Hash табели

Структури за брзо пребарување

algoritam.blog.com.mk
2007

Hash структура реализирана со низа од листи

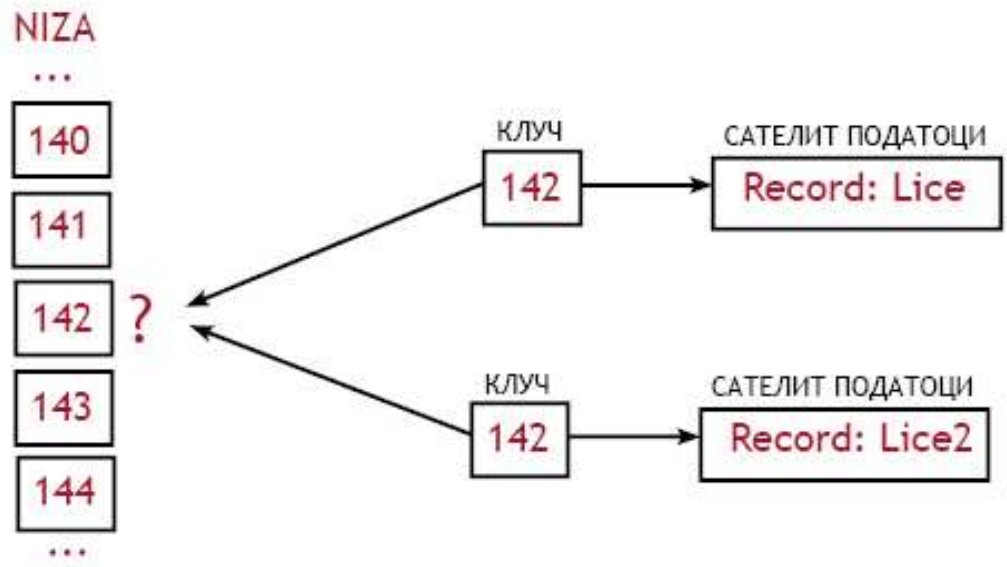
Структурите за брзо пребарување се многу важен дел од програмите во кои постојано се пребарува по некакви податоци. Најчесто, пребарувањето се врши според некаков ***клуч*** и истото треба да се изврши доста брзо. Клучот може да претставува некаков стринг или пак некој број, најчесто целоброен.

Во таквите структури за брзо пребарување се зачувуваат ***елементи*** кои воедно го претставуваат и нивниот клуч(елементите самите за себе претставуваат клучеви) , или пак можно е покрај клуч, овие елементи да содржат и сателит-податоци.

Слика 1: Елемент со клуч и сателит-податоци

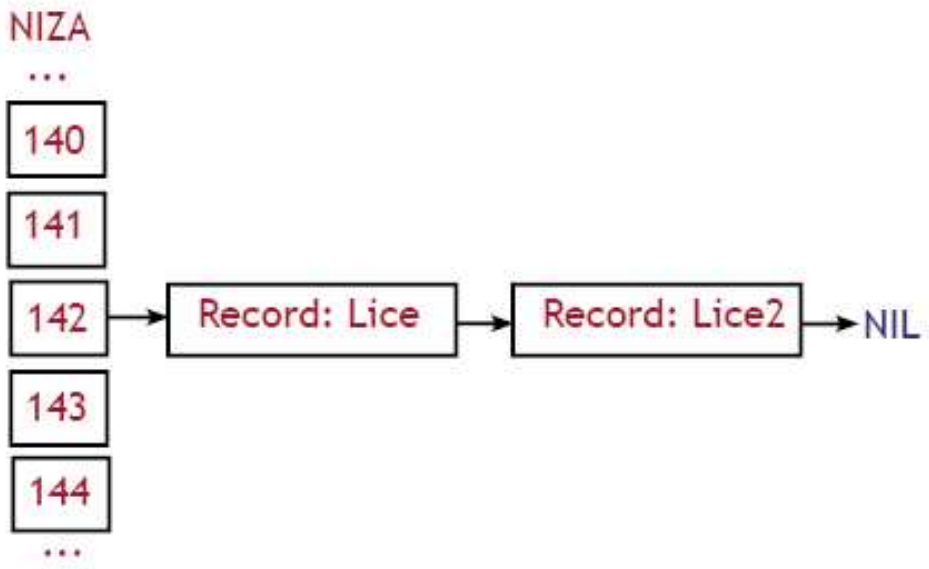
Отсега па натака во текстот ќе претпоставуваме дека клучот претставува позитивен цел број. Нека множеството клучеви што можат да се појават го обележиме со **U**.

Една од најпростите структури за брзо пребарување е ***низата***, за чии ранг на индекси **{A,..,B}** ќе земеме да важи дека $A \le \min(U), \max(U) \le B$, односно низата ќе има множество индекси кое ќе го опфати(покрие) множеството клучеви што можат да се појават(**U**). Ова значи дека за секој клуч постои едно единствено поле во низата, чиј индекс е еднаков на вредноста на клучот. Но се појавува следниов проблем: Што ако два елементи со ист клуч треба да се зачуваат во оваа низа ?



Слика 2: Проблем при појава на елементи со исти клучеви

За да се реши овој проблем се создава структурата ***низа од листи***, така што за секој клуч се одвојува по една (динамичка)листа. Па така доколку се појават елементи со исти клучеви, тие ќе бидат сместени во листата за соодветниот клуч.



Слика 3: Низа од листи

Во овој случај, множеството индекси на низата не мора да го покрие целото множество клучеви **U**. Така бројот на полиња во низата можеме да го направиме помал отколку бројот на различни клучеви, што можат воопшто да се појават. Нека тој број го обележиме со ***m***, а низата нека има индекси ***0,..,m-1***.

Така, ако се појави елемент со клуч чија вредност е ***k***, тогаш овој елемент ќе го сместиме во листата со индекс ***k mod m***. Оваа операција како резултат дава една од вредностите ***0,..,m-1***, што како вредности ни одговараат потполно, бидејќи со добиениот остаток ја идентификуваме низата во која ќе го запишеме елементот. Како вредност за бројот ***m*** се препорачува да се одбере прост број (*види* [1] *стр. 231*).

Ваквата низа од листи ја нарекуваме ***Hash табела***, а функцијата ***k mod m*** ја нарекуваме ***Hash функција.***

Програмата што следува се однесува на внесување и пребарување на стрингови во hash табела. Но пред да можеме да внесеме било каков стринг, ќе мора стрингот со некаква ***функција*** да го претвориме во цел позитивен број. Не попусто е замастен зборот функција, бидејќи секоја функција одредува една единствена слика(во случајов тоа е цел позитивен број) за секој елемент од доменот (во случајов тоа е стрингот).

Така на пример, нека оваа функција се вика ***StrToInt(s:string)*** и истата нека враќа цел позитивен број за влезен стринг ***s*** како аргумент. Имаме елемент- стринг ***'ABCD'*** кој треба да го сметиме во hash табелата. Пред да го внесеме во hash табелата ќе мора истиот овој стринг да го претвориме во цел позитивен број со помош на функцијата ***StrToInt()***. Нека ***d=StrToInt('ABCD')***. Ова значи дека вредноста ***d*** ќе ја игра улогата на ***k*** во hash функцијата изнесена погоре, и стрингот ***'ABCD'*** ќе го запишеме во листата со индекс ***d mod m***.

По некое време сакаме да провериме дали во hash табелата сме внесле стринг **'ABCD'**, затоа најпрво стрингот го претвораме во број, со помош на функцијата ***StrToInt()***. Нека ***d= StrToInt('ABCD')***. Затоа стрингот ***'ABCD'*** го бараме во листата со индекс ***d mod m***.

Пред да ја прикажеме програмата ќе дадеме две функции за претворање на стринг во цел позитивен број. Од предложените две функции ќе ја коритиме втората бидејќи дава подобри перформанси.

```
function StrInInt(const kluc:string):longint;
var
   i:integer;
   hashVrednost:longint;
begin
     hashVrednost:=0;
     for i:=1 to length(kluc) do
         hashVrednost:=((hashVrednost*17)+ord(kluc[i])) mod DOLZ_NIZA;
     {17 e mal prost broj, koj se koristi kako (brojna)osnova
     za vrednuvanje na bukvite zavisno nivnata pozicija}
     StrInInt:=hashVrednost;
end;
```

```
function StrInIntPJW(const kluc:string):longint;{PJW = P.J.Weinberger}
var
   G,hashVrednost:longint;
   i:integer;
begin
     hashVrednost:=0;
     for i:=1 to length(kluc) do
         begin
              hashVrednost:=(hashVrednost shl 4) + ord(kluc[i]);{najprvo bitovite so
koi e zapisana vrednosta vo prom. hashVrednost gi pomestuvame za 4 mesta vo levo a
potoa go dodavame redniot broj na i-tiot znak od stringot}
              G:=hashVrednost and $F0000000;{G gi zema najvaznite cetiri bita od
hashVrednost a drugite bitovi(28) gi zamenuva so 0; longint zafaka 4B = 32 b.}
              if (G<>0)
                 then
                     hashVrednost:=hashVrednost xor (G shr 24) xor G;
         end;
     StrInIntPJW:=hashVrednost;
end;
```

Забелешка: Функцијата **`StrInIntPJW`** дава подобри перформанси одколку **`StrInInt`** бидејќи првата функција за секој знак од стрингот користи операција за пресметување остаток што во суштина опфаќа операција делење.

Hash структура реализирана со низа од листи:

```
program HASH1;
const
    DOLZ_NIZA=19;{m=19}
type
    pok=^tekst;
    tekst=record
                s:string;
                sleden:pok;
    end;
```

```
var
   niza:array[0..DOLZ_NIZA-1]of pok;

procedure pecati(var kade:pok);
var
   pom:pok;
begin
     pom:=kade;
     while pom<>nil do
           begin
                write(pom^.s:12,' -> ');
                pom:=pom^.sleden;
           end;
     writeln;
end;

procedure pecatiSve;
var
   i:integer;
begin
     for i:=0 to DOLZ_NIZA-1 do
         begin
              write(i:4,': ');
              pecati(niza[i]);
         end;
end;


function StrToIntPJW(const kluc:string):longint;{P.J.Weinberger}
var
   G,hashVrednost:longint;
   i:integer;
begin
     hashVrednost:=0;
     for i:=1 to length(kluc) do
         begin
              hashVrednost:=(hashVrednost shl 4) + ord(kluc[i]);
              G:=hashVrednost and $F0000000;
              if (G<>0)
                 then
                     hashVrednost:=hashVrednost xor (G shr 24) xor G;
         end;
     StrToIntPJW:=hashVrednost;
end;

function hashInsert(const s:string):integer;
var
   pom:pok;
   indeks:integer;
begin
     indeks:=StrToIntPJW(s) mod DOLZ_NIZA;{go presmetuvame indeksot na listata vo
     koja ke go zacuvame stringot}
     new(pom);
     pom^.s:=s;
     pom^.sleden:=niza[indeks];
     niza[indeks]:=pom;
     hashInsert:=indeks;
end;

function hashSearch(const s:string):boolean;
var
```

```
   kojaLista:pok;
   indeks:integer;
   dali:boolean;
begin
     indeks:=StrToIntPJW(s) mod DOLZ_NIZA;{go presmetuvame indeksot na listata
     vo koj bi MOZEL da bide baraniot string}
     kojaLista:=niza[indeks];
     dali:=false;
     while kojaLista<>NIL do
           begin
                if kojaLista^.s=s
                   then
                       begin
                            dali:=true;
                            break;
                       end;
                kojaLista:=kojaLista^.sleden;
           end;
     hashSearch:=dali;
end;

procedure initHash;
var
   i:integer;
begin
     {na pocetok site pokazuvaci za sekoja lista e NIL}
     for i:=0 to DOLZ_NIZA-1 do
         niza[i]:=NIL;
end;

begin
     initHash;
     hashInsert('niko lazov');
     hashInsert('AAAA');
     hashInsert('bzzz');
     hashInsert('bfsdf');
     hashInsert('kako kao');
     pecatiSve;
     writeln(hashSearch('AAAA'));
     writeln(hashSearch('AAA'));
     readln;
end.
```

Отворено адресирање

Наместо да создаваме низа од листи, може да се искористи друг вид на Hash-функции. Тој вид се нарекува ***отворено адресирање*** (***анг.*** Open Addressing) . Кај овој тип на hash, се зема една единствена низа од покажувачи, кои на почетокот се иницијализирани на **NIL**. За бројот на полиња во низата ***m*** важи дека ***m*** > ***n*** , каде ***n*** е максималниот број на елементи што би можеле да ги запишеме во низата, односно hash-от. Ова значи дека сите елементи ќе бидат сместени во низата. Но како се постигнува тоа?

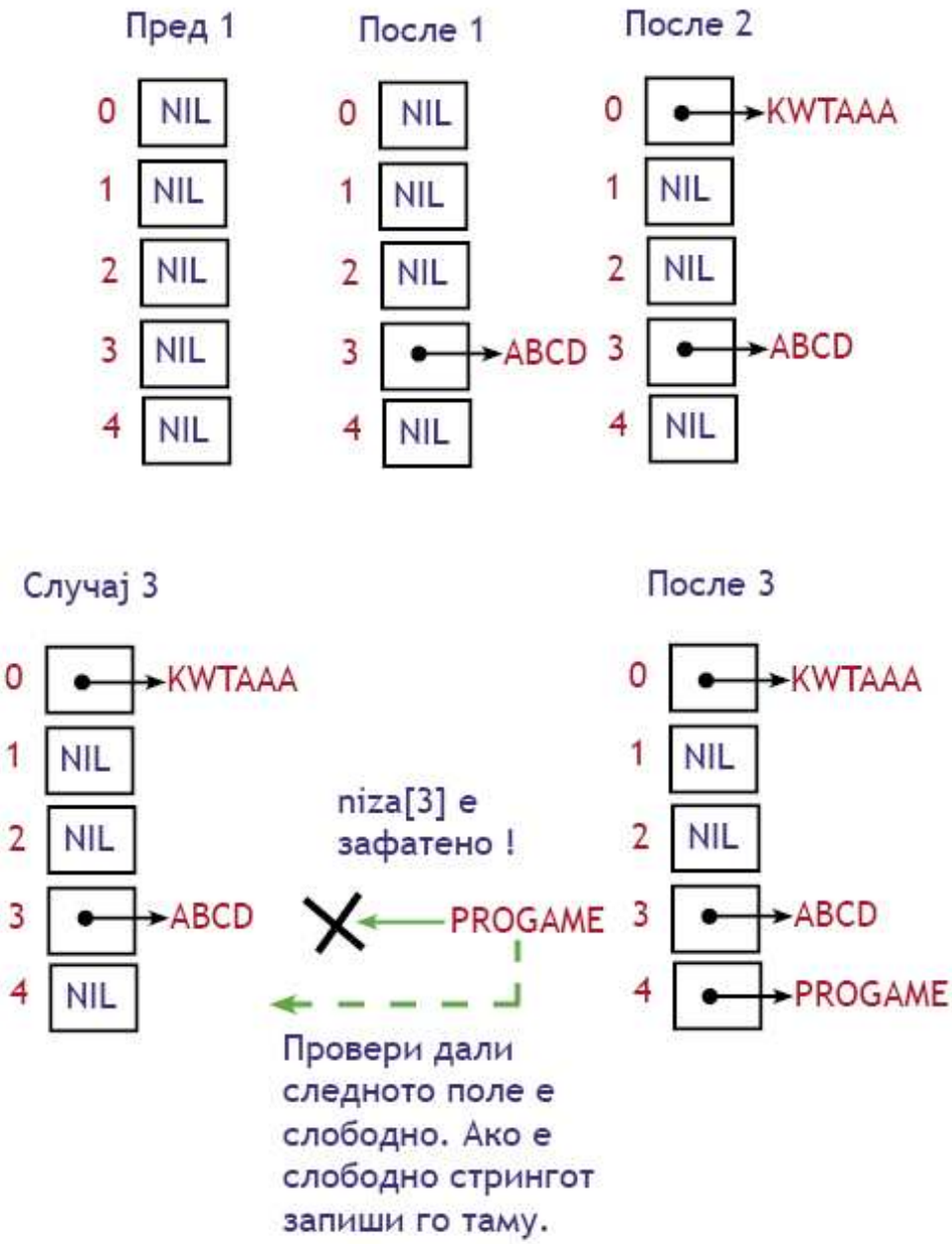
Нека е дадена низа со само 5 покажувачи иницијализирани на **NIL**. Покажувачите се индексирани со 0, 1, 2, 3, 4 . Во случајов повторно ќе сместуваме стрингови, па за да го одредиме индексот на полето во кое ќе го сметиме стрингот, ја користиме функцијата ***StrToIntPJW(s) mod m***, каде во случајов ***m*** = **5**.

Да ја разгледаме следнава секвенца наредби:

1. `s = 'ABCD'`
   `indeks = StrToIntPJW(s) mod m;` → **3**
   `niza[3]` Слободно место ! Запиши го стрингот на тоа место !

2. `s = 'KWTAAA'`
   `indeks = StrToIntPJW(s) mod m;` → **0**
   `niza[0]` Слободно место ! Запиши го стрингот на тоа место !

3. `s = 'PROGAME'`
   `indeks = StrToIntPJW(s) mod m;` → **3**
   `niza[3]` Зафатено место ! Оди на следно слободно место !
   Дали `niza[4]` е слободно место ?
   Слободно е ! Запиши го стрингот на тоа место !



Слика 4

Уште да забележиме за претходниот пример дека доколку полето **niza[4]** не беше слободно, следно за проверка ќе беше полето **niza[0]**.

Пребарувањето на одреден стринг се прави на сличен начин: Најпрво се пресметува индексот за даденият стринг. Доколку пресметаниот индекс посочува на поле од низата кое е празно(**NIL**), тоа ќе значи дека бараниот стринг воопшто го нема во hash структурата. Но доколку во посоченото поле е запишан стринг, се проверува дали тој стринг се совпаѓа со бараниот стринг. Доколку тоа не резултира со погодок, се повторува истата постапка со следното поле во низата.

Така за примерот прикажан погоре, сакаме да провериме дали во hash-структурата се наоѓа стрингот ***PROGAME***. Индексот на ***PROGAME*** се пресметува како ***StrToInt('PROGAME') mod m*** , што за горниот пример изнесува 3. Затоа одиме во полето **niza[3]**. Во полето е запишан стринг и затоа го споредуваме со стрингот ***PROGAME***. Но **niza[3]** = ***ABCD*** е различно од ***PROGAME*** и затоа истата постапка ја продолжуваме со следното поле во низата, а тоа е **niza[4]**. За полето **niza[4]** добиваме потврден одговор, и со тоа го завршуваме пребарувањето.

Бришењето на елементи од ваквата hash – структура не е воопшто лесно. Размислете дали би можело да се пребара стрингот ***PROGAME*** доколку претходно го избришеме стрингот ***ABCD*** од полето **niza[3]** и во истото запишеме ***NIL*** ?

Од ова може да се заклучи дека hash структурата со отворено адресирање е погодна за ***внесување***(*анг.* Insert) и ***пребарување***(*анг.* Search) на стрингови, но не и за нивно бришење.

Значи пред да ја прикажеме програмата со која е изградена hash – структурата со отворено адресирање, ќе ги повториме следниве работи кои се важни да се запамтат:

- Бројот на полиња ***m*** во низата од листи и низата за отворено адресирање е ***ПРОСТ*** број. Во случајот на низата од листи важи ***m*** < ***n***, а во случајот на низа за отворено адресирање ***m*** > ***n***, каде ***n*** е бројот на елементи кој мислиме да го внесеме во hash – структурата.
- Низите со ***m*** полиња ги индексираме со ***0,1,2,...,m-1***. Доколку во hash – структурата сместуваме некаков запис кој е составен од повеќе други податоци, тогаш за истиот тој запис мора да генерираме број ***d***, што значи дека овој запис ќе се најде во полето од низата (или во листата закачена за тоа поле) со индекс ***d mod m***.

Да споменеме уште една работа која не треба да се изостави кога се спомнува техниката на отворено адресирање а тоа е следното: Никогаш не смее да се внесува нов елемент во преполнета низа наменета за отворено адресирање. Тоа е затоа што за пребарување и внесување нов елемент при ваков случај ќе се потребни повеќе чекори додека не се дојде до празно поле. Затоа при избор на големината на hash структурата, ***m***, треба да се запазат следниве својства: ***m*** е прост број и $\frac{2}{3}m \geq n$, или во краен случај $\frac{3}{4}m \geq n$. Ова значи дека максимално дозволената исполнетост на низата, при која што пребарувањето сеуште ќе се смета за ефикасно, изнесува 75%. За поподробни анализи и разгледи види особено [1] стр. 221-252.и може [2] стр. 227 – 276.

```
program linearProbing;
const
     KAPACITET=13;{m=13}
type
    pok=^string;
var
   hash:array[0..KAPACITET-1]of pok;{Ovaa hash tabela e pogodna za
   vkupno 2/3*KAPACITET elementi}

function StrToIntPJW(const kluc:string):longint;{P.J.Weinberger}
```

```
var
   G,hashVrednost:longint;
   i:integer;
begin
     hashVrednost:=0;
     for i:=1 to length(kluc) do
         begin
              hashVrednost:=(hashVrednost shl 4) + ord(kluc[i]);
              G:=hashVrednost and $F0000000;
              if (G<>0)
                 then
                     hashVrednost:=hashVrednost xor (G shr 24) xor G;
         end;
     StrToIntPJW:=hashVrednost;
end;

function hashInsert(s:string):integer;
var
   pom:pok;
   i,j,Indeks:integer;
begin
     Indeks:=StrToIntPJW(s) mod m;
     i:=0;
     while (i<KAPACITET) do
{i e ograniceno so KAPACITET zatoa sto proverkata za naoganje na prazno mesto ne smee
da trae poveke otkolku sto ima raspolozlivi polinja vo nizata }
           begin
                if hash[Indeks]=NIL{ako indeks posocuva na pole koe e prazno, togas
tamu go zapisuvame stringot sto go vnesuvame}
                  then
                      begin
                           new(pom);
                           pom^:=s;
                           hash[Indeks]:=pom;
                           break;
                      end
                  else{dokolku indeksot posocuva na pole koe ne e prazno togas samo
go zgolemuvame i: ova znaci odime na sledno pole}
                           inc(i);

                Indeks:=Indeks+1;{indeks go zgolemuvame za 1: znasi odime na sledniot
indeks no vnimavame da ne sme go nadminale indeksot na poslednoto pole a toa e m-1}
                if Indeks=KAPACITET {ako indeksot e m-1 togas sledno pole za proverka
e prvoto pole od nizata koe e indeksirano so 0}
                  then
                      Indeks:=0;
           end;
     hashInsert:=Indeks;
end;

function hashSearch(s:string):integer;
var
   i,j,Indeks:integer;
   dali:boolean;
begin
     Indeks:=StrToIntPJW(s) mod m;
     i:=0;
     dali:=false;
     while (hash[Indeks]<>NIL)and(i<KAPACITET) do{(hash[Indeks]<>NIL): ovoj del od
uslovot stoi bidejki prebaruvame se dodeka ne dojdeme do prazno NIL pole. Dokolku
dojdeme do NIL pole toa ke znaci deka ne sme go nasle stringot}
```

```
          begin
               if hash[Indeks]^=s{ako indeks posocuva na pole vo koe se naoga nasiot
string togas POGODOK}
                  then
                      begin
                           dali:=true;{boolean promenliva koja ke ni kaze dali
prebaruvanjeto e uspesno}
                           break;
                      end
                  else
                      inc(i);{ako ne posocuvanjeto ne rezultira so pogodok odime na
sledno pole od nizata }

               Indeks:=Indeks+1;
               if Indeks=KAPACITET
                  then
                      Indeks:=0;
          end;
     if dali{ako prebaruvanjeto e uspesno, togas}
        then
            hashSearch:=Indeks{vrati go indeksot na poleto vo koe e zapisan stringot
sto go prebaruvame}
        else
            hashSearch:=-1;{inaku vrati -1}
end;

procedure initHash;{procedura za inicijalizacija na nizata za otovreno adresiranje}
var
   i:integer;
begin
     for i:=0 to KAPACITET-1 do
         hash[i]:=NIL;
end;

procedure pecati;
var
   i:integer;
begin
     writeln('HASH');
     for i:=0 to KAPACITET-1 do
         if hash[i]=NIL
            then
                writeln(i:3,' : ':4)
            else
                writeln(i:3,' : ':4,hash[i]^,' (',StrToIntPJW(hash[i]^),')');
end;


begin
     initHash;
     hashInsert('nikola lazovski niksa');
     hashInsert('aaa');
     hashInsert('cccc');
     hashInsert('jhdjsagfi');
     pecati;
     writeln(hashSearch('niksa'));
     writeln(hashSearch('aaa'));
     readln;
end.
```

Hash структурата со отворено адресирање има различна имплементација која зависи пред се од изборот на hash функцијата.

Така, во претходниот пример беше реализирана само една позната имплементација на отвореното адресирање уште попозната под името ***линеарно испробување*** (*анг*. linear probing).

Од претходната програма може да се забележи дека, во случајот кога во hash структурата се внесува нов стринг, почнувајќи од некое почетно поле се испитуваат полињата се додека не се дојде до некое празно поле(**NIL**).

Но еве да замислиме дека целата структура е полна, т.е. нема ниту едно **NIL** поле и сакаме да внесеме нов стринг. Во тој случај, почнувајќи од некое почетно поле(почетен индекс), ќе проверуваме дали постои празно поле. Во случајов, ние ќе ги поминеме сите полиња односно сите индекси и накрај нема да најдеме празно поле.

Значи при пребарување на празно поле, па дури и при пребарување на елемент во hash структура со отворено адресирање, потребен ни е редослед на индексите по кој ќе се врши пребарување на полињата во hash структурата со отворено адресирање. Тој редослед на индексите од низата со која е претставена hash-структурата претставува пермутација на индексите ***0,...,m-1***.

Така, во погорната програма, поточно во функцијата ***hashInsert*** променливата ***i*** може да добие вредност од **0** до **KAPACITET-1**, односно од ***0*** до ***m-1***. За ***i = 0***, полето кое го испитуваме е **hash[pocetenIndeks]**. За ***i = 1***, полето кое го испитуваме е **hash[pocetenIndeks + 1]** итн. Доколку формализираме, за одредена вредност на ***i***, ние го разгледуваме полето од низата **hash** со индекс ***(pocetenIndeks + i) mod m***, каде ***pocetenIndeks*** се пресметува со функцијата ***StrToIntPJW(s)***.

Значи за да добиеме индекс на поле кое ќе го разгледуваме ја користиме функцијата

$$h(s,i)=(StrToIntPJW(s) + i) \bmod m.$$

Но линеарното испробување има свои недостатоци, заради тоа што создава т.н. ***кластери*** односно групи од пополнети полиња во низата. За повеќе информации види [1],[2].

Подобрување на претходната имплементација е наречена ***двојно хеширање*** (*анг*. double hashing) кое го користи истиот принцип на функционирање, како и линеарното испробување, само што користи друга hash функција која е дефинирана на следниов начин:

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

каде

$$h_1(s) = StrToIntPJW(s) \bmod m$$

$$h_2(s) = 1 + (StrToIntPJW(s) \bmod (m-1)$$

Во случајот на двојно хеширање има значително олеснување при изборот на ***m***. За ***m*** може да се избере произволен број, тоа значи дека **m** немора да е прост број. Најпогодно е за ***m*** да се избере вредност од облик $\mathbf{2^p}$, затоа што во тој случај за делење или множење на број со вака избрана вредност ќе се искористат операторите ***shl*** и ***shr***, односно операторите за поместување на битовите од деленикот за ***p*** места на лево односно на десно. Во програмскиот јазик ***C++*** , како и во ***Java*** овие оператори го имаат обликот << и >>.

За повеќе информации во врска со изборот на токму ваквата дефиниција на hash функцијата види [1] стр. 241.

Програмата што следува е пример за двојно хеширање:

```
program doubleHashing;
```

```
const
     KAPACITET=16;{m = 2^p}
     STEPEN=4;{za izrazuvanje na 16 mesta ni se potrebni 4 bita}
type
    pok=^string;
var
   hash:array[0..KAPACITET-1]of pok;{Ovaa hash tabela e pogodna za
   vkupno 2/3*16 elementi}

function dajHashVrednostPJW(const kluc:string):longint;{P.J.Weinberger}
var
   G,hashVrednost:longint;
   i:integer;
begin
     hashVrednost:=0;
     for i:=1 to length(kluc) do
         begin
              hashVrednost:=(hashVrednost shl 4) + ord(kluc[i]);
              G:=hashVrednost and $F0000000;
              if (G<>0)
                 then
                     hashVrednost:=hashVrednost xor (G shr 24) xor G;
         end;
     dajHashVrednostPJW:=hashVrednost;
end;

function hashInsert(s:string):integer;
var
   pom:pok;
   i,Indeks,h1,h2:integer;
begin
     Indeks:=dajHashVrednostPJW(s);{presmetka na pocetniot indeks}
     h1:=((Indeks shr STEPEN)shl STEPEN) xor Indeks;{presmetuvanje na ostatok pri
delenje na pocetniot indeks so KAPACITET odnosno m. Ova e mozno samo vo slucajot za
h1. Zosto?}
     h2:=1+(Indeks mod (KAPACITET-1));{presmetka na h2}
     Indeks:=h1;{se pocnuva od indeks h1 + i*h2 = h1 + 0*h2 = h1}
     i:=0;
     while (i<KAPACITET) do
           begin
                if hash[Indeks]=NIL
                   then
                       begin
                            new(pom);
                            pom^:=s;
                            hash[Indeks]:=pom;
                            break;
                       end
                   else
                       inc(i);

                Indeks:=Indeks+h2;{bidejki funkcijata h(s,i) opfaka del h1(s)+i*h2(s),
za sekoe naredno i ke dodavame uste po edno h2}
                if Indeks>=KAPACITET
                   then
                       Indeks:=Indeks-KAPACITET;
           end;
     hashInsert:=Indeks;
end;

function hashSearch(s:string):integer;
```

```
var
   i,Indeks,h1,h2:integer;
   dali:boolean;
begin{prebaruvanjeto odi na slicen nacin kako i vnesuvanjeto stringovi}
     Indeks:=dajHashVrednostPJW(s);
     h1:=((Indeks shr STEPEN)shl STEPEN) xor Indeks;
     h2:=1+(Indeks mod (KAPACITET-1));
     Indeks:=h1;
     i:=0;
     dali:=false;
     while (hash[Indeks]<>NIL)and(i<KAPACITET) do
           begin
                if hash[Indeks]^=s
                   then
                       begin
                            dali:=true;
                            break;
                       end
                   else
                       inc(i);

                Indeks:=Indeks+h2;
                if Indeks>=KAPACITET
                   then
                       Indeks:=Indeks-KAPACITET;
           end;

     if dali
        then
            hashSearch:=Indeks
        else
            hashSearch:=-1;
end;

procedure initHash;{procedura za inicijalizacija na hash strukturata}
var
   i:integer;
begin
     for i:=0 to KAPACITET-1 do
         hash[i]:=NIL;
end;

procedure pecati;
var
   i:integer;
begin
     writeln('HASH');
     for i:=0 to KAPACITET-1 do
         if hash[i]=NIL
            then
                writeln(i:3,' : ':4)
            else
                writeln(i:3,' : ':4,hash[i]^,' (',dajHashVrednostPJW(hash[i]^),')');
end;


begin
     initHash;
     hashInsert('nikola lazovski niksa');
     hashInsert('aaa');
     hashInsert('cccc');
```

```
    hashInsert('jhdjsagfi');
    pecati;
    writeln(hashSearch('niksa'));
    writeln(hashSearch('aaa'));
    readln;
end.
```

Најдобри перформанси од претставените имплементации дава двојното хеширање.

Користена литература

[1] Introduction to algorithms – Thomas H. Corman, Charles E. Leiseron, Ronald L. Rivest, Clifford Stein; MIT Press, 2001.
[2] The tomes of Delphi: Algorithms and Data Structures – Julian Bucknall; Wordware Publishing, 2001.