

Laborator 6: Tabele De Dispersie

1 Obiective

În acest laborator vom lucra cu tabele de dispersie. În contextul tabelor de dispersie se vor studia următoarele operații: creare, inserare, căutare și ștergere a unor elemente.

2 Tipuri de tabele

Un *tabel* reprezintă o colecție de elemente de același tip. Aceste elemente sunt identificate prin intermediul unor *chei*. Tabelele sunt organizate în două moduri:

- *Tabele fixe* – în acest caz numărul de elemente este cunoscut de la început programului.
- *Tabele dinamice* - aceste tabele au un număr variabil de elemente.

Un exemplu de tabel fix poate fi considerat un tabel de maxim 7 elemente conținând toate zilele săptămânii. Tabelele dinamice pot fi organizate sub forma de liste liniare simplu înlanțuite, arbori de căutare sau tabele de dispersie. Dacă tabelele dinamice sunt organizate ca liste, căutările sunt efectuate liniar, lucru care încetinește operația de găsim a unui element. După cum am văzut în lucrarea precedentă, un arbore de căutare reduce timpul de găsim a unui element la un timp logaritm. În acest laborator vom merge mai departe și vom încerca să construim o structură de date în care găsim a unui element se va efectua în timp constant $O(1)$.

3 Funcții de dispersie

O *funcție de dispersie* este o funcție care face conversia unei chei într-un număr natural numit și *valoare de dispersie*. Funcția de dispersie este definită ca

$$f : K \rightarrow H,$$

unde K reprezintă un set de chei iar H o mulțime de numere naturale. Funcția f reprezintă o mapare *many to one*. Presupunând că avem două key, k_1 și k_2 cu $k_1 \neq k_2$ și rezultatul funcției de dispersie e același, de exemplu $f(k_1) = f(k_2)$, atunci se spune că între cele două chei apare o *coliziune* iar datele corespunzătoare sunt numite *înregistrări sinonime*. Două restricții sunt impuse asupra funcției f :

1. Pentru oricare $k \in K$, valoarea de dispersie trebuie să se obțină cât mai rapid.
2. Trebuie să minimizeze numărul de coliziuni.

Un exemplu de funcție de dispersie este:

$$f(k) = \gamma(k) \bmod B,$$

unde γ este o funcție care mapează o cheie la un număr natural, iar B un număr natural, dacă se poate un număr prim. Definirea funcției γ depinde de tipul cheilor. Dacă acestea sunt numerice, atunci $\gamma(k) = k$ poate fi o posibilă funcție de dispersie. Pentru cheile care reprezintă șiruri de caractere, o funcție simplă de dispersie ar fi suma codurilor ASCII al caracterelor din șir, ca de exemplu:

```

#define B ? /*suitable value for B */
int f( char *key )
{
    int i, sum;
    sum = 0;
    for ( i = 0; i < strlen( key ); i++ )
        sum += key[ i ];
    return( sum % B );
}

```

Alte exemple de functii de dispersie: suma cifrelor, metoda gruparii (folding method) etc.

4 Rezolvarea coliziunilor

În momentul în care două elemente sunt dispersate în aceeași poziție (slot) în tabela de dispersie, trebuie să se stabilească o metodă sistematică pentru plasarea celui de-al doilea element. Metodele cele mai comune de rezolvare a coliziunilor sunt metoda de înlănțuire (chaining) și adresarea deschisă.

4.1 Tehnica înlănțuirii (chaining)

Metoda chaining presupune inserarea tuturor cheilor care au aceeași valoare de dispersie într-o listă simplu înlănțuită. Tabela va conține în fiecare poziție o referință către o listă. Un exemplu de o astfel de tabelă este ilustrat în Fig. 1.

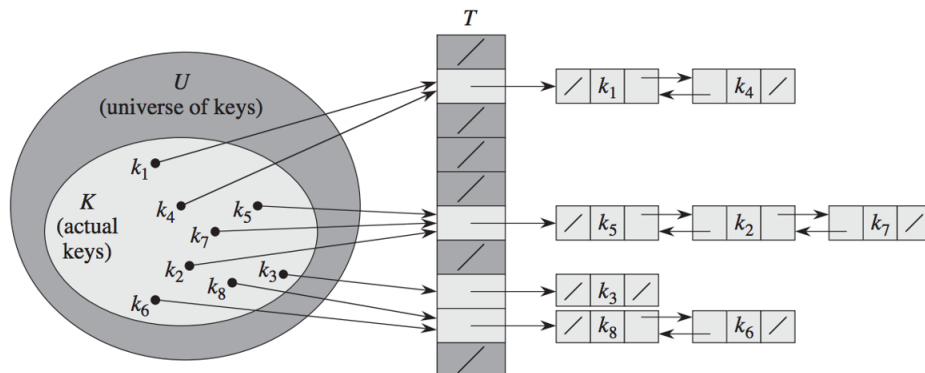


Figura 1: Tabele de dispersie: chaining

Operații:

- **Inserare** - Inițial toate sloturile tabelului sunt initializate cu NULL. În momentul inserării unui element x în tabela T , se calculează valoarea de dispersie $h(x.key)$ pentru a determina slotul la care se face inserarea, apoi se efectuează operația de inserare la începutul listei care se află în tabelă la poziția $h(x.key)$:
 $insert_first(T[h(x.key)], x)$
- **Căutarea** - Căutarea unei chei key în tabela T corespunde operației de căutare după cheie din lista de pe poziția $h(key)$:
 $list_search(T[h(x.key)], key)$
- **Ștergere** - Ștergerea unei chei key din tabela T corespunde ștergerii după cheie din lista de pe poziția $h(key)$:
 $list_delete_key(T[h(x.key)], key)$

4.2 Adresare deschisă

O altă metodă de rezolvare a coliziunilor este **adresarea deschisă** care presupune căutarea în tabela de dispersie a unui slot liber în care elementul ce a cauzat coliziunea poate fi stocat.

Exista o funcție de dispersie care generează o permutare a spațiului de adrese astfel încât la căutarea sau inserarea unui element în tabela de dispersie se probează sloturile libere în funcție de acea permutare. Exista următoarele tehnici de generare a secvențelor de proba:

1. linear probing (Verificare liniară)
2. quadratic probing (Verificare patratică)
3. double hashing (Dispersie dublă)

4.2.1 Adresare deschisă - verificare liniară

Pentru inserarea unei chei k se folosește următoarea funcție de dispersie pentru a genera locația în care se probează tabela de dispersie:

$$h(k, i) = (h'(k) + i) \bmod m \quad (1)$$

unde

- $h'(k)$ este o funcție de dispersie auxiliară
- $i = 0, 1, \dots, m - 1$ este o variabilă care crește atâta timp cât mai avem coliziuni.
- m este dimensiunea tabelului de dispersie

Fiind dată o cheie k locațiile vor fi examinate în ordinea următoare: $T[h(k, 0)]; T[h(k, 1)] \dots T[h(k, m - 1)]$.

Presupunând că avem următorul șir de numere întregi: 18, 6, 36, 72, 43 și funcția de hashing $f = \text{key}$ modulo m ca în Fig. 2, unde $m = 8$ reprezintă dimensiunea tabelului, **inserarea** valorilor 10, 5 și 15 se realizează astfel:

- Elementul de inserat va fi stocat în următorul loc liber din tabel (dacă acesta nu e plin).
- Se implementează o căutare liniară a unui loc liber începând din poziția în care a avut loc coliziunea.
- Dacă ajungem la sfârșitul fizic al tabelului, căutarea continuă cu începutul tabelului.
- Dacă nu s-a găsit niciun loc liber și s-a ajuns cu căutarea la locul coliziunii, atunci tabela este plină.

[0]	72	Add the keys 10, 5, and 15 to the previous table. Hash key = key % table size 2 = 10 % 8 5 = 5 % 8 7 = 15 % 8	[0]	72
[1]			[1]	15
[2]	18		[2]	18
[3]	43		[3]	43
[4]	36		[4]	36
[5]			[5]	10
[6]	6		[6]	6
[7]			[7]	5

Figura 2: Tabele de dispersie: adresare directă

Căutarea unui element după o cheie dată începe din locul unde a avut loc coliziunea și se parcurge tabela verificând elementele de pe pozițiile date de funcția de dispersie. Dacă parcurgerea ajunge din nou la locul coliziunii, înseamnă că elementul căutat nu se află în tabelă.

Problema care apare la verificarea liniară este data de faptul că două chei care sunt mapate inițial la adrese diferite pot concura pentru aceleași locații în iterații succesive. Acest fenomen este cunoscut sub numele de **clusterizare primară**. Astfel se formează șiruri lungi de locații ocupate.

4.2.2 Adresare deschisă - verificare polinomială – quadratic probing

Verificarea polinomială folosește o funcție de dispersie de forma:

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

unde:

- $h'(k)$ este o funcție de dispersie auxiliara
- $c_1! = 0$ și $c_2! = 0$ sunt constante auxiliare
- $i = 0, 1 \dots m - 1$ este o variabilă care crește atâta timp cât mai avem coliziuni.
- m este dimensiunea tabelului de dispersie

Se va verifica astfel ca primă poziție $T[h(k, 0)]$, urmată de poziții care depind quadratic de i .

Și la această metodă apare problema de **clusterizare secundară** dacă $h(k_1, 0) = h(k_2, 0)$. Astfel două chei care sunt mapate inițial la aceleași adrese pot concura pentru aceleași locații în iterații succesive.

4.2.3 Adresare deschisă - verificare dublă – double hashing

Verificarea dublă folosește ideea aplicării a două funcții de dispersie atunci când are loc o coliziune. Această abordare este o alternativă foarte bună deoarece permutările generate au proprietăți apropiate de dispersie uniformă. Funcția de dispersie este de forma:

$$h(k, i) = (h_1(k) + i \times h_2(k)) \bmod m \quad (3)$$

unde:

- $h_1(k)$ și $h_2(k)$ sunt funcții de dispersie auxiliare.
- $i = 0, 1 \dots m - 1$ este o variabilă care crește atâta timp cât mai avem coliziuni.
- m este dimensiunea tabelului de dispersie

Astfel la o coliziune va fi probată locația $T[h_1(k)]$, urmată de poziții incrementate cu $h_2(k) \bmod m$ și avem astfel un offset variabil.

5 Tabele de dispersie pentru diferite tipuri de date

În toate operațiile din exemplele de mai sus s-au folosit chei numerice cu valori mici. Se pune problema definirii unui comportament pentru cazul în care se dorește stocarea unor date diverse cum ar fi cuvinte, nume de persoane, adrese, date despre cărți, date despre filme, etc. Se dorește atribuirea unei chei pentru valoarea fiecărui obiect stocat astfel încât căutarea să se facă rapid (dorim să stocăm perechi de tipul cheie/valoare). Imaginați-vă ca doriți să împrumutați o carte de la o bibliotecă în care sunt peste 1 milion de cărți. O căutare rapidă a unei cărți minimizează semnificativ timpul de așteptare. Pentru a stoca perechi de tipul cheie/valoare se pot folosi șiruri pentru care cheile reprezintă indici. Dar ce se întâmplă dacă indicii sunt foarte mari (de ordinul milioane). În acest caz o soluție elegantă pentru stocarea perechilor cheie/valoare este data de tabele de dispersie.

În aceste cazuri funcția de dispersie este compusă din 2 parti:

- **Cod de dispersie:** $h_1 : \text{chei} \rightarrow \text{intregi}$
- **Funcție de compresie:** $h_2 : \text{intregi} \rightarrow [0, m - 1]$, unde h_2 este o funcție de genul celor utilizate la verificarea liniară, quadratică sau dublă.

Indicele la care se va încerca inserarea / căutarea în tabelă va fi dat de: $h(x) = h_2(h_1(x))$.

6 Mersul lucrării

6.1 Probleme obligatorii

Ex. 1 — Să se scrie un program care ilustrează lucrul cu o tabelă de dispersie folosind metoda chaining de rezolvare a coliziunilor. Operațiile care trebuie să fie implementate sunt:

- inserarea în tabelă a unui nod cu cheia key

```
void insertElement(NodeT* hTable[M], int key)
```

- căutarea unui nod cu cheia key în tabelă

```
NodeT* FindElement(NodeT* hTable[M], int Key)
```

Funcția returnează adresa nodului cu cheia key sau NULL dacă această cheie nu este în arbore.

De exemplu pentru următoarele date:

- $m = 7$
- $h(k) = k \bmod m$
- Insert 36, 14, 26, 21, 5, 19, 4

tabela va arata ca în figura 3.

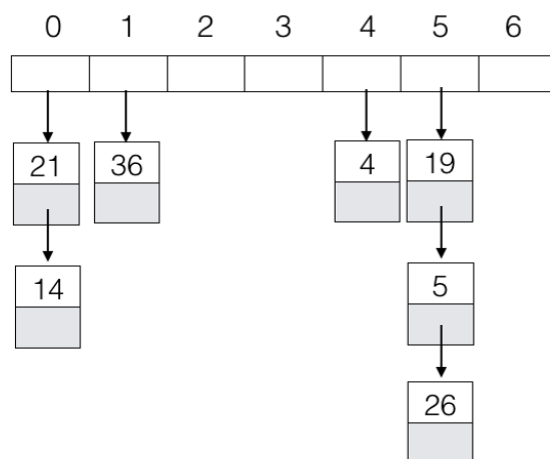


Figura 3: Exemplu Chaining

Ex. 2 — Sa se scrie un program care ilustreaza lucrul cu o tabela de dispersie folosind metoda de adresare deschisa de rezolvare a coliziunilor. Sa se foloseasca **verificarea liniara** ca functie de dispersie – vedeti BoilerPlateCode cu urmatoarele date:

- $m = 7$
- $h_0(k) = k \bmod m$
- functia de dispersie: $h(k; i) = (h_0(k) + i) \bmod m$
- dupa succesiunea de inserare a numerelor: 19, 36, 5, 21, 4, 26, 14 tabela arata ca in figura 4.

21	36	26	14	4	19	5
0	1	2	3	4	5	6

Figura 4: Rezultat verificare liniara

Ex. 3 — Sa se scrie un program care ilustreaza lucrul cu o tabela de dispersie folosind metoda de adresare deschisa de rezolvare a coliziunilor. Sa se foloseasca functie de dispersie cu **verificare polinomiala (cuadratică)**. IMPLEMENTATI functia de dispersie, functia de inserare si functia de cautare a unei chei. Utilizati urmatoarele date:

- $m = 7$
- $h'(k) = k \bmod m$
- Functia de verificare polinomiala - functia de dispersie: $h(k, i) = (h'(k) + c_1 \times i + c_2 \times i^2) \bmod m$ unde $c_1 = 1$, $c_2 = 1$
- dupa succesiunea de inserare a numerelor 19, 36, 5, 21, 4, 26, 14 tabela arata ca in figura 5.

5	36	21	26	4	19	14
0	1	2	3	4	5	6

Figura 5: Verificare polinomiala - rezultat

Ex. 4 — Sa se scrie un program care ilustreaza lucrul cu o tabela de dispersie folosind metoda de adresare deschisa de rezolvare a coliziunilor. Sa se foloseasca **verificarea dubla** ca functie de dispersie. IMPLEMENTATI functia de dispersie, functia de inserare si functia de cautare a unei chei. Utilizati urmatoarele date:

- $m = 7$
- Functia de verificare dubla - functia de dispersie: $h(k, i) = (h_1(k) + i \times h_2(k)) \bmod m$
- $h_1(k) = k \bmod m$
- $h_2(k) = 5 - (k \bmod m)$
- dupa succesiunea de inserare a numerelor 19, 36, 5, 21, 4, 26, 14 tabela arata ca in figura 6.

21	36	26	5	4	19	14
0	1	2	3	4	5	6

Figura 6: Rezultat - verificare dubla

6.2 Probleme aplicative

Ex. 5 — Fiind dat un text al unei limbi și o mulțime de cuvinte din limba respectivă, care sunt toate de aceeași dimensiune, determinați numărul de poziții candidat la care se pot găsi cuvintele textului. Pe prima linie a fișierului Date.in se găsește alfabetul în care vom căuta, iar pe a doua linie se găsesc cuvintele. În fișierul Date.out să se scrie numărul de poziții candidat la care apar cuvintele.

Date.in	Date.out
bbcabbabcb abba bbca abcb bbca aaaa	3

Explicatie: bbca se găsește pe poziția 0, abba se găsește pe poziția 3, abcb se găsește pe poziția 6. Celelalte cuvinte nu se găsesc în text. În total am găsit 3 poziții posibile.

Ex. 6 — Ștergeți elementele duplicate dintr-un array nesortat de dimensiune n . Pe prima linie a fișierului Date.in se găsește numărul n iar pe a doua linie se găsește secvența de numere. În fișierul Date.out să se scrie secvența fără duplicate.

Date.in	Date.out
9 1 2 10 6 2 5 3 1 15 6	1 2 10 6 5 3 15

Ex. 7 — Se citesc de la tastatura n cuvinte care contin doar litere mici. Sa se stocheze aceste cuvinte intr-o tabela de dispersie implementata cu tehnica de inlantuire. Functia de dispersie este compusa din urmatoarele:

- Cod de dispersie: $f_1(cuvant) = \text{suma codurilor ascii ale caracterelor din cuvant} (\text{suma codurilor ascii impartita la numarul de litere din cuvant}) \% n$.
- Functia de dispersie $f_2(a) = a \bmod m$ unde m este dimensiunea tabeli de dispersie.

Astfel indicele locatiei unde se va face inserarea sau cautarea in tabela a unui cuvant este $index = f_2(f_1(cuvant))$. Afisati continutul tabeli de dispersie pentru inserarea urmatoarelor $n = 10$ cuvinte:

mar, var, lup, mop, dop, urs, cuc, laborator, date, structura.

Folositi $m = 7$.