

# TABELA DE DISPERSIE

## Hash Table

- Este o structură de date **eficientă** pentru implementarea dicționarelor (și nu numai).
- Compilator – **tabelă de simboluri** (cheia = șir de caractere corespunzătoare unui identificator)
- TD poate fi folosită pentru implementarea containerelor pe care operațiile specifice sunt: **adăugare** element, **căutare** element, **ștergere** element. Ex: dicționare, colecții, mulțimi
  - JAVA - HashMap, HashSet.
  - STL – unordered\_set, unordered\_map
- TD este o generalizare a noțiunii mai simple de **tabelă cu adresare directă**
- Notății
  - $n$  – numărul de elemente din container
  - un element  $e$  din container este o pereche cheie ( $c$ ) – valoare ( $v$ )  
( $TElement = TCheie \times TValoare$ )
  - $U$  – universul (domeniul) cheilor
  - $K$  – domeniul actual al cheilor (mulțimea cheilor efectiv memorate în container)

## Tabelă cu adresare directă

- Notății
  - chei numere naturale, chei distincte
  - $U = \{0, 1, 2, \dots, m-1\}$  -  $m$  relativ mic
  - $K$  – domeniul actual al cheilor (mulțimea cheilor efectiv memorate în container)
- Tabela cu adresare directă este memorată sub forma unui vector  $T[0..m-1]$ 
  - Locația  $T[c]$  va corespunde cheii  $c$
  - Dacă o cheie  $c \notin K$ , atunci  $T[c]$  va conține NIL (sau o valoare specială care marchează locație goală)
  - $T[c]$  poate memora un pointer spre elementul având cheia  $c$  sau elementul (cheia și valoarea asociată)

### **TElement**

$c$ :TCheie

$v$ :TValoare

### **TabelaAdresareDirecta**

$m$ :Întreg

$e$ :TElement[0..m-1]

### CAUTĂ ( $T, c$ )

//pre:  $T$  este o tabelă cu adresare directă,  $c$  este de tip TCheie  
@ returnează  $T.e[c]$

### ADAUGĂ ( $T, e$ )

//pre:  $T$  este o tabelă cu adresare directă,  $e$  este de tip TElement  
@  $T.e[e.c] \leftarrow e$

### ȘTERGE ( $T, c$ )

//pre:  $T$  este o tabelă cu adresare directă,  $c$  este de tip TCheie  
@  $T.e[e.c] \leftarrow \text{NIL}$

- Observații
  - funcționează bine dacă universul cheilor este mic
  - complexitatea timp a operațiilor este  $\phi(1)$
  - spațiul de memorare este  $\phi(|U|)$

## PROBLEMĂ

Sugerați cum se poate implementa o tabelă cu adresare directă în care cheile elementelor memorate nu sunt neapărat distincte și elementele pot avea date adiționale.

## Tabela de dispersie

- spațiul de memorare este  $\phi(|K|)$
- complexitatea timp *medie* pentru toate operațiile pe TD (adăugare, căutare, ștergere) este  $\phi(1)$ .
  - **căutarea** unui element într-o TD poate necesita  $\phi(n)$  în caz *defavorabil* (ca și căutarea în liste)
    - în practică, dispersia funcționează foarte bine
    - timpul *mediu* preconizat pentru căutare este  $\phi(1)$
- funcție de dispersie (*hash function*)  $d : U \rightarrow \{0, 1, \dots, m-1\}$ 
  - $d(c)$  este valoarea de dispersie a cheii  $c$
  - $c$  se dispersează în locația  $d(c)$
- **dispersia perfectă** (*perfect hashing, perfect hash function*)
  - fără coliziuni (când se cunosc cheile – ex. compilatoare)
- $d(c_1) = d(c_2)$  coliziune
- **adăugarea unui nou element**  $e=(c, v)$ 
  - se calculează  $i = d(c)$
  - dacă locația  $i$  este liberă, atunci se adaugă elementul

- dacă la locația  $i$  mai e memorat un alt element  $\Rightarrow$  **rezolvare coliziune**
- funcție de dispersie bună
  - este ușor de calculat (folosește operații aritmetice simple) -  $\phi(1)$
  - produce cât mai puține coliziuni.

### Interpretarea cheilor ca numere naturale

- Majoritatea funcțiilor de dispersie presupun universul cheilor din mulțimea numerelor naturale
- În cazul în care cheile nu sunt numere naturale, trebuie găsită o modalitate de a le interpreta ca numere naturale – o funcție care asociază fiecărei chei un număr natural (implementare -  $hashCode: TCheie \rightarrow \{0,1,2,\dots\}$ )
  - identificatorul **pt** poate fi interpretat ca un număr în baza **128** –  $(pt)_{128} = 112 \cdot 128 + 116 = 14452$ .
  - pentru un șir de caractere putem considera suma codurilor ASCII ale caracterelor.
- În cazul în care în container elementele sunt de tip *TElement* (nu au asociată o cheie - ex. mulțime, colecție),  $hashCode: TElement \rightarrow \{0,1,2,\dots\}$
- Pp. în cele ce urmează că avem chei naturale.

### Funcții de dispersie

- O funcție de dispersie bună satisface (aproximativ) *ipoteza dispersiei uniforme simple* (**Simple Uniform Hashing - SUH**): fiecare cheie se poate dispersa cu aceeași probabilitate în oricare din cele  $m$  locații.
  - $P(d(c) = j) = \frac{1}{m}, \forall j = 0, \dots, m-1 \quad \forall c \in U$
  - dacă această ipoteză este verificată, atunci se minimizează numărul de coliziuni
  - în practică se pot folosi tehnici euristice pentru a crea funcții de dispersie care să se comporte bine.

### I. Metoda diviziunii

- Dispersia prin diviziune
- $d(c) = c \bmod m$
- Experimental: valori bune pentru  $m$  sunt numerele prime nu prea apropiate de puteri exacte ale lui 2 (ex: 13,...)
- $m=13$ 
  - $c=63 \Rightarrow d(c)=11$
  - $c=26 \Rightarrow d(c)=0$

## II. Metoda înmulțirii

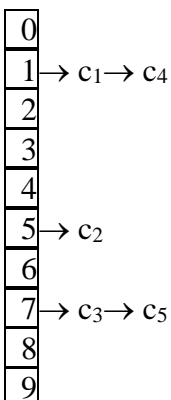
- $d(c) = [m \cdot (c \cdot A \bmod 1)]$  unde " $c \cdot A \bmod 1$ " reprezintă  $c \cdot A - [c \cdot A]$
- Valoarea lui  $m$  nu e critică (în general este o putere a lui 2)
- Knuth: valoarea optimă pentru  $A$  este  $\frac{\sqrt{5}-1}{2} \approx 0.6180339887$  (**golden ratio-1**)
  - $m = 13, A = 0.6180339887$  (Knuth)
  - $c=63 \Rightarrow d(c) = [13 \cdot \text{frac}(63 \cdot A)] = 12$
  - $c=52 \Rightarrow d(c) = [13 \cdot \text{frac}(52 \cdot A)] = 1$
  - $c=129 \Rightarrow d(c) = [13 \cdot \text{frac}(129 \cdot A)] = 9$

## III. Dispersia universală

- $c = \langle c_1, c_2, \dots, c_k \rangle$
- $d(c) = (\sum_{i=1}^k c_i \cdot x_i) \bmod m$  unde  $\langle x_1, x_2, \dots, x_k \rangle$  este o secvență de numere aleatoare fixate (selectate de-a lungul inițializării funcției de dispersie)
- Proprietate (foarte bună): oricare ar fi două chei distincte  $a$  și  $b$ , probabilitatea ca o funcție de dispersie aleatoare  $d$  să le mapeze în aceeași locație în tabela de dispersie este  $\frac{1}{m}$ .

## A. Rezolvare coliziuni prin liste independente (înlănțuire) – SEPARATE CHAINING

- Elementele care se dispersează în aceeași locație (sunt într-o coliziune), vor fi puse într-o listă înlănțuită
- Locația  $j$  conține un pointer către capul listei înlănțuite a elementelor care se dispersează în locația  $j$  (dacă această listă e vidă, se memorează NIL).
- Ex:  $m=10, U=\{c_1, \dots, c_{10}\}, K=\{c_1, c_2, c_3, c_4, c_5\}$  și  $d(c_1)=d(c_4)=1, d(c_2)=5, d(c_3)=d(c_5)=7$



## Reprezentare și operații

### **TElement**

$c$ :TCheie  
 $v$ :TValoare

### **Container**

$m$ :Întreg  
 $l$ :TListă[0.. $m$ -1]

- $d$  este funcția de dispersie
- pp. cheia are o singură valoare asociată
- Container poate fi, de ex., dicționar, mulțime, colecție.

### **CAUTĂ** ( $C, ch$ )

// pre:  $C$  este un container reprezentat sub forma unei TD (coliziuni prin înlanțuire),  $ch$  este de tip TCheie  
@ caută elementul cu cheia  $ch$  în lista  $C.l[d(ch)]$

### **ADAUGĂ** ( $C, e$ )

// pre:  $C$  este un container reprezentat sub forma unei TD (coliziuni prin înlanțuire),  $e$  este de tip TElement  
@ se adaugă elementul  $e$  în capul listei înlanțuite  $C.l[d(e.c)]$

### **ȘTERGE** ( $T, ch$ )

// pre:  $C$  este un container reprezentat sub forma unei TD (coliziuni prin înlanțuire),  $ch$  este de tip TCheie  
@ se șterge elementul cu cheia  $ch$  din lista înlanțuită  $C.l[d(e.c)]$

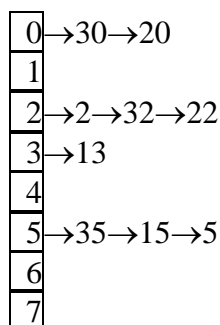
## Observații

- Este posibil ca listele independente să fie memorate ordonat după cheie sau valoare
- Funcția de dispersie este considerată *bună* dacă listele au aproximativ aceeași lungime
- Dacă apar multe liste de vide sau liste prea lungi  $\Rightarrow$  redispersare (**rehashing**)

## Exemplu

$m=10, d(c)=c \bmod m$

c	5	15	13	22	20	35	30	32	2
d(c)	5	5	3	2	0	5	0	2	2



## Iterator

- dacă listele înlănțuite ar fi reprezentate folosind alocare dinamică

### **Telement**

$c$ :TCheie  
 $v$ :TValoare

### **Nod**

$e$ :TElement  
 $urm$ : $\uparrow$ Nod

### **Container**

$m$ :Întreg  
 $l$ : $\uparrow$ Nod [0..m-1]

### **IteratorContainer**

$c$ :Container //referință (în implementare)  
 $pozCurent$ :Întreg  
 $curent$ : $\uparrow$ Nod

## Timp defavorabil pentru operații

Pp  $n$  este numărul elementelor din container.

- **CAUTĂ** –  $O(n)$
- **ADAUGĂ** -  $\phi(1)$
- **ȘTERGE** – presupune (1) căutare nod în lista înlănțuită + (2) ștergere nod  $O(n)$

## Analiza dispersiei cu înlănțuire

### **Notății și presupuneri**

- $\alpha = \frac{n}{m}$  *factorul de încărcare* al tabeli (numărul mediu de elemente memorate într-o înlănțuire)
- Pp. că timpul de calcul al funcției de dispersie este  $\theta(1)$  (!! la timpul de căutare se adaugă și timpul de calcul al funcției de dispersie)

– La **căutare** apar 2 cazuri

- Căutare **cu succes**
- Căutare **fără succes**

**Teorema 1.** Într-o TD în care coliziunile sunt rezolvate prin înlănțuire, în *ipoteza dispersiei uniforme simple* (SUH), o căutare **fără succes**, necesită, în *medie*, un timp  $\theta(1 + \alpha)$ .

**Teorema 2.** Într-o TD în care coliziunile sunt rezolvate prin înlănțuire, în *ipoteza dispersiei uniforme simple* (SUH), o căutare **cu succes**, necesită, în *medie*, un timp  $\theta(1 + \alpha)$ .

## CONCLUZII

- Dacă  $n = O(m) \Rightarrow \alpha = \frac{O(m)}{m} = O(1) \Rightarrow$  **căutarea** necesită, în *medie*, timp constant  $\theta(1)$
- Adăugarea necesită  $\theta(1)$
- Dacă listele sunt dublu înlănțuite atunci ștergerea unui nod se poate face în  $\theta(1)$

$\Rightarrow$  **TOATE OPERAȚIILE (adăugare, căutare, ștergere) POT FI EXECUTATE ÎN *MEDIE* ÎN  $\theta(1)$**

## PROBLEME

1. Presupunem că folosim o funcție de dispersie aleatoare **d** pentru a dispersa  $n$  chei distincte într-o tabelă  $T$  de dimensiune  $m$ . Care este numărul mediu de coliziuni? (cardinalul probabil al mulțimii  $\{(x, y) \in T_{Cheie} \times T_{Cheie} : d(x) = d(y)\}$ )
2. Presupunem că folosim o TD în care coliziunile sunt rezolvate prin înlănțuire (liste independente), dar fiecare listă este ordonată după cheie. Care va fi timpul de execuție pentru **căutare** (cu succes, fără succes), **adăugare** și **ștergere**?
3. Arătați că dacă  $|U| > n \cdot m$ , atunci există o submulțime a lui  $U$  de mărime  $n$  ce conține chei care se dispersează toate în aceeași locație, astfel încât timpul de căutare pentru dispersia cu înlănțuire, în cazul cel mai defavorabil, este  $\phi(n)$ .