

## TABELA DE DISPERSIE

- continuare -

### **B. Rezolvare coliziuni prin liste întrepătrunse (întrepătrunderea listelor) – COALESCED CHAINING**

- Toate listele înlănțuite (care memorează coliziuni) se memorează în tabelă, nu sunt liste în afara tablei (vezi lista înlănțuită cu înlănțuiri reprezentate pe tablou)
- Nu se folosesc pointeri pentru memorarea înlănțuirilor
- Factorul de încărcare este subunitar  $\alpha < 1$ , altfel tabela este plină
- Gestiunea spațiului liber în tabelă poate fi făcută ca la lista înlănțuită cu înlănțuiri reprezentate pe tablou (folosind o listă înlănțuită a spațiului liber)
- Dezavantaj: tabela se poate umple ( $\alpha = 1$ ). Soluție: se crește  $m$ , ceea ce presupune redispersarea elementelor.
- Experimental: funcția de dispersie se consideră bună dacă spațiul de memorie e ocupat mai puțin de 75% ( $\alpha < 0.75$ )

**Teoremă.** Într-o TD în care coliziunile sunt rezolvate prin liste întrepătrunse, în ipoteza dispersiei uniforme simple (SUH), o **TOATE** operațiile (**adăugare, căutare, ștergere**), necesită, în *medie*, un timp  $\theta(1)$ .

*Donald E. Knuth, The Art of Computer Programming, Second edition, University of Stanford, 1998*

- Timpul mediu pentru **căutare fără succes**  $T(\alpha) \approx 1 + \frac{1}{4}(e^{2\alpha} - 1 - 2\alpha)$
- Timpul mediu pentru **căutare cu succes**  $T(\alpha) \approx 1 + \frac{1}{8\alpha}(e^{2\alpha} - 1 - 2\alpha) + \frac{\alpha}{4}$

### **EXEMPLU**

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

<b>C</b>	5	15	13	22	20	35	30	32	2
<b>d(c)</b>	5	5	3	2	0	5	0	2	2

### **Reprezentare**

Presupuneri:

- Se memorează doar cheile.
- Chei distincte.
- Dacă o locație nu are legătură spre o altă locație din tabelă, se memorează **-1** în câmpul *urm*.
- Chei naturale (se memorează **-1** dacă locația e liberă)
- Spațiul liber e gestionat secvențial (de la stânga la dreapta) – sau de la dreapta la stânga

## Container

$m$ : Intreg //capacitatea tabelii  
 $ch$ : TCheie[] //cheile  
 $urm$ : Intreg[] //legaturile  
 $primLiber$ : Intreg //prima locatie libera

Funcția de dispersie este  $d: T_{Cheie} \rightarrow \{0, 1, \dots, m-1\}$

## ADĂUGARE

Indice	0	1	2	3	4	5	6	7	8	9
Cheie	15	20	22	13	35	5	30	32	2	
Următor	1	4	7	-1	6	0	-1	8	0	-1

### subalgoritm $actPrimLiber(c)$ este

//se actualizează  $primLiber$  după ce locația a fost ocupată

//operația nu este în interfața containerului

$c.primLiber \leftarrow c.primLiber + 1$

**cât timp**  $(c.primLiber \leq c.m-1)$  și  $(c.ch[c.primLiber] \neq -1)$  **execută**

$c.primLiber \leftarrow c.primLiber + 1$

**sfcât timp**

**sfactPrimLiber**

- algoritmul de adăugare la sfârșitul listei înlănțuite (în caz de coliziune) – **LISCH** (*Late Insertion Standard Coalesced Hashing*)

### subalgoritm $adaugă(c, ch)$ este

//pre:  $c$  e containerul,  $ch$  cheia care se adaugă

$i \leftarrow c.d[ch]$

**dacă**  $c.ch[i] = -1$  **atunci** //locația e liberă, memorăm

$c.ch[i] \leftarrow ch$

**dacă**  $i = c.primLiber$  **atunci**

$actPrimLiber(c)$

**sfdacă**

**altfel**

//adăugăm la finalul listei înlănțuite care este memorată de la locația  $i$

// dacă mai găsim cheia, ne oprim

**cât timp**  $(i \neq -1)$  și  $(c.ch[i] \neq ch)$  **execută**

$j \leftarrow i$

$i \leftarrow c.urm[i]$

**sfcât timp**

**dacă**  $i \neq -1$  **atunci** //am mai găsit cheia

@ cheia existentă

**altfel**

**dacă**  $c.primLiber \leq c.m-1$  **atunci** //tabela nu este plină

$c.ch[c.primLiber] \leftarrow ch$

$c.urm[j] \leftarrow c.primLiber$   
**actPrimLiber**(c)  
**altfel**  
 @ depășire tabelă  
**sfdacă**  
**sfdacă**  
**sfdacă**  
**sfadaugă**

## CĂUTARE

- pp. că vrem să căutăm cheia  $ch$
- o căutăm în lista înlănțuită care pornește de la locația de dispersie a cheii  $ch$ , adică  $d(ch)$
- dacă găsim cheia în lista înlănțuită  $\Rightarrow$  **căutare cu succes**, altfel **căutare fără succes**
- exemplu
  - o căutăm **35 (cu succes)**:  $5 \rightarrow 15 \rightarrow 20 \rightarrow 35$
  - o căutăm **45 (fără succes)**:  $5 \rightarrow 15 \rightarrow 20 \rightarrow 35 \rightarrow 30 \rightarrow -1$

## STERGERE

- pp. că vrem să ștergem cheia  $ch$
- localizăm cheia
- exemplu
  - o  $ch=5$   
 $(5,5) \rightarrow (0,15) \rightarrow (1,20) \rightarrow (4,35) \rightarrow (6,30)$
- tabela rezultată în urma ștergerii

Indice	0	1	2	3	4	5	6	7	8	9
Cheie	20	-1	22	13	35	15	30	32	2	
Următor	4	-1	7	-1	6	0	-1	8	0	-1

## Variante pentru îmbunătățirea performanței (reducerea numărului de locații verificate în caz de coliziune)

- organizarea tabelii - zonă separată (**primary area**) pentru elementele care nu sunt în coliziune și zonă separată pentru elementele care sunt în coliziuni (**overflow area**)
  - o **Address Factor** =  $|primary\ area| / dimensiunea\ tabelii$
  - o o valoare de aprox. 0.86 pentru **Address Factor** conduce la o performanță aproape optimală pentru **căutare** pentru majoritatea valorilor lui  $\alpha$ .
- modalitatea de înlănțuire a elementelor dintr-o coliziune
- modalitatea de selectare a spațiului liber (**primLiber**)
  - o de la dreapta spre stânga (de la finalul tabelii)
  - o alegerea aleatoare a spațiului liber (doar 1% îmbunătățire) – alg. de adăugare **REISCH** (R - random)
  - o alternarea selecției spațiului liber între începutul și finalul tabelii - **BLISCH** (B – bidirectional)
  - o experimental: pentru  $\alpha > 0.2$  **LISCH** e mai performant

## IMPLEMENTAREA OPERAȚIILOR DE CĂUTARE ȘI STERGERE LA SEMINARUL 6