; next 2: - 1;**
- **h(9)** = 9 mod 10 = 9; **ulaz 9: 9; next 9: - 1;**
- **h(25)** = 25 mod 10 = 5; **ulaz 5: 25; next 5: - 1;**
- **h(62)** = 62 mod 10 = 2; ulaz 2 je zauzet pa kljuc smestamo na prvu slobodnu lokaciju sa kraja na koju ukazuje ***pokazivac free***, a to je **ulaz 8**; **next 2: 8;**
- **h(88)** = 88 mod 10 = 8; ulaz 8 je zauzet pa kljuc smestamo na prvu slobodnu lokaciju sa kraja na koju ukazuje, a to je **ulaz 7**; **next 8: 7;**
- **h(50)** = 50 mod 10 = 0; **ulaz 0: 50; next 0: -1;**
- **h(19)** = 19 mod 10 = 9; ulaz 9 je zauzet pa kljuc smestamo na prvu slobodnu lokaciju sa kraja, a to je **ulaz 6; next 9: 6;**
- **h(78)** = 78 mod 10 = 8; ulaz 8 je zauzet pa kljuc smestamo na prvu slobodnu lokaciju sa kraja, a to je **ulaz 4; next 7: 4;**

Komparativno poredjenje tehnika unutrasnjeg hesiranja

U cilju poredjenja performansi, u narednoj tabeli su prikazani brojevi pokusaja pri uspesnom i neuspesnom pretrazivanju za odredjene vrednosti popunjenosti (α) hes tabele kod:

- **linearnog pretrazivanja (LP)**
- **dvostrukog hesiranja (DH)**
- **odvojenog ulancavanja (SC)**

	S			U		
α	**LP**	**DH**	**SC**	**LP**	**DH**	**SC**
0.1	1.056	1.054	1.050	1.118	1.111	1.005
0.2	1.125	1.116	1.100	1.281	1.250	1.019
0.3	1.214	1.189	1.150	1.520	1.429	1.041
0.4	1.333	1.277	1.200	1.889	1.667	1.070
0.5	1.500	1.386	1.250	2.500	2.000	1.107
0.6	1.750	1.527	1.300	3.625	2.500	1.149
0.7	2.167	1.720	1.350	6.060	3.333	1.197
0.8	3.000	2.012	1.400	13.000	5.000	1.249
0.9	5.500	2.558	1.450	50.500	10.000	1.307
0.95	10.500	3.153	1.475	200.50	20.000	1.337

Za manje popunjenosti tabele, do 50%, performanse ovih metoda su dosta bliske, pogotovo pri uspesnom pretrazivanju.

Razlike u performansama su izuzetno vidljive kada je tabela prilicno popunjena (oko 95%). Tehnika odvojenog ulancavanja pokazuje najbolje performanse jer se moze ocekivati da se u 1.5 poredjenja nadje kljuc. Ocigledno je da performanse ove metode najmanje variraju sa povecanjem popunjenosti. Mana tehnike odvojenog ulancavanja je potreba za dodatnim prostorom, ima vise nezauzetih ulaza u memoriji, a i pokazivaci predstavljaju dodatnu cenu.

Kako popunjenost raste, linearno pretrazivanje pokazuje najslabije performanse. Ipak, valja podsetiti da otvoreno adresiranje za razresavanje kolizije koristi samo osnovnu tabelu, dok odvojeno ulancavanje koristi dinamicki prostor van tabele.

Performanse objedinjenog hesiranja nisu mnogo losije od odvojenog ulancavanja, pogotovo za manju zauzetost tabele, a koriscenje prostora je svakako efikasnije.

Sto se tice izbora hes funkcije, ako raspodela kljuceva nije unapred poznata, metod deljenja se pokazao najprakticnijim.

Na kraju, moze se zakljuciti da je hesiranje dosta efikasna tehnika, ali je jedna od mana sto ne moze da nam garantuje maksimalnu duzinu ispitnog niza.

Kod otvorenog adresiranja imamo dodatni problem sto velicina hes tabele ogranicava broj kljuceva. Ako se tabela napuni, a nailaze i dalje kljucevi, resenje moze da bude u povecanju tabele za odredjeni procenat i ponovnom hesiranju svih kljuceva.

Hesiranje po pravilu ne omogucava pristup u poretku. To znaci da ako imamo kljuceve u hes tabeli i sada zelimo da im pristupimo u uredjenom poretku od najmanjeg do najveceg, to nije lako. Neke hes funkcije, ipak cuvaju taj poredak. U tom slucaju je sekvencijalno pretrazivanje vrlo olaksano. Medjutim, problem je sto te hes funkcije obicno nisu uniformne.

Spoljašnje heširanje

//

46. (*ispitno pitanje*)

Objasniti glavne specificnosti postupka spoljasnjeg hesiranja u odnosu na unutrasnje.

odgovor:

Hesiranje se kao tehnika pretrazivanja koristi i za pristup podacima na spoljasnjim memorijama, najcesce u datotekama sa direktnim pristupom.

Za spoljasnje hesiranje se u osnovi mogu koristiti veoma slicne hes funkcije i metode razresenja kolizije kao i kod unutrasnjeg hesiranja. Medjutim, karakteristike pristupa disku se drasticno razlikuju od pristupa operativnoj memoriji, sto uslovljava neke specificnosti primene tehnike hesiranja na spoljasnjim medijumima.

Pristup disku je daleko najskuplja operacija pa se pri spoljasnjem hesiranju mnogo vise vodi racuna o vremenu nego o prostoru pogotovo kad se zna da je prostor na disku veoma jeftin. Zato je jako vazno smanjiti broj kolizija i pristupa disku, cak i na racun nesto slabijeg iskoriscenja prostora.

Glavna razlika izmedju spoljasnjeg hesiranja i unutrasnjeg hesiranja je u organizaciji hes tabele. ***Kod unutrasnjeg hesiranja smo koristili hes tabele kod kojih jedan ulaz sadrzi jedan kljuc.***

Kod spoljasnjeg hesiranja ulaz hes tabele moze da sadrzi vise od jednog kljuca. Pri tome se ulazi hes tabele organizuju kao baketi gde jedan baket predstavlja broj podataka koji se prihvata jednim pristupom disku.

Zbog velicine baketa moguce je da hes tabela sadrzi citave zapise, a ne samo kljuceve. U tom slucaju hes tabela ne postoji kao odvojena indeksna struktura, vec je ona prakticno objedinjena sa primarnom oblascu datoteke.

Time se izbegava dodatni pristup disku za citanje, ali se smanjuje kapacitet baketa sto moze dovesti do povecanja verovatnoce kolizije.

//

Standardne tehnike spoljašnjeg heširanja

//

47. (*ispitno pitanje*)

Objasniti standardne tehnike za razresavanje kolizije kod spoljasnjeg hesiranja. Uporediti njihove performanse.

odgovor:

Neka se hes tabela sastoji od ***n* baketa** i neka svaki baket moze da sadrzi ***b* zapisa** sa svojim kljucevima. Tada hes tabela sadrzi ***nb* kljuceva,** a popunjenost se racuna kao:

$$\alpha = m / nb$$

gde je ***m* broj kljuceva** koji se trenutno nalaze u hes tabeli. Nas cilj je da proredimo broj pristupa pa je prema tome uniformnost kriticna. Ako hes funkcija nije uniformna, onda ona opterecuje pojedine ulaze vise i time uslovljava da se pristupa vise puta. Posto u jedan ulaz moze da ide vise od jednog, a u ovom slucaju do ***b* kljuceva,** onda umetanje do ***b* kljuceva** u jedan ulaz ne proizvodi koliziju. Zato se ovo ponekad smatra posebnom tehnikom za smanjenje kolizije i naziva se ***adresiranjem baketa.***

Kad treba da se ***umetne novi zapis u datoteku***, kljuc se transformise primenom hes funkcije, pa se dobije adresa maticnog baketa. Ovaj baket se ucita u operativnu memoriju i ukoliko ima manje od ***b*** zauzetih mesta, novi zapis sa svojim kljucem se ubacuje u njega.

Ukoliko je baket pun, onda treba da se primeni neka tehnika za razresavanje kolizije jer kljuc ne moze da stane u svoj maticni baket pa mu treba naci mesto u nekom drugom baketu.

Trazenje zapisa sa datim kljucem pocinje ucitavanjem maticnog baketa koji se zatim pretrazuje u memoriji. Ako je ***b*** manje, obicno se isplati sekvencijalno pretrazivanje. Za vece ***b,*** moze da se primeni i binarno pretrazivanje ako su kljucevi u bakatu uredjeni. Ako se kljuc ne pronadje u maticnom baketu, ucitava se drugi baket po istom principu koji je primenjen za razresavanje kolizije kod umetanja. Neuspesno pretrazivanje se proglasava ako baket nije pun, a kljuc se ne nadje u njemu.

Kod ***brisanja***, ukoliko je baket pun, ne mozemo bas slobodno da brisemo kljuc iz njega. Moramo da ga eventualno dopunimo sa kljucem iz sledece lokacije u ispitnom nizu. Ako baket nije pun, mozemo slobodno da brisemo.

Kolizije se kod spoljasnjeg hesiranja mogu razresavati primenom istih tehnika kao kod unutrasnjeg, a to su otvoreno adresiranje ili ulancavanje.

Analize pokazuju da za manji kapacitet baketa imamo slicnu situaciju kao kod unutrasnjeg hesiranja. Tada je linearno pretrazivanje najslabije. Dvostruko hesiranje ima najbolje performanse za iole vece bakete. Mada su generalno gledano performanse prilicno ujednacene za vece kapacitete baketa.

Treba voditi racuna o cinjenici da pored broja citanja baketa, vazna je i njihova relativna pozicija zbog vremena pozicioniranja glava na disku. Svakako je brze kada se citaju dva fizicki susedna baketa nego ako su oni fizicki udaljeni. Ova cinjenica istice u prvi plan ***linearno pretrazivanje,*** koje kod kolizije pokusava sa sledecim, susednim baketom.

Kod ***odvojenog ulancavanja***, kolizije se razresavaju u posebnoj oblasti prepunjenja gde se smestaju zapisi koji ne stanu u maticni baket. Ova oblast moze da se organizuje u posebne bakete za svaku listu, ali i ne mora, jer nije previse verovatno da ce posle prepunjenja nekog maticnog baketa biti dovoljno kljuceva da ispune jos jedan baket. Zato se u oblasti prepunjenja mogu ulancavati individualni zapisi u okviru zajednickih baketa.

Objedinjeno ulancavanje ne koristi posebnu oblast popunjenja vec kolizije razresava ulancavanjem u baketima u primarnoj oblasti koji nisu puni. U tom cilju se odrzava lista baketa koji imaju slobodnih mesta. Kada se baket napuni, onda se on iskljucuje iz ove liste.

Nemogucnost sekvencijalnog pristupa zapisima po uredjenom poretku njihovih kljuceva prisutna kod metoda unutrasnjeg hesiranja, karakterise i spoljasnje hesiranje.

//

Fleksibilne tehnike spoljašnjeg heširanja

Datoteke su relativno dugovecne strukture, ali tokom svog trajanja one mogu da se povecavaju ili smanjuju na nepredvidljiv nacin. U takvim uslovima su metode spoljasnjeg hesiranja nefleksibilne. Velicina hes tabele se projektuje prema nekom predvidjenom broju zapisa, sa namerom da se postigne odredjena popunjenost, a time i zeljene performanse. Medjutim, ako tokom rada broj zapisa poraste mnogo iznad predvidjenog broja, popunjenost raste, a performanse se pogorsavaju. Ako broj zapisa dosta opadne, onda se prostor neefikasno koristi buduci da je velicina hes tabele fiksna.

Prema tome, pozeljno bi bilo uciniti spoljasnje hesiranje fleksibilnijim. U tom cilju su razvijene tehnike ***dinamickog, prosirljivog i linearnog hesiranja***. Mi cemo na ovom kursu govoriti o dinamickom i prosirljivom hesiranju.

Dinamičko heširanja

12.nedelja, 3.cas, [14:15 – 22:35]

Proširljivo heširanja

12.nedelja, 3.cas, [22:35 – 33:40]