6 Tabele de dispersie

6.1 Obiective

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

6.2 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 continând toate zilele săptămânii. Tabelele dinamice pot fi organizate sub forma de liste liniare simplu înlănțuite, arbori de căutare sau tabele de dispersie. Dacă tabelele dinamice sunt organizate ca liste, căutările sunt efectuate liniar, lucru care încetinește operația de găsire a unui element. După cum am văzut în lucrarea precedentă, un arbore de căutare reduce timpul de găsire al unui element la un timp logaritmic. În acest laborator vom merge mai departe și vom încerca să construim o structură de date în care găsirea unui element se va efectua în timp constant O(1) - in medie, in cazul mediu statistic.

6.3 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

$$f : K \to H,$$

unde K reprezintă un set de chei iar H o mulțime de numere naturale. Functia f reprezintă o mapare *many to one*. Presupunând că avem două key, $k1$ și $k2$ cu $k1 \neq k2$ și rezultatul funcției de dispersie e același, de exemplu $f(k1) == f(k2)$, 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:

$$f(k) = \gamma(k) \text{ modulus } B,$$

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 ale 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), acumularea polinomiala, etc.

6.4 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 înlănţuire (chainning) și adresarea deschisă.

6.4.1 Tehnica înlănțuirii (chaining)

Metoda chaining presupune inserarea tuturor cheilor care au aceeși valoare de dispersie într-o listă simplu înlănțuită. Tabela va conține în fiecare poziţie o referință către o listă. Un exemplu de o astfel de tabelă este ilustrat in Fig. 6.1.

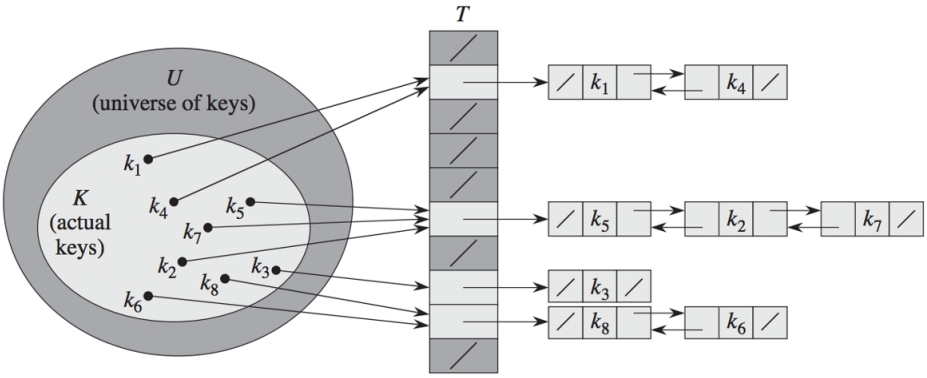

Figure 6.1: Tabele de dispersie: chaining

Operații:

- **Inserare** - Inițial toate sloturile tabelei 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 operatia 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 $h(key)$:
  $list\_delete\_key(T[h(x.key)], key)$

6.4.2 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 functie de dispersie care genereaza o permutare a spatiului de adrese astfel încat la căutarea sau inserarea unui element în tabela de dispersie se probează sloturile libere in functie de acea permutare. Exista urmatoarele tehnici de generare a secventelor de proba:

1. linear probing (Verificare liniara)
2. quadratic probing (Verificare patratica)
3. double hashing (Dispersie dubla)

Adresare deschisă - verificare liniară

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

$$h(k, i) = (h'(k) + i) mod\ m \tag{6.1}$$

unde

- $h'(k)$ este o functie de dispersie auxiliara
- $i = 0, 1, \ldots m-1$ este o variabila creste atata timp cat mai avem coliziuni.
- m este dimensiunea tabelei de dispersie

Fiind data o cheie k locatiile vor fi examinate in ordinea urmatoare: $T[h(k,0)]; T[h(k,1)] \ldots T[h(k,m-1)]$.

Presupunând că avem următorul șir de numere întregi: 18, 6, 36, 72, 43 și funcția de hashing $f = key$ modulo m ca în Fig. 6.2, unde $m = 8$ reprezintă dimensiunea tabelei, **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 tabelei, căutarea continuă cu începutul tabelei.
- Dacă nu s-a găsit niciun loc liber și s-a ajuns cu căutarea la locul coliziunii, atunci tabela este plină.

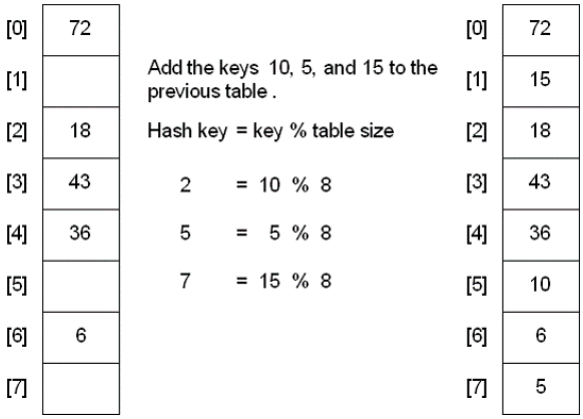

Figure 6.2: Tabele de dispersie: adresare directa

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

Problema care apare la verificarea liniara este data de faptul ca doua chei care sunt mapate initial la adrese diferite pot concura pentru aceleași locații în iteratii succesive. Acest fenomen este cunoscut sub numele de **clusterizare primara**. Astfel se formeaza șiruri lungi de locatii ocupate.

Adresare deschisă - verificare polinomială – quadratic probing

Verificarea polinomială folosește o funcție de dispersie de forma:

$$h(k,i) = (h'(k) + c_1 \times i + c_2 \times i^2) \; mod \; m \tag{6.2}$$

unde:

- $h'(k)$ este o functie de dispersie auxiliara
- $c_1! = 0$ si $c_2! = 0$ sunt constante auxiliare
- $i = 0, 1 \ldots m-1$ este o variabila care creste atata timp cat mai avem coliziuni.
- m este dimensiunea tabelei de dispersie

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

Și la această metodă apare problema de **clusterizare secundara** 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 iteratii succesive.

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:

$$h(k,i) = (h_1(k) + i \times h_2(k)) \; mod \; m \tag{6.3}$$

unde:

- $h_1(k)$ și $h_2(k)$ sunt funcții de dispersie auxiliare.
- $i = 0, 1 \ldots m-1$ este o variabila care creste atâta timp cat mai avem coliziuni.
- m este dimensiunea tabelei de dispersie

Astfel la o coliziune va fi probată locația $T[h_1(k)]$, urmată de pozitii incremetate cu $h_2(k) \; mod \; m$ si avem astfel un offset variabil.

6.5 Tabele de dispersie pentru diferite tipuri de date

In toate operatiile din exemplele de mai sus s-au folosit chei numerice cu valori mici. Se pune problema definirii unui comportament pentru cazul in 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 sa 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 siruri pentru care cheile reprezinta indici. Dar ce se întâmplă dacă indicii sunt foarte mari (de ordinul milioanelor). In acest caz o soluție elegantă pentru socarea perechilor cheie/valoare este data de tabele de dispersie.

In aceste cazuri functia de dispersie este compusa din 2 parti:

- **Cod de dispersie:** $f_1 : chei \rightarrow intregi$
- **Functie de compresie:** $f_2 : intregi \rightarrow [0, m-1]$, unde f_2 este o functie de genul celor utilizate la verificarea liniara, cuadratica sau dubla.

Indicele la care se va incerca inserarea / cautarea in tabela va fi dat de: $h(x) = h2(h1(x))$.

6.6 Mersul lucrarii

6.6.1 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.

- stergerea unui nod cu cheia key din tabelă

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

De exemplu pentru urmatoarele date:

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

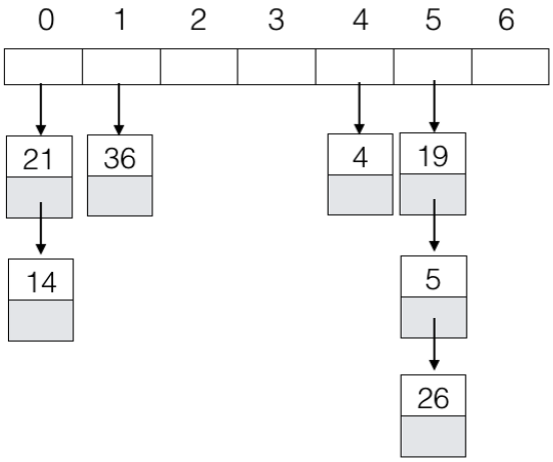

Figure 6.3: Exemplu Caining

tabela va arata ca in figura 6.3.

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
- $h_0(k) = k \ mod \ m$
- functia de dispersie: $h(k;i) = (h0(k) + i) \ mod \ m$
- dupa succesiunea de inserare a numerelor: 19, 36, 5, 21, 4, 26, 14 tabela arata ca in figura 6.4.

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

Figure 6.4: 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 **verificarea polinomiala** ca functie de dispersie. IMPLEMENTATI functia de dispersie, functia de inserare si functia de cautare a unei chei. Utilizati urmatoarele date:

- m = 7
- $h'(k) = k \ mod \ m$
- Functia de verificare polinomiala - functia de dispersie: $h(k,i) = (h'(k) + c_1 \times i + c_2 \times i^2) \ mod \ m$ 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 6.5.

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

Figure 6.5: 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: $h(k,i) = (h_1(k) + i \times h_2(k))\ mod\ m$
- $h_1(k) = k\ mod\ m$
- $h_2(k) = 5 - (k\ mod\ 5)$
- dupa succesiunea de inserare a numerelor 19, 36, 5, 21, 4, 26, 14 tabela arata ca in figura 6.6.

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

Figure 6.6: Rezultat - verificare dubla

6.6.2 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 găsii 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
bbcabbabcba 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: $f_1(cuvant)$ = suma codurilor ascii ale caracterelor din cuvant (suma codurilor ascii impartita la numarul de litere din cuvant) $\%n$.
- Functia de dispersie $f_2(a) = a \ mod \ m$ unde m este dimensiunea tabelei 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 tabelei de dispersie pentru inserarea urmatoarelor $n = 10$ cuvinte:
mar, var, lup, mop, dop, urs, cuc, $laborator$, $date$, $structura$.
Folositi $m = 7$.