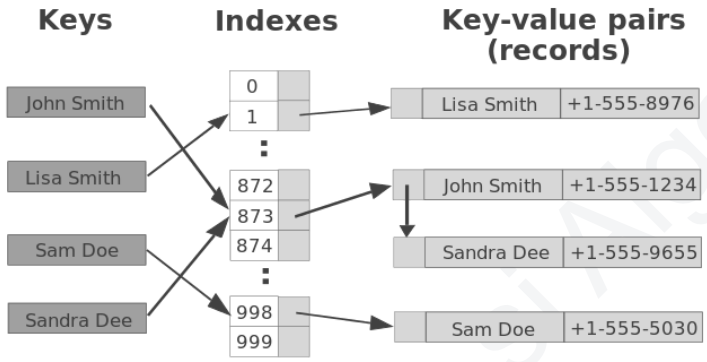
Tabele de dispersie (Hash Tables)

1. Considerații teoretice

În această lucrare sunt prezentate principalele operaţii asupra unei tabele de dispersie: construirea tabelei de dispersie, inserarea unei înregistrări, căutarea unei înregistrări, afişarea înregistrărilor. De asemenea se fac câteva consideraţii asupra alegerii funcţiei de dispersie.



1.1 Tipuri de tabele

Tabelul este o **colecţie de elemente de acelaşi tip**, identificabile prin **chei**. Elementele sale se mai numesc **înregistrări**.

Tabelele pot fi :

- **fixe**, cu un număr de înregistrări cunoscut dinainte (in momentul crearii);
- **dinamice,** cu un numar variabil de elemente.

Tabelele dinamice pot fi organizate sub formă de:

- listă dinamică simplu sau dublu înlănţuită;
- arbore de căutare;
- tabele de dispersie.

Din categoria **tabelelor fixe** face parte tabelul de cuvinte rezervate dintr-un limbaj de programare. Acesta este organizat ca un tablou de pointeri spre cuvintele rezervate, introduse în ordine alfabetică. Căutarea utilizată este cea binară.
Tabelele dinamice organizate sub formă de liste au dezavantajul căutării liniare. Arborele de căutare reduce timpul de căutare. În cazul în care cheile sunt alfanumerice, comparaţiile sunt mari consumatoare de timp. Pentru astfel de situaţii, cele mai potrivite sunt tabelele de dispersie.

1.2 Funcţia de dispersie (hashing)

Funcţia de dispersie este o funcţie care transformă o cheie într-un număr natural numit cod de dispersie sau valoare hash:

f: K -> H

unde K este mulţimea cheilor, iar H este o mulţime de numere naturale.

Funcţia **f** nu este injectivă. Două chei pentru care $\mathbf{f(k_1)=f(k_2)}$ se spune că intră în ***coliziune,*** iar înregistrările respective se numesc sinonime.

Asupra lui **f** se impun două condiţii:

- valoarea ei pentru orice cheie data să rezulte cât mai simplu şi rapid;
- să minimizeze numărul de coliziuni.

1.2.1 Dispersie modulară

Un exemplu de funcţie de dispersie modulară este următoarea:

$$\mathbf{f(k)=\gamma(k) \bmod M}$$

unde ***γ(k)*** este o funcţie care transformă cheia într-un număr natural, iar ***M*** este un număr natural recomandat a fi prim.

Funcţia ***γ(k)*** se alege în funcţie de natura cheilor. Dacă ele sunt numerice, atunci aceasta poate fi **γ(k)=k**.

În cazul cheilor alfanumerice, cea mai simplă funcţie ***γ(k)*** este suma codurilor ASCII ale caracterelor din componenţa lor; ca urmare funcţia ***f*** de calcul a dispersiei este următoarea:

```C/C++
#define M …
int f(char *key)
{
  int i,suma;
  suma=0;
  for(i=0;i<length(key);i++)
       suma=suma+*(key+i);
  return suma%M;
}
```

Valoarea lui ***M*** reprezintă dimensiunea tabelei de dispersie și are o influență semnificativă asupra modului în care funcția reușește să disperseze cheile în tabelă. Este important ca aceasta dispersie să fie cât mai uniformă pentru a evita clusterizarea: aglomerarea sinonimelor (coliziunilor) în câteva poziții din tabelă.

Se recomandă utilizarea unor valori prime pentru ***M***, cât mai depărtate de o putere a lui 2.

1.2.2 Dispersie prin metoda înmulțirii

Un exemplu de astfel de funcție de dispersie este

$$\mathbf{f(k) = [M * \{k * A\}]}, \qquad \text{cu } 0 < A < 1,$$

unde **[v]** reprezintă partea întreagă a lui v, iar **{v}** reprezintă partea fracționară a lui **v**, adică **{v} = v – [v]**.

Se observă că funcția f produce numere între 0 și M-1:

$$\mathbf{0 \le \{k*A\} < 1} \text{ și } \mathbf{0 \le M * \{k*A\} < M}.$$

Observație:
Valoarea lui M nu mai are o mare importanţă. M poate fi cât de mare ne convine, eventual o putere a lui 2. În practică, s-a observat că dispersia este mai bună pentru unele valori ale lui A şi mai proastă pentru altele.

Donald Knuth propune valoarea **A=(sqrt(5)-1)/2 ≈ 0,618034**.

1.2.3 Dispersie prin metoda înmulțirii

C/C++

Knuth

```
//char *s -> cheia
len = strlen(s);
for (int h = len; len--; )
	h = ((h << 7) ^ (h << 27)) ^ *s++;
```

Bernstein

```
h = 5381;
while (c = *s++)
	h += (h << 5) + c;
```

SDBM

```
for ( h = 0; c = *s++;)
	h = (h << 6) + (h << 16) – h + c;
```

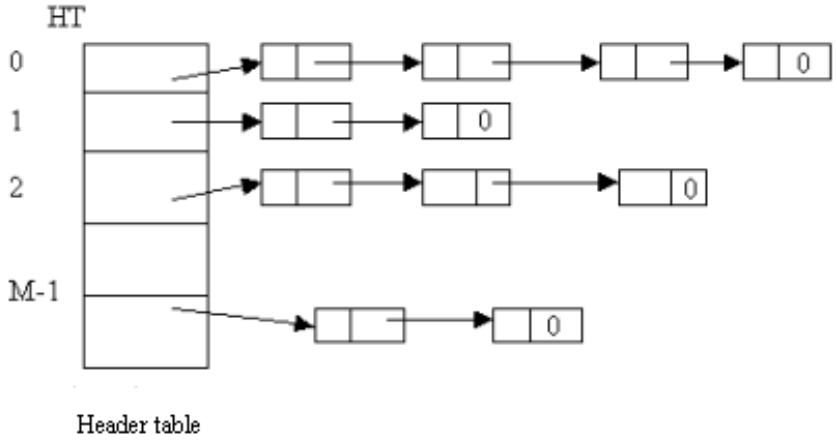
CRC

```
h = 0;
len = strlen(s);
for (i = 0; i < len; i++) {
	//shiftare circulara la stanga cu 5 pozitii
	highorder = h & 0xf8000000;
	h << 5;
	h = h ^ (highorder >> 27);
	//---> end shiftare
	h = h ^ s[i];
}
h = h % M;
```

1.3 Rezolvarea coliziunilor în tabela de dispersie

În funcție de tipul tabelei de dispersie, deschisă vs. închisă, se definesc strategii specifice pentru rezolvarea coliziunilor. În cele ce urmează ne vom referi la rezolvarea coliziunilor în **tabele deschise**.

Rezolvarea coliziunilor se face astfel: toate înregistrările pentru care cheile intră în coliziune sunt inserate într-o listă simplu înlănţuită. Vor exista astfel mai multe liste, fiecare conţinând înregistrări cu acelaşi cod de dispersie. Pointerii spre primul element din fiecare listă se păstrează într-un tablou, la indexul egal cu codul de dispersie. Ca urmare, modelul unei tabele de dispersie deschisă este următorul:



Un nod al listei are structura următoare:

```C/C++
struct Nod {
char *cheie;
// alte informatii utile
//...
struct Nod *urm;
};
```

Tabloul HT (hash table) este declarat astfel: **Nod *HT[M];**
Iniţial el conţine pointerii nuli:

```
for(i = 0; i < M; i++)
    HT[i] = 0;
```

1.4 Operații în tabela de dispersie

Căutarea într-o tabelă de dispersie a unei înregistrări având pointerul **key** la cheia sa, se face astfel:

- se calculează codul de dispersie:

  h = f(key);

- se caută înregistrarea având pointerul **key** la cheia sa, din lista având pointerul spre primul nod HT[h].

Căutarea este liniară:

```C/C++
p = HT(h);
while(p != 0)
  {
            if(strcmp(key, p->cheie) == 0) return p;
            p = p->urm;
  }
     return 0;
```

Inserarea unei înregistrări într-o tabelă de dispersie se face astfel:

```C/C++
//(1) se construieşte nodul de pointer p, care va conţine informaţia utilă şi
pointerul la cheia înregistrării:

p = new Nod;
citire_nod(p);

//(2) se determină codul de dispersie al înregistrării:
h = f(p->cheie);

//(3) dacă este prima înregistrare cu codul respectiv, adresa sa este depusă în
tabelul HT:
if(HT[h] == 0)
{
              HT[h] = p;
              p->urm = 0;
       }
//(4) în caz contrar se verifică dacă nu cumva mai există  o înregistrare cu
cheia respectivă. În caz afirmativ se face o prelucrare a înregistrării
existente ( ştergere, actualizare) sau este o eroare (cheie dublă ). Dacă nu
există o înregistrare de cheia respectivă, se inserează în listă ca prim
element nodul de adresă p:
q = cautare(p->cheie);
if(q == 0){  /* nu exista o înregistrare cu cheia respectiva */
             p->urm = HT[h];
             HT[h] = p;
}
else prelucrare(p,q);/* cheie dubla */
```

Construirea tabelei de dispersie se face prin inserarea repetată a nodurilor.

Listarea tuturor înregistrărilor pe coduri se face simplu, conform algoritmului următor:

```
C/C++
for(i = 0; i < M; i++){
       if(HT[i] != 0){
              cout << "Inregistrări avand codul de dispersie" << i << endl;
              p = HT[i];
              while(p != 0){
                     afisare(p);
                     p = p->urm;
              }
        }
}
```

1.5 Performanțele unei tabele de dispersie

Se definește **factorul de încărcare** al unei tabele de dispersie **α = N / M**, unde N este numărul de elemente din colecția de date iar M este dimensiunea tabelei de dispersie.

În cazul tabelelor de dispersie deschise, α reprezintă lungimea ideală a listei înlănțuite asociate unei valori hash (se obține în cazul unei dispersii uniforme). Costul căutării unei chei în tabela de dispersie este atunci O(1 + α). Dacă α este mărginită superior de o valoare fixă, α_{max}, atunci operația de căutare rămâne în O(1).

În practică vom obține cele mai bune performanțe cand α se păstrează într-un interval îngust [0.5, 2].

Dacă α < 0.5, atunci tabela hash va avea multe goluri. Îmbunătățirea performanței se poate face micșorând dimensiunea acesteia.

Dacă α > 2, atunci costul traversării listelor înlănțuite limitează performanța tabelei.

În concluzie, alegerea valorii M, a dimensiunii tabelei, reprezintă un factor important.

În mod ideal, funcția hash definită trebuie testată pentru a verifica dacă se comporta bine cu date reale. O modalitate este de a măsura **gradul de clusterizare** a elementelor:

$$C = \sum_{i=0}^{M-1} \frac{x_i^2}{N} - \alpha,$$

unde x_i reprezintă numărul de chei mapate pe valoarea hash (indexul) **i**.

O funcție hash uniformă (care dispersează uniform înregistrările în tabela hash) produce o clusterizare aproape de 1.0 cu probabilitate mare.

Un factor C > 1 indică faptul că tabela hash este afectată de clusterizare.

2. Aplicații

1. Implementaţi un program care să verifice corectitudinea scrierii unor cuvinte dintr-un text folosind o tabelă de dispersie. Fişierul *dictionar.txt* ce însoțește laboratorul conţine un set

de cuvinte ce va constitui dicţionarul. Dacă un cuvânt este găsit în dicţionar, se presupune că acesta a fost scris corect. Altfel, se presupune a fi incorect scris.

a. stabiliți dimensiunea tabelei de dispersie (valoarea M) pe baza analizei factorului de încărcare
b. construirea tabelei de dispersie şi implementarea funcţiei hash bazate pe dispersia modulară
c. introducerea în tabelă a cuvintelor din dicţionar pe baza codului de dispersie (funcţia de inserare) şi afişarea tabelei
d. testarea corectitudinii scrierii cuvintelor dintr-un text citit de la intrarea standard sau dintr-un fişier (funcţia de căutare)
e. scrieți o funcție de ștergere a unei înregistrări de cheie dată.

Obs.: Rezolvarea coliziunilor se va face prin înlănțuire.

2. Analizati performanța tabelei de dispersie prin măsurarea gradului de clusterizare
   a. implementați o funcție de calcul al gradului de clusterizare pentru o tabelă dată
   b. implementați 3 variante de funcție de dispersie (modulară, prin înmulțire și o metoda cu deplasare pe biți)
   c. variați dimensiunea tabelei de dispersie și funcția de dispersie și analizați cum evoluează gradul de clusterizare. Completați valorile obtinute pt. gradul de clusterizare în următorul tabel:

	f1	**f2**	**f3**
M1 =			
M2 =			
M3 =			

3. Să se afişeze frecvenţa de apariţie a literelor dintr-un text utilizând o tabelă de dispersie.
   Intrarea va fi textul.
   Ieşirea va fi următoarea:
   Aa 11
   Bb 12
   Cc 1
   ….
   Zz 4

4. Reluați problema anterioară pentru a afişa frecvenţa de apariţie a cuvintelor dintr-un text utilizând o tabelă de dispersie.

Notare:
Problema 1 - **5p**
Problema 2 - **2p**
Problema 3 - **3p**
Problema 4 - **2p**

Aplicațiile neterminate în timpul orelor de laborator rămân ca teme pentru studiu individual!