Datum: 29.11.2018.

Hešing i ATP tabela

U ovom materijalu je ukratko prikazana ATP tabela i upoređene su njene tri reprezentacije:

- strukturni niz u rastućem sekvencijalnom redosljedu po ključevima,
- AVL stablo i
- heš tabela sa upotrebom dvostrukog heširanja.

Ovo upoređivanje će otkriti prednosti i mane heširanja.

ATP tabela

Tabela T je apstraktno sredstvo u kome se čuvaju slogovi tabele koji su ili prazni, ili uređeni parovi sa formom (K,I), gdje je K jedinstveni ključ, a I informacija povezana sa K. Različiti slogovi tabele imaju različite ključeve.

ATP tabela je definisana slijedećim operacijama:

- Iniciranje tabele T da bude prazna tabela. Prazna tabela je ispunjena praznim slogovima (K0,I0), gdje je K0 poseban, prazan ključ, različit od svih ostalih nepraznih ključeva.
- Određivanje je li tabela T puna.
- Umetanje novog sloga (K,I) u tabelu T, pod uslovom da T nije puna.
- Brisanje sloga (K,I) iz tabele T.
- Kada je dat ključ K, pretraživanje tabele u potrazi za K i preuzimanje informacije iz sloga (K,I) tabele T.
- Ažuriranje sloga (K,I) tabele T, zamjenom sa novim slogom (K,I'), kojim se sa K povezuje nova informacija I'.
- Numerisanje slogova (K,I) tabele T u rastućem poretku po ključevima K.

Jedan primjer apstraktne tabele je naredna tabela, gdje u svakom slogu (K,I), troslovni aerodromski kod predstavlja jedinstveni ključ K, a informacija I objašnjava u kom gradu i državi se nalazi aerodrom sa datim kodom, K.

Ključ K: aerodromski kod	**Informacija I: Informacija o aerodromu**
AKL	Auckland, New Zealand
DCA	Washington, D.C.
FRA	Frankfurt, Germany
GCM	Grand Cayman, Cayman Islands
GLA	Glasgow, Scotland
HKG	Hong Kong, China
LAX	Los Angeles, California
ORY	Paris, France
PHL	Philadelphia, Pennsylvania

Heširanje

Heširanje (engleski Hashing) je nastalo iz potrebe da se ubrza pretraživanje tabele. Ukoliko niz ima n elemenata, i treba se pronaći određena vrijednost, onda se ili mora pretražiti čitav niz (ukoliko je nesortiran) ili (ukoliko je niz sortiran u na primjer binarno stablo za pretraživanje) može pretražiti samo određena grana. Sortiranjem u stablo se vrijeme u najgorem slučaju smanjuje na logaritamsko (O(logn)). No, ukoliko se zna indeks elementa niza u koji je smješten traženi sadržaj, vrijeme za pretraživanje postaje konstantno (O(1)). Heširanje je u tom smislu proces u kome se ključevima tabele (u opštem slučaju ključevi mogu biti alfanumerički) pridodaju numeričke vrijednosti (na određeni, sistematičan, način) i time omogućava da se ključevi nađu u konstantnom vremenu (ukoliko je metoda povoljno odabrana i ne dođe do velike klasterizacije). Primjer ključa koji se heširа i jednostavno nalazi je bibliotečki broj knjige; ukoliko se heširа i heš vrijednost smjesti u heš tabelu, pretraga za brojem se provodi samo po heš tabeli, a ona obično podrazumjeva mali broj koraka.

Uvod u heširanje pomoću jednostavnih primjera

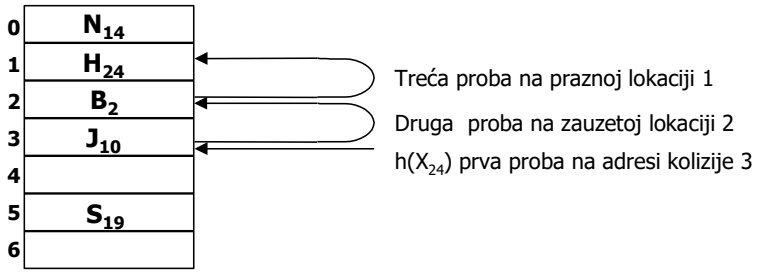
Za uvođenje osnovnih pojmova hešinga ćemo razmotriti nekoliko jednostavnih primjera. U ovim primjerima ćemo za ključeve koristiti slova engleskog alfabeta indeksirana brojevima (kao A_1, B_2, C_3, R_{18}, Z_{26}), gdje indeks slova označava njegovu poziciju u alfabetu. Tabela T, koju ćemo koristiti je namjerno izabrana da bude mala: T ima mjesta za smještanje samo 7 slogova koji se smještaju u redove tabele T numerirane od 0 do 6. Radi jednostavnosti ćemo pretpostaviti da slog sadrži samo ključ, a ne i informaciju.

Pošto imamo sedam mjesta u tabeli T, da bi odredili u koji red ćemo smjestiti ključ K_i, podjelićemo vrijednost indeksa i sa 7. Ostatak djeljenja[1] i/7 nam određuje u koji red da smjestimo ključ K_i. Tako se ključevi B_2, J_{10} i S_{19} smještaju u redove 2, 3 i 5 respektivno. Funkcija kojom se određuje lokacija L_n na koju smještamo ključ sa indeksom n je

$$h(L_n) = n \% 7.$$

Funkciju $h(L_n)$ zovemo **heš funkcija** ključa L_n. Dobra heš funkcija, $h(L_n)$ će preslikati ključeve L_n uniformno i slučajno na cijeli raspon mogućih lokacija (0:6) u tabeli T. Sada ćemo pokušati da umetnemo ključeve N_{14}, X_{24} i W_{23} u tabelu T. Ključ N_{14} se može umetnuti direktno u tabelu, na lokaciju $h(N_{14})=0$. Kada pokušamo da umetnemo ključ X_{24} na poziciju $h(X_{24})=3$, vidimo da je u red 3 već smješten ključ J_{10}. Ovakva situacija se naziva **kolizija**, jer dva ključa dolaze u koliziju (treba da se smjeste na istu lokaciju u tabeli) pošto imaju iste heš adrese, $h(J_{10})=h(X_{24})=3$. Potrebno je odrediti na koji način će se razrješavati ovakve situacije. Jednostavan način za rješavanje kolizije je da se u tabeli T potraži prva slobodna niža lokacija i u nju smjesti dati ključ. Na primjer, pošto je $h(X_{24})=3$ zauzeta, provjerimo je li slobodna lokacija 2, pa pošto ni ona nije slobodna, provjerimo je li slobodna lokacija 1. Kako je lokacija 1 slobodna, ključ X_{24} se smješta u prvi red tabele i to iz trećeg pokušaja (vidi sliku).

[1] Račun po modulu 7



Na kraju ćemo umetnuti ključ W_{23}. Prvi pokušaj umetanja W_{23} na njegovu heš adresu, $h(W_{23})=2$, dovodi do kolizije jer je u red 2 već smješten ključ B_2. Zato provjeravamo redom lokacije 1 i 0, koje su obe zauzete. Zatim "zaokrenemo" i pokušamo sa poslijednjom lokacijom u tabeli. Kako je lokacija 6 slobodna, smještamo ključ W_{23} na lokaciju 6 u tabeli T.

Lokacije koje ispitujemo prilikom pokušaja umetanja novog ključa u tabelu T se nazivaju **probni niz**. (Probni niz za ključ W_{23} je 2,1,0,6,5,4,3.) Probni niz je tako određen da ispituje svaku lokaciju tabele T tačno jednom. Da bi obezbjedili da uvijek pronađemo praznu lokaciju primjenom probnog niza, definiraćemo punu tabelu, T, takvu da T sadrži tačno jedan prazan red. Na ovaj način obezbjeđujemo uspješnu potragu za praznim lokacijama i ne moramo prebrojavati lokacije u probnom nizu da bi obezbjedili kraj pretrage. Ovakav probni niz se naziva linearnim.

Metod umetanja ključeva u prazne redove tabele se naziva **otvoreno adresiranje**. Kasnije ćemo pokazati da otvoreno adresiranje sa linearnom probnim nizom ima ozbiljne nedostatke, pogotovo u slučaju gotovo pune tabele.

Postoje drugi vidovi otvorenog adresiranja koji se ponašaju mnogo bolje od otvorenog adresiranja sa linearnim probnim nizom. Jedan od njih, **dvostruko heširanje**, koristi nelinearni probni niz tako što se izračunavaju različite probne adrese za različite ključeve. Prikazaćemo dvostruko heširanje na primjeru iste prazne tabele kao i kod linearnog probnog niza.

Prvo ćemo definisati funkciju za računanje probnog umanjenja, p(Ln). Za jednostavnu ilustraciju, neka je

$$p(L_n) = \max(1, n/7).$$

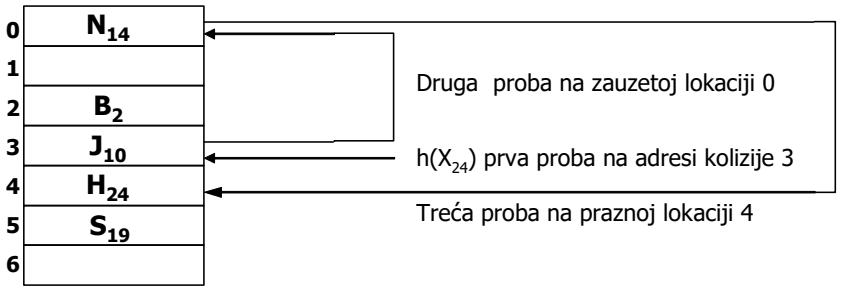
U narednoj tabeli su prikazane vrijednosti funkcija $p(L_n)$ i $h(L_n)$ za ključeve koje ćemo umetnuti u tabelu T.

KLJUČ L_N	$H(L_N)$	$P(L_N)$
J_{10}	3	1
B_2	2	1
S_{19}	5	2
N_{14}	0	2
X_{24}	3	3
W_{23}	2	3

Počećemo umetanjem ključeva J_{10}, B_2, S_{19} i N_{14} u praznu tabelu T. Opet koristimo vrijednosti heš funkcije, $h(L_n)$ za određivanje heš lokacije na koju ćemo prvo pokušati da

umetnemo ključ. Kako nema kolizije među ovim adresama, ključevi se umeću direktno u redove 3, 2, 5 i 0, respektivno.

Zatim pokušavamo da umetnemo ključ X_{24} na njegovu heš adresu 3. Kako je ova adresa zauzeta, umanjujemo je za 3 i pokušavamo da umetnemo ključ X_{24} na adresu 0. I ova adresa je zauzeta, pa je ponovo umanjimo za 3 (7-3=4) i pokušamo da umetnemo ključ X_{24} na adresu 4, koja nije zauzeta i tu se umeće X_{24} (vidi sliku),

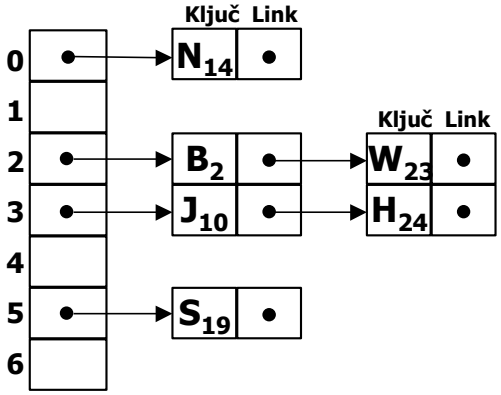


Konačno, pokušavamo da umetnemo ključ W_{23} na njegovu heš adresu 2. Pošto je adresa 2 zauzeta, umanjimo je za 3, (2-3=-1, 7-1=6, ili brojimo 1,0,6) pa umetnemo ključ W_{23} na adresu 6 (pošto ona nije bila zauzeta). Tako su ključevi J_{10}, B_2, S_{19}, N_{14}, X_{24} i W_{23} umetnuti na adrese 3, 2, 5, 0, 4 i 6, respektivno.

Kod dvostrukog heširanja kada su dva ključa sa istom heš adresom u koliziji, oni obično imaju različito umanjenje i različite probne nizove. Ključevi koji su u koliziji, praćenjem različitih probnih staza, brže ispunjavaju prazne redove tabele. Kod otvorenog adresiranja sa linearnim probnim nizom svi ključevi prate jedan probni niz (probe sequence), pa je zato otvoreno adresiranje sa dvostrukim heširanjem bolji metod za razrješavanje kolizije.

U narednom primjeru ćemo predstaviti treći način razrješenja kolizije u heš tabeli T, pravljenjem lanaca (**chaining**). Ideja je da se svi ključevi sa istom heš adresom povežu u linearnu listu koja počinje na toj heš adresi. (Svaku od tih listi možemo da zamislimo kao lanac, pa odtuda i naziv za ovu tehniku.)

Na primjer, opet ćemo ključeve, J_{10}, B_2, S_{19}, N_{14}, X_{24} i W_{23}, da umetnemo u istu, inicijalno praznu, tabelu T. Rezultat je prikazan na sljedećoj slici.



Da kompletiramo ovaj niz primjera, ukratko ćemo skicirati **bucketing** metod hešinga. Ovaj metod se koristi za baratanje sa velikim kolekcijama podataka.

Pretpostavimo da imamo veliku heš tabelu, T, sa 20000 praznih redova. Ovu veliku tabelu možemo podjeliti u 200 manjih podtabela, od kojih svaka ima 100 praznih redova i svaku takvu, malu, podtabelu zovemo **bucket**. Možemo se dogovoriti da se u svaki bucket ključevi smještaju sekvencijalno, po rastućem redosljedu. Inicijalno, treba da odredimo u koji bucket ćemo smjestiti dati ključ. Heš funkcija h(K) može uzimati cjelobrojne vrijednosti u rasponu 0:199, čime se specifira podtabela. Zatim možemo iskoristiti binarnu pretragu za lociranje ključa K u uređenom nizu redova u bucket–u određenom heš adresom h(K). Ova tehnika nema superiorne performanse kada je tabela T smještena u primarnoj memoriji, ali se jako dobro ponaša za tabele smještene na relativno sporo-rotirajuće spoljne memorijske medije, kakvi su diskovi.

Kolizije, faktori unosa i klasteri

Za dalje razvijanje ideje hešinga, treba da uvedemo i definiramo još neke pojmove. Prvo ćemo detaljnije razmotriti same hešing funkcije. Neka nam je T heš tabela sa M redova čije su adrese u rasponu 0:M-1. (Takva tabela se može posmatrati i kao jednodimenzioni niz. Idealna heš funkcija, h(K), preslikava ključeve K na adrese tabele u rasponu 0:M-1 na uniformni (za sve adrese je ista vjerovatnoća da će biti izabrane) i slučajni način. U praksi je prilično komplikovano odabrati dobru heš funkciju.

Kolizije

Do kolizije dolazi kada se dva različita ključa K i K’ preslikavaju u istu heš adresu tabele T, odnosno, **kolizija** između dva različita ključa, K i K’, nastaje kada pokušamo da umetnemo ključeve K i K’ u heš tabelu T, a oba ključa imaju istu heš adresu, h(K)=h(K’).

Način rješavanja kolizije je metoda kojom se pronalazi slobodno prazno mjesto u tabeli T u koju se smješta ključ K', ako je njegova heš adresa h(K') već zauzeta drugim ključem K prethodno smještenim u tabelu T.

Suprotno našoj intuiciji, činjenica je da su **kolizije relativno česte**, čak i u slabo popunjenim heš tabelama. Postoji paradoks koji se zove Rođendanski paradoks **R. fon Misesa** koji nam pomaže da shvatimo učestanost kolizija. Prema njemu, ako su u nekoj prostoriji 23 ili više osoba, šansa da dvije ili više osoba imaju rođendan istog dana je veća od 50%. (Druga varijanta kaže da ako je 88 ili više osoba u istoj prostoriji, onda je šansa da 3 ili više osoba imaju rođendan istog dana ,također, veća od 50%.)

Faktori unosa i grupisanje

Za metodu metode otvorenog adresiranja se treba definisati faktor unosa heš tabele T. Pretpostavimo da je heš tabela T veličine M, što znači da ima mjesta za M slogova tabele, i da je N od ovih M slogova zauzeto (to jest, M-N slogova je prazno). Tada je definicija faktora unosa slijedeća:

Faktor unosa,u oznaci, α, heš tabele dimenzije M sa N zauzetih slogova, definiše se sa α=N/M.

Na primjer, ako je heš tabela T veličine 100 sa 75 zauzetih slogova i 25 praznih slogova, onda je njen faktor unosa α=75/100=0.75. Dakle, faktor unosa je broj između 0 i 1. (U otvorenom adresiranju mi definiramo punu tabelu kao tabelu sa tačno jednim praznim slogom, stoga faktor unosa ne može biti jednak 1. Ovo je važna garancija da bi algoritmi pretraživanja i umetanja novog ključa u tabelu, efikasno završavali rad.)

Takođe, faktor unosa možemo shvatiti kao procenat zauzetih mjesta u tabeli T (ako je α=0,25 to znači da ima 25% zauzetih mjesta, ako je α=0,5 onda ima 50% zauzetih mjesta...).

Pređimo, sada, na grupisanje (clustering). **Klaster** je sekvenca susjednih zauzetih slogova u heš tabeli. Klasteri ne sadrže prazne ključeve i sastoje se od uzastopnih nizova zauzetih slogova. Iz toga proizilazi da je metod linearne probe uzrok primarnog grupisanja. Možemo videti da kada je neki broj ključeva u koliziji na datoj adresi i kada za razrješavanje te kolizije koristimo linearnu probu, ključevi u koliziji se smještaju na praznu lokaciju odmah ispod lokacije kolizije (jer linearna proba traga za lokacijom praznog mjesta u tabeli na prvoj manjoj adresi od lokacije kolizije). Ovo može izazvati stvaranje male grupe ključeva na lokaciji kolizije. Grubo govoreći, u primarnom grupisanju dešava se slijedeće: mala grupa ključeva raste sve više po broju i veličini. Kako god pokušamo da umetnemo novi ključ u sredinu grupe, linearna proba nas primorava da gledamo na donju granicu grupe kako bi pronašli prvu slobodnu lokaciju za umetanje novog ključa. Zbog toga veće grupe "privlače više pogodaka" novih ključeva za umetanje, i izuzetno brzo rastu na donjoj granici (gde se riječ donja odnosi na smjer manje adrese u tabeli, a veće na smjer veće adrese u tabeli). Čak šta više, manje grupe zajedno formiraju veće, a tako formirane veće grupe rastu još brže. Ovaj fenomen formiranja grupa, njihov rast i spajanje u veće zove se primarno grupisanje.

Suprotno tome, kada kolizije razrješavamo dvostrukim heširanjem, a ne linearnom probom, nema primarnog grupisanja. Pokazuje se da je dvostruko heširanje bolje za rad od linearne probe, zbog odsustva grupisanja.

Dva primjera primarnog grupisanja i njegove odsutnosti

Sada ćemo pokazati dva pažljivo odabrana primjera kako bi ilustrirali formiranje primarnih klastera kada koristimo linearnu probu, i njihovo odsustvo kada koristimo dvostruko heširanje. Koristimo tabelu male veličine da bismo pokazali šta se dešava i u većim tabelama.

Prethodne programske strategije izvršene su na heš tabeli T veličine M=11 sa slogovima numeriranim 0:10, i koristili smo troslovne kodove aerodroma kao ključeve.

KLJUČ K=KOD AERORDROMA	HEŠ FUNKCIJA H(K)	P(K) ZA DVOSTRUKO HEŠIRANJE	P(K) ZA LINEARNU PROBU
PHL	4	4	1
ORY	8	1	1
GCM	6	1	1
HKG	4	3	1
GLA	8	9	1
AKL	7	2	1
FRA	5	6	1
LAX	1	7	1
DCA	1	2	1

U tabeli su vrijednosti heš funkcija h(K) i dvije odvojene probne funkcije umanjenja p(K) od kojih je jedna korištena za linearnu probu, a druga za dvostruko heširanje.

Heš funkcija h(k), koja se koristi u tabeli računa se pomoću ključeva koji predstavljaju vrijednosti cijelih brojeva u bazi 26 i redukovanjem tih cijelih brojeva na veličinu tabele M=11. Kada pustimo da troslovni kod aerodroma, $X_2X_1X_0$, bude predstavljen u bazi 26 to znači da slovo "A" predstavlja cifra vrijednosti 0, slovo "B" vrijednosti 1, "C" - 2, "D" - 3, i tako redom do slova "Z" koje ima vrijednsot 25.

Sada se troslovni kod aerodroma, K= $X_2X_1X_0$, može prevesti iz broja u bazi 26 u decimalni cijeli broj, pomoću formule:

$$\text{Base26ValueOf}(K) = X_2*262 + X_1*261 + X_0*260 .$$

Na primjer, ako je K="DCA", onda je

$$\text{Base26ValueOf}(\text{"DCA"}) = 3*262 + 2*261 + 0*260 = 3*676 + 2*26 + 0*1 = 2080$$

Sada definirajmo heš funkciju h(K) kao ostatak pri djeljenju vrijednosti ključa K u bazi 26 sa 11, što se računa po formuli:

$$h(K) = \text{Base26ValueOf}(K) \% 11.$$

Na primjer, h("DCA") = 2080 % 11 = 1, jer je 2080 = 11*189 + 1.

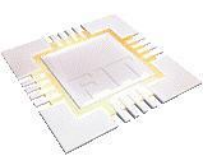
Vrijednost probne funkcije umanjenja p(K) u trećoj koloni table 8.7 računa se pomoću formule:

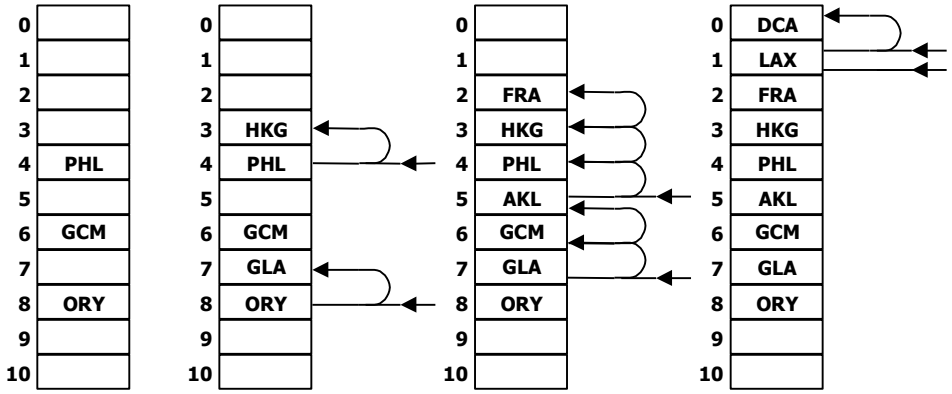
$$p(K) = \max(1, (\text{Base26ValueOf}(K)/11) \% 11).$$

Konačno, probna funkcija umanjenja p(K) za linearnu probu definira se formulom p(K)=1, što se vidi u četvrtoj koloni tabele.

Sada, kada smo odredili vrijednosti h(K) i p(K), možemo umetati neke kjučeve aerodromskih kodova u početno praznu tabelu T. Koristimo, prvo, otvoreno adresiranje sa linearnom probom (p(K)=1) za sve ključeve. Umećemo ključeve u tabelu po redosljedu datom u prvoj koloni tabele 8.7 koristeći programsku strategiju za heš umetanje.

Poslije umetanja prva tri ključa PHL, ORY i GCM, dobijamo konfiguraciju prikazanu skroz levo na narednoj slici. Posle umetanja slijedeća dva ključa HKG i GLA dobijamo konfiguraciju u drugoj koloni slike. Kada smo pokušali da umetnemo ključ HKG, vidjeli smo da je u koliziji na lokaciji 4 sa ključem PHL, koji je već u tabeli, pa smo ga onda smjestili na prvu manju lokaciju koja je prazna (lokacija 3). Slično se dešava i sa ključem GLA, koji je u koliziji sa ključem ORY na lokaciji 8, pa ga smještamo na prvu manju praznu lokaciju 7.





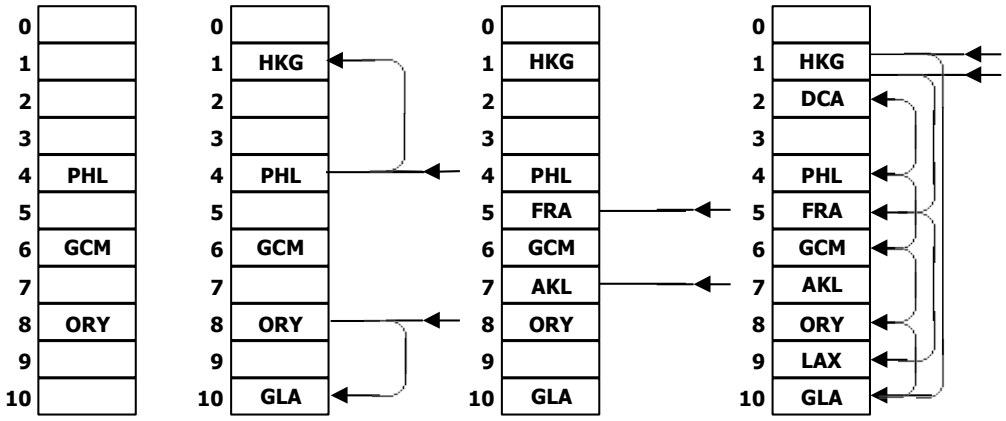
Primjetimo da su ključevi PHL i HKG, koji su u koliziji, stvorili mali primarni klaster sa dva ključa na mjestu kolizije. Takođe, primjetimo da je umetanje GLA prouzrokovalo stvaranje klastera od tri ključa. Primarni klasteri se formiraju čak i u ovako jednostavnom primjeru.

Treću kolonu slike dobijamo umetanjem ključeva AKL i FRA. Prvo, AKL je u koliziji sa ključem GLA koji je već na lokciji 7, a prvo slobodno mjesto sa manjom lokacijom je 5, pa ga tu umećemo. Na ovaj način pretodna dva primarna klastera spajamo u jedan veliki klaster. Zato, ključ FRA koji je trebalo smjestiti na lokaciju 5 smještamo na lokaciju 2. Konačno, kada umećemo posljednja dva ključa LAX i DCA, vidimo da LAX ide na praznu lokaciju 1, DCA je u koliziji sa LAX i ide na lokaciju 0.

Prema tome, kada bi sada tražili ključ AKL, koristeći programsku strategiju za heš pretraživanje, prvo bi pokušali da ga pronađemo na njegovoj heš lokaciji h(AKL)=7, a kako nije na tom mjestu, smanjujući indeks probnog niza za 1, našli bi ga na lokaciji 5.

Ako bi sada pokušali da u tabeli T pronađemo ključ MIA koji se tu ne nalazi, a ima heš adresu h(MIA)=4, prvo bi ga tražili na lokaciji 4, pa na 3, 2, 1, 0, 10,... tražeći slog tabele koji sadrži MIA ili prvi prazan slog. Kako smo naišli na prvi prazan slog pre nego što smo pronašli MIA, zaključujemo da MIA nije u tabeli i vraćamo specijalnu adresu tabele, -1 (znači da MIA nije u tabeli).

Pređimo sada na dvostruko heširanje i pogledajmo šta se dešava kada punimo tabelu T ključevima iz tabele sa ključevima istim redom kao i ranije. Koristimo, sada vrijednosti iz treće kolone u tabeli sa ključevima za p(K). Naredna slika prikazuje četiri stadija procesa umetanja.



U prvoj koloni slike umetnuli smo ključeve PHL,GCM i ORY. Nema kolizije, pa su oni umetnuti na svoje originalne heš lokacije (tako su prve kolone obe slike jednake). Sada umećemo dva slijedeća ključa HKG i GLA, koji daju konfiguraciju druge kolone slike, tako vidimo kako se razrješava kolizija pomoću dvostrukog heširanja i koliko se ona razlikuje od rješavanja kolizije pomoću linearne probe.

Prvo, kada pokušamo da umetnemo HKG na njegovu heš lokaciju 4, vidimo da je ona već zauzeta sa PHL. Razrješavamo koliziju tako što pročitamo vrijednost p(HKG)=3 u tabeli sa ključevima, oduzmemo 3 od 4 da pronađemo novu praznu probnu lokaciju za HKG (lokacija 1) , kako je lokacija 1 prazna umećemo HKG na tu lokaciju. Postupak nastavljamo dalje za GLA, kolizija sa ORY na lokaciji 8, umećemo ga na lokaciju 10 (p(GLA)=9, 8-9=-1, 11-1=10).

Upoređujući druge kolone vidimo da sada izbegavamo formiranje primarnih klastera. U slijedećem koraku, (treća kolona slike) umećemo AKL i FRA i imamo sreće, njihove heš lokacije su prazne, dakle nema kolizija (označeno sa pravim strelicama na slici). Konačno, ubacujemo posljednja dva ključa, LAX i DCA. Kada pokušamo da umetnemo LAX na njegovu heš lokaciju 1, vidimo koliziju sa HKG. Sada računamo p(LAX)=7, brojimo 7 lokacija nadole od lokacije 1: 1, 0, -1=10, 9, 8, 7,6 , dolazimo do lokacije 5 na kojoj se nalazi FRA, pa ponavljamo postupak i dolazimo do lokacije 9 koja je prazna i tu umećemo LAX.

Konačno, umećemo ključ DCA, h(DCA)=1, pa je DCA u koliziji sa HKG na lokaciji 1, koristimo p(DCA)=2 za računanje nove lokacije. Redosljed je 10, 8, 6, 4 i 2. Lokacija 2 je prazna i tu smještamo DCA.

Primjetimo da, iako su ključevi LAX i DCA na početku u koliziji na lokaciji 1, prolaze različite probne nizove nakon kolizije (za LAX 1, 5, 9, za DCA 1, 10, 8, 6, 4, 2).Tako dvostruko heširanje ključeve koji su u koliziji na početku vodi različitim putevima (probnim nizovima) do pronalaženja njihovih lokacija. Zbog toga nema primarnog grupisanja u dvostrukom heširanju.

Šta se dešava sa ključem MIA koji nije u četvrtoj koloni slike 8.9. h(MIA)=4, p(MIA)=8. Programska strategija za heš pretraživanje nam kaže da ključ prvo tražimo na lokaciji 4, gde je kolizija sa ključem PHL, zatim tražimo lokaciju oduzimajući 8. Put, dakle, ide ovim redosljedom: 4, 7, 10, 2, 5, 8, 0. Lokacija 0 je prazna, zaključujemo da MIA nije u tabeli T, i vraćamo specijalnu adresu, -1, da označimo da MIA nije u tabeli T.

Kako se osiguravamo da probni niz pokriva tabelu?

Kako bi ispunili da heš umetanje otvorenim adresiranjem i algoritam heš pretrage rade kako treba, moramo garantovati da svaki korišćeni probni niz prođe sve lokacije u heš tabeli. U suprotnom, algoritam heš umetanja neće garantovano pronaći prazan slog u tabeli u koji umećemo novi ključ, a algoritam heš pretrage neće garantovano pronaći prazan slog da signalizira sopstveno zaustavljanje, kada traži ključ koji nije u tabeli.

Prilično je očigledno da metoda linearne probe generiše probni niz koja pokriva sve moguće lokacije u tabeli, jer, polazeći od početne kolizije kreće se nadole, sistematično računajući linearni niz susednih adresa tabele, dok "ne padne" sa dna tabele i onda se vraća od vrha ponovo na dole - vraćajući se, eventualno, do mjesta početne kolizije u jednom neprekidnom ciklusu (zaokretanje - wrap-around).

Nije toliko očigledno da probno umanjenje p(K), koja se koristi u dvostrukom heširanju uvjek pokriva sve lokacije tabele u nekom redosljedu u probnim nizovima koje generiše, a za više informacija možete konsultovati literaturu.

Formule uspješnosti

Posvetimo se sada karakterizaciji uspješnosti otvorenog adresiranja koristeći i metodu linearne probe i metodu dvostrukog heširanja.

Jasno je da ukoliko se faktor unosa α heš tabele T povećava, da se uspješnost umetanja novog ključa K smanjuje, zato što kako se tabela sve više puni, šansa da ćemo pogoditi praznu lokaciju u prvom heširanju se smanjuje, a u slučaju kolizije, trebaće nam više vrijemena da izračunamo lokacije tabele u probnom nizu dok ne pronađemo praznu lokaciju kako bi umetnuli ključ K.

Pretpostavimo da heš tabela T ima tačno N zauzetih slogova, tj. da je njen faktor unosa α=N/M. Definirajmo dvije veličine, C_N i C'_N, gde je C_N prosječan broj probnih adresa provjerenih tokom uspješne pretage, a C'_N tokom neuspješne pretrage (ili, što je identično, tokom umetanja novog ključa K).

Za otvoreno adresiranje sa linearnom probom, imamo dvije formule uspješnosti:

$$\left.\begin{aligned} C_N &\approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right) && \text{za uspješnu pretragu} \\ C'_N &\approx \frac{1}{2}\left(1+\left(\frac{1}{1-\alpha}\right)^2\right) && \text{za neuspješnu pretragu} \end{aligned}\right\}$$

Ove formule se primjenjuju kada je tabela T popunjena do 70%, tj. kada je $\alpha \leq 0.7$.

Za otvoreno adresiranje dvostrukim heširanjem odgovarajuće formule su:

$$\left.\begin{aligned} C_N &\approx \frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right) && \text{za uspješnu pretragu} \\ C'_N &\approx \left(\frac{1}{1-\alpha}\right) && \text{za neuspješnu pretragu} \end{aligned}\right\}$$

Konačno, za razrješenje kolizije pravljenjem odvojenih lanaca (u kojima čuvate sve ključeve koji su u koliziji na datoj heš adresi u povezanoj listi počevši od te lokacije, kao na slici 8.5), odgovarajuće formule su:

$$\left.\begin{array}{ll} C_N \approx 1+\frac{1}{2}\alpha & \text{za uspješnu pretragu} \\ C_N^{'} \approx \alpha & \text{za neuspješnu pretragu} \end{array}\right\}$$

Izbor heš funkcije

Vrlo je važno da heš funkciju h(K) odaberemo pravilno, kako bi metode heširanja objašnjene u prethodnim odjeljcima radile dobro. Idealno bi bilo da h(K) označava ključeve uniformno i slučajno na cijelom opsegu lokacija heš tabele, a da svaka lokacija bude pogođena sa h(K) za slučajno odabran ključ K. Loše odabrana heš funkcija može preslikavati ključeve neuniformno na lokacije tabele, ili može preslikavati susjedne klastere ključeva na klastere lokacija heš tabele.

Metod dijeljenja

Jedan metod za odabiranje heš funkcije h(K), koji daje dobar rad aplikacije dvostrukog heširanja, je odabrati prost broj za veličinu tabele M i interpretirati ključeve K kao cele brojeve, podijeliti K sa M, dobiti količnik Q i ostatak R. Ostatak R koristimo kao vrijednost za h(K), a količnik Q kao vrijednost probnog umanjenja za dvostruko heširanje (sem kad je Q=0, onda stavljamo p(K)=1), ili simbolima:

$$\left.\begin{array}{lll} h(K) = K \,\%\, M, & \text{i} \\ p(K) = \max(1, K/M) & \end{array}\right\}$$

Praktično, korisno je smanjiti p(K) na broj u rasponu 1:M-1, zamjenjujući p(K)>M sa max(1,p(K) % M), ako je potrebno. Ipak, treba biti oprezan prilikom biranja prostog broja za M i njegovog korišćenja u konjunkciji sa metodom dijeljenja. Praktično, ako je r baza skupa karaktera ključeva K, i k i a dovoljno mali cijeli brojevi, tada ne treba da biramo prost broj M oblika rk±a.

Druge metode za biranje heš funkcije

Tri druge metode za odabir heš funkcije su:

- "preklapanje",
- "kvadriranje sredine" i
- "odsijecanje".

U "preklapanju", ključevi se dijele na sekcije, a sekcije se sabiraju. Na primjer, ako imamo devetocifrene ključeve, K=013402122, K možemo podijeliti na tri sekcije: 013,402 i 122, kada ih saberemo dobićemo 537 za vrijednost h(K). (Takođe, možemo koristiti množenje, oduzimanje,itd... na neki način da kombinujemo sekcije da bi dobili krajnju vrijednost.)

U "kvadriranju sredine", ako ponovo imamo devetocifreni K=013402122, možemo uzeti srednje cifre 402 i kvadrirati ih, tako imamo h(K)=161604, ako je h(K)>M, onda možemo, recimo, uzeti četiri srednje cifre kao rezultat kvadriranja i staviti h(K)=6160.

U "odsijecanju", jednostavno obrišemo deo ključa i koristimo preostale cifre (ili bitove ili karaktere). Na primjer, ako je K=013402122, možemo ignorirati sve sem poslednje tri cifre i staviti h(K)=122. Dok "odsijecanje" nije teško izračunati, ono nema osobinu da raspoređuje ključeve uniformno i slučajno na prostor lokacija heš tabele. Iz tog razloga, ono se najčešće koristi u konjunkciji sa nekom drugom pomjenutom metodom odabira heš funkcije, skoro nikad se ne koristi samo.

Poređenje metoda pretrage upotrebom ATP-tabela

Podsjetimo se šta su ATP-tabele. Sjetite se da je apstraktna tabela T definisana kao kolekcija parova oblika (K,I), gde je K identificirani ključ, a I neka informacija povezana sa K, takva da ne postoje dva para u kolekciji koji imaju identične ključeve i takva da je na njoj definisano sedam operacija: inicijalizacija T, određivanje da li je T puna, numeriranje slogova u T, i umetanje, brisanje, pretraživanje i preuzimanje informacija, i ažuriranje slogova u tabeli T.

REPREZENTACIJE	INICIRAJ	PUNA	PRETRAŽI PREUZMIINFO AŽURIRAJ	UMETNI	BRIŠI	NUMERIŠI
SORTIRANI NIZ **STRUKTURA**	O(n)	O(1)	O(log n)	O(n)	O(n)	O(n)
AVL-STABLO **STRUKTURA**	O(1)	O(1)	O(log n)	O(log n)	O(log n)	O(n)
HEŠ TABELA	O(n)	O(1)	O(1)	O(1)	O(1)	O(nlog n)