Алгоритми и структури на податоци

– белешки од предавања –

7. Хеширање (Hashing)

Hash табелите се користат за едно од најефикасните типови на пребарување - хешингот (hashing). Во основа, хеш табела се состои од низа во која до податоците може да се пристапи преку посебен индекс (клуч). Хеш табелите реализираат само три основни операции: внесување, бришење и пребарување. Хеш табелата врши мапирање помеѓу множеството на сите можни клучеви и сите можни позиции во низата со помош на така наречена хеш функција. Хеш функцијата прима вредност на клуч и враќа позиција во низата. Клучевите може да бидат од различни домени но резултатите се секогаш целобројни. Бидејќи пресметката на хеш фунцијата е обично со константно време на извршување, со помош на хеш табелата може да се изврши наоѓање на клучот (или местото на податокот во низата) за константно време. Од тука произлегува дека хешингот може да се користи кога се извршува пребарување во константно време, односно се користи кога сакаме да добиеме на ефикасноста на пребарувањето. Дополнително во теорија, хеш функциите овозможуваат внесување и бришење на податоците во константно време.

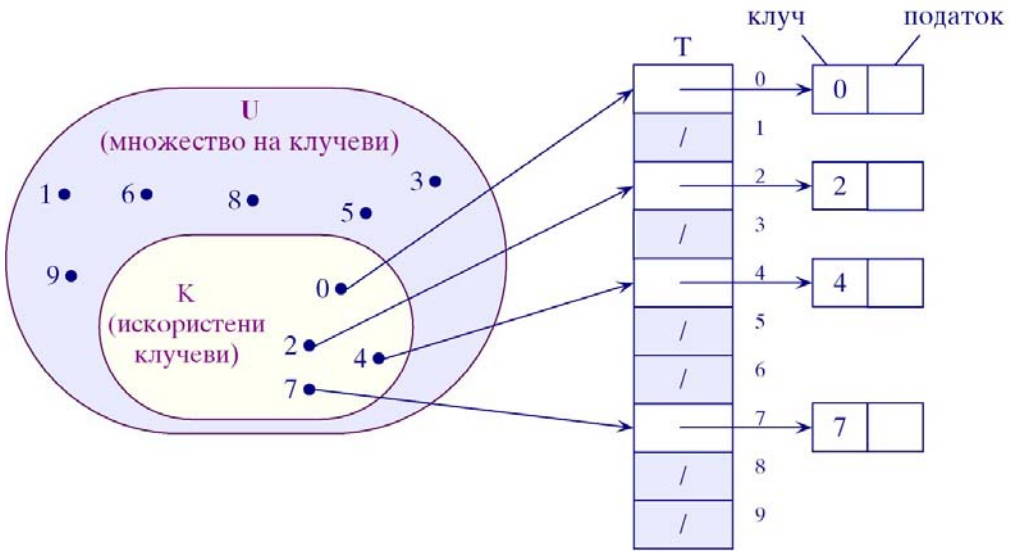
Кога хеш функцијата е така избрана што гарантира дека не постојат два клуча кои ќе вратат иста хеш вредност, се вели дека хеш табелата е со директно адресирање (што имплицира константна вредност на времетраењето на пребарувањето). Ова е идеален случај, но ваквиот тип на адресирање е многу редок во праксата поради комплексноста (често и неможноста за нејзина реализација). На пример, доколку сакаме да адресираме телефонски именик со директно адресирање, а за влез во хеш функцијата да го користиме името (низа од букви од 15 карактера), хеш табелата би имала 32^{15} = 37778931862957161709568 можни влеза од кој најголемиот дел не би се користел затоа што не би претставувале реално име.

Вообичаено е бројот на можни влезови во хеш табелата да биде мал во споредба со бројот на можни клучеви. Следствено хеш функцијата вообичаено мапира повеќе клучеви на иста позиција во табелата. Кога два клуча се мапираат до иста позиција тие колизираат. Добрите хеш функции ги минимизираат колизиите, но сепак мора да постои стратегија за справување со нив. Постојат повеќе типови на хеш табели (поврзани, со отворено адресирање), повеќе реализации на хеш функции, како и повеќе стратегии за разрешување на колизиите.

Хеш табелите се користат во: системите за управување со бази на податоци (и тоа за оптимизација на случајниот (random) пристап до податоците), табелите на симболи кои се користат за ефикасен пристап до симболите кои се користат во програмите при процесот на компајлирање, имениците на податоци за подршка на ефикасно пребарување низ податочните структури и сл.

7.1 Табели со директно адресирање

Директното адресирање е едноставна техника која работи добро доколку множеството на клучеви U е релативно мало па може да се претпостави дека постои 1:1 релација помеѓу множеството клучеви U и множеството податоци - низата T кои треба да се сместат (види слика 7.1). На секој можен клуч од множеството клучеви U = {0,1, . . . , 9}, соодветствува еден индекс во табелата. Искористените клучеви K = {2, 3, 5, 8} ги одредуваат полињата од табелата до кои има придружено елементи. Останатите полиња од табелата (кои не содржат покажувачи кон елементи туку нулеви покажувачи) се затемнети.



слика 7.1 Табела со директно адресирање

Операциите кои работат со хеш табелата од ваков тип се тривијални за имплементација (и имаат *O*(1) време на извршување):

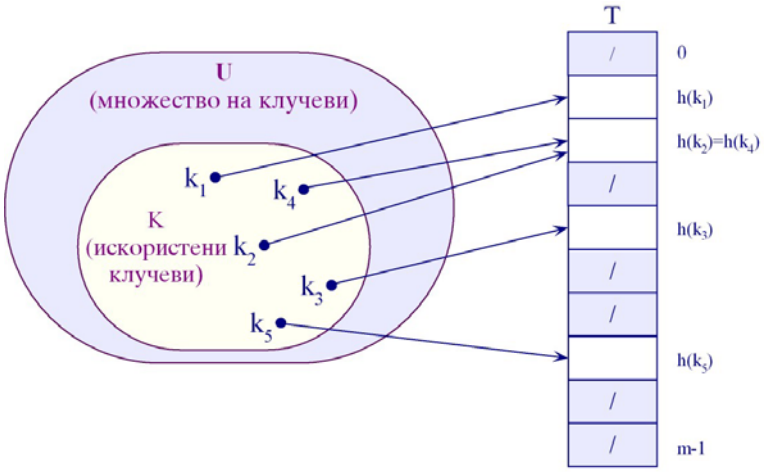
```
DIRECT-ADDRESS-SEARCH(T,k)
return T[k]
DIRECT-ADDRESS-INSERT(T,x)
T[key[x]] ← x
DIRECT-ADDRESS-DELETE(T,x)
T[key[x]] ← NIL
```

Хеш табелата во овој случај е имплементирана како низа која содржи покажувачи кон структура која го содржи клучот и елементот. Оваа табела може да се реализира и како обична низа (која во полето со индекс кој соодветствува на клучот ќе ја содржи податочната вредност), но тогаш треба дополнително да се ализира означување на поле од табелата кое не е искористено.

Важно е да се напомене дека операциите наоѓање минимум или максимум од податочните вредности не се подржани со користење на хеш табелите (па ни на најефикасните хеш табели). Во најлош случај, наоѓање на минималниот елемент има O(n) време на извршување.

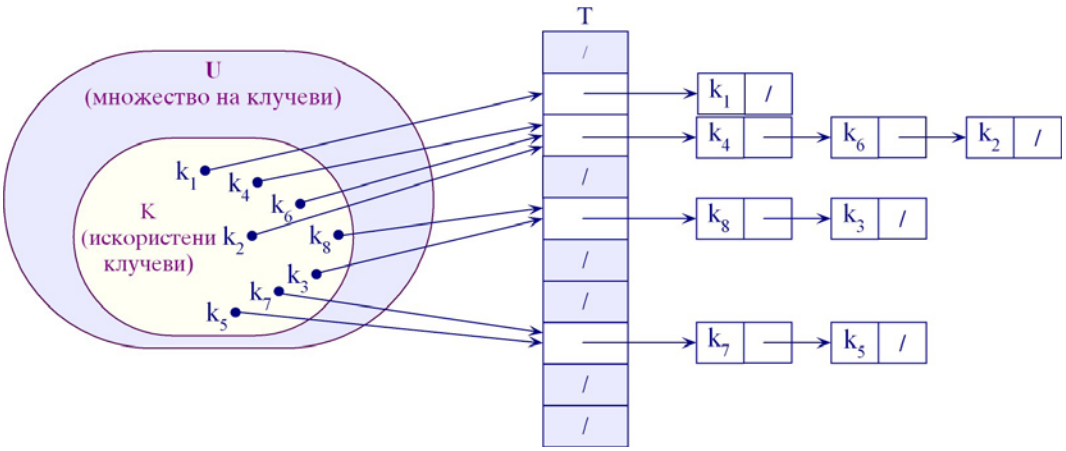
7.2 Хеш табела

Во реалноста бројот на клучеви (а со тоа и бројот на вредности) е поголем од големината на хеш табелата. Тогаш е очигледно дека ќе постојат клучеви кои со примена на хеш функцијата ќе одредат исто поле во табелата. Доколку во праксата не се користат сите клучеви, овој пристап е дозволен затоа што има подобро искористување на меморијата. Ваквото користење на процесот на хеширање, односно ваквиот тип на табела е познат како хеш табела. Една ваква табела е прикажана на слика 7.2. Процесот кога хеш функцијата h(x) за два клуча дава иста позиција во табелата се нарекува колизија.



слика 7.2 Хеш табела со колизија

Наједноставен начин за надминување на колизиите е користење на поврзани листи како простор каде што се вметнуваат елементите кај кои се случува колизија. Ваквиот пристап за надминување на колизиите се нарекува chaining[1]. Приказ на вакво решение е даден на слика 7.3. Интересно е да се забележи дека на овој начин всушност се формира m патно пребарувачко стебло со висина 2 и степен еднаков на бројот на елементи во хеш табелата (низата).



слика 7.3 Хеш табела која ги надминува колизиите со користење на букети

[1] Преводот што се сретнува во македонската литература е „користење на букети“

Операциите кои се од интерес се сведуваат на операции со поврзани листи, при што вметнувањето (доколку е на почетокот или крајот на листата) може да се имплементира да биде константно. Пребарувањето и бришењето на елемент е линеарно зависно од просечниот број на елементи во листата. Од тука следи дека добра хеш функција би била онаа функција која „рамномерно“ би ги распределувала клучевите во секоја од поврзаните листи (букетите).

```
CHAINED-HASH-INSERT(T,x)
vmetni go x na pocetokot na listata T[h(key[x])]
CHAINED-HASH-SEARCH(T,k)
najdi elemen so kluc k vo listata T[h(k)]
CHAINED-HASH-DELETE(T,x)
izbrisi go x od listata T[h(key[x])]
```

7.3 Хеш функции

Добра хеш функција е онаа функција која со еднаква веројатност ги дистрибуира множеството клучеви во одреден интервал на целобројни вредности. Ваквата дистрибуција често се нарекува униформна дистрибуција. Бидејќи такви дистрибуции тешко се постигнуваат, во праксата често се користат хеуристички (случајни но „ветувачки“) методи за дефинирање на хеш функциите.

Лоши хеш функции се функции кои ги групираат клучевите во еден кластер (букет). Дали функцијата е добра или лоша зависи од вредностите на клучевите кои се сместуваат во хеш табелата. За разлика на тешкотијата за дефинирање на добра хеш функција, лошата хеш функција може лесно да се примети. Пример за лоша хеш функција е функција која се дефинира како остаток при делење на клучот што се сместува со бројот 10, во случај кога сите клучеви завршуваат на нула. Зошто?

Во продолжение на текстот ќе се смета дека клучевите имаат вредности на природни броеви. Иако тоа не е општ случај, може да се претпостави дека постои метода која произволен клуч еднозначно го трансформира во природен број. Во случај на цели броеви тоа би било собирање со најмалиот цел број од даденото множество клучеви. Во случај на букви, може да се разгледуваат ASCI вредностите на буквите (симболите). Во случај на зборови, може да се разгледува збир од тежински фактори на ASCI кодовите.

Постојат многу типови на функции со кои клучот се трансформира во адреса (индекс) на хеш табелата. Искуството покажува дека сосема добри резултати се добиваат со едноставни методи на трансформација. Операциите кои инсистираат на голем број на математички операции траат подолго и ја намалуваат вкупната ефикасност на пребарувањето. Тие се применуваат само кога друг пристап не дава задоволителни резултати.

Хеш функции базирани на операцијата делење

Многу често хеш функцијата се дефинира како остаток што се добива при делење на клучот со димензијата на хеш табелата (бројот на полиња во хеш табелата). Така за клучот k, може да ја дефинираме следната хеш функција за табела која има m полиња:

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

На пример, за табела со m = 12 полиња и клуч k = 100, добиваме $h(k)$ = 4, односно дека овој клуч треба да се смести во четвртиот букет. Може да се покаже дека функциите се повеќе униформни кога m е прост (или барем непарен) број.

Хеш функции базирани на операцијата исклучиво или

Овој пристап на креирање на хеш функциите дава добри резултати во случај на долги клучеви. Се сведува на делење на битовите на клучот на парчиња долги колку големината на табелата и применување на операцијата исклучиво или над нив.

Хеш функции базирани на операцијата множење

Хеш функциите базирани на операцијата множење се изведуваат во два чекора. Во првиот чекор, клучот се множи со некоја константа A помеѓу нула и еден, при што понатаму се користи само делот од резултатот кој е после децималната запирка. Таа вредност се множи со бројот на полиња во хеш табелата и како резултат се зема долниот цел дел од кличникот. Поформално

$h(k) = \lfloor m\ (k\ A \bmod 1) \rfloor$,

каде што "$k\ A \bmod 1$" го означува делот од kA после децималната запирка односно, $kA - \lfloor kA \rfloor$.

Иако, во принцип константата A може да има било која вредност помеѓу нула и еден, некој константи даваат подобри распределби од други зависно од природата и опсегот на клучевите. Често се зема дека добра вредност за константа A е

$$A \approx (\sqrt{5} - 1)/2 = 0.6180339887$$

На пример, за горе наведената вредност на A и k = 123456, m = 10000, се добива

$h(k) = \lfloor 10000 * (123456 * 0.61803 \ldots \bmod 1) \rfloor$
$= \lfloor 10000 * (76300.0041151 \ldots \bmod 1) \rfloor$
$= \lfloor 10000 * 0.0041151 \ldots \rfloor$
$= \lfloor 41.151 \ldots \rfloor$
$= 41$.

Универзално хеширање

Без разлика на изборот на хеш функцијата, не постои гаранција дека за дадено множество клучеви нивната распределба по букети ќе биде рамномерна. Она што треба да се избегнува е најлошиот случај, во кој сите клучеви се сместуваат во еден букет. Една стратегија за избегнување на овој најлош случај е при секое хеширање да се избира различна хеш функција од конечно множество на предходно изберени функции и тоа по случаен избор. Статистички може да се докаже дека ваквиот избор дава најдобри можни резултати. Ваквиот начин на хеширање се нарекува универзално хеширање. Тоа дава добри резултати и во случај кога било која од хеш функциите поединечно потфрла.

7.4 Табели со отворено адресирање

Доколку дозволиме сместување на точно еден елемент во полето од хеш табелата (односно полето од хеш табелата или да содржи елемент или да биде нулево), работиме со табели со отворено адресирање. Ваквите хеш табели се во принцип со ограничен простор, разрешувањето на колизиите е покомплицирано, но не мора да користат покажувачи. Идејата на пребарувањето во табелите со отворено адресирање е пребарувањето да не се врши низ некој букет (поврзана листа), што значи да се проверува вредност на една мемориска локација, да се дознае вредноста на следната мемориска локација, да се отиде на неа, да се провери вредноста на таа мемориска локација итн, туку да се креира механизам кој ќе ги генерира вредностите на мемориските локации (индексите на полето) каде што се сместени елементите што се пребаруваат.

За да се изврши вметнување кај табелите со отворено адресирање се врши „пробување“ (probing), односно хеш функцијата се извршува онолку пати колку што е потребно да се најде празно место во табелата. Бидејќи согласно дефиницијата на хеш функцијата, таа за ист клуч секогаш дава ист резултат, хеш функциите кај табелите со отворено адресирање имаат два аргумента. Првиот аргумент е клучот (исто како кај хеш функциите кај табелите со директен пристап). Вториот аргумент е индикатор на бројот на пробите, односно вредноста на повикот на хеш функцијата за даден клуч. Псевдокодот даден во продолжение ја илустрира оваа постапка. При тоа функцијата h(k,i) е било која хеш функција, на пример:

h(k,i) = (k+i) mod m

```
HASH-INSERT(T,k)
1  i ← 0
2   repeat j ← h(k,i)
3          if T[j] = NIL
4             then T[j] ← k
5                  return j
6             else i ← i + 1
7  until i = m
8 error "hash table overflow"
```

Алгоритмот за пребарување на клучот к, ги пробува истите позиции во табелата и тоа по истиот редослед и проверува дали во нив се наоѓа клучот кој се бара. Поради алгоритмот кој се користи за сместување на клучот, очигледно е дека доколку клучот постои во табелата, тој ќе биде најден после оној број на обиди (проби) кои биле потребни за негово сместување. Доколку при пребарувањето се налета на нулева вредност (или пак се испитаат сите полиња од табелата), а клучот не се најде, може да се заклучи дека тој не постои во табелата. Со следниот псевдокод е претставен алгоритмот на пребарување кај табелите со отворено адресирање.

```
HASH-SEARCH(T, k)
1  i ← 0
2   repeat j ← h(k, i)
3          if T[j]= j
4             then return j
5           i ← i + 1
6    until T[j] = NIL or i = m
7  return NIL
```

Операцијата на бришење на клуч од табела со отворено адресирање е комплицирана операција, затоа што доколку вредноста што се брише едноставно се замени со нулева вредност, тоа може да предизвика погрешно работење на алгоритмот за пребарување. Наједноставно решение за овој пробле е воведување на ознака (tag) со кој ќе се означува дека некој елемент е избришан. Ова бара воведување на дополнителни полиња во табелата и промена на алгоритмот за внесување на елемент (може да се внесе елемент на поле кое не е нулево ако е означено како избришано) и алгоритмот за пребарување (доколку елементот е означен како избришан тој не треба да се споредува со клучот кој се пребарува). Овие измени на алгоритмот може да се реализираат како домашна вежба.

Во праксата обично се користат три техники за реализација на хеш функциите кај табелите со отворено адресирање: линеарно пробување, квадратно пробување и двојно хеширање.

Линеарно пробување

За дадена хеш функција *h*', методата на линеарно пробување (***linear probing)*** ја користи следната трансформација (ја дефинира следната хеш функција):

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

Овој пристап не дава униформни резултати за голем број на стандардни хеш функции и клучеви.

Квадратно пробување

Кавадратното пробување ***(Quadratic probing)*** ја користи следната хеш функција:

$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m,$

каде h' е обична хеш функција, c_1 и $c_2 \neq 0$ се произволни константи и i прима вредности од интервалот $i = 0, 1, \ldots, m - 1$.

Двојно хеширање

Двојното хеширање (***Double hashing***) користи хеш функција од следниот изглед:

$h(k, i) = (h_1(k) + i h_2(k)) \bmod m,$

каде h_1 и h_2 се предходно одредени обични хеш функции.