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 tabelei (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* mod *m*

C	5	15	13	22	20	35	30	32	2
d(c)	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 (se la stânga la dreapta)

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

Funcția de dispersie este d:TCheie$\rightarrow\{0,1\ldots,m\text{-}1\}$

ADĂUGARE

primLiber=9

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←-*c.primLiber*+1
câttimp (*c.primLiber*≤ *c.m*-1) și (*c.ch***[***c.primLiber***]** ≠-1) **execută**
c.primLiber←-*c.primLiber*+1
sfcâttimp
sf*actPrimLiber*

subalgoritm *adaugă*(*c, ch*) este
//pre: *c* e containerul, *ch* cheia care se adaugă
i←*c.d***[***ch***]**
dacă *c.ch***[**i**]**=-1 **atunci** //locația e liberă, memorăm
*c.ch***[**i**]** ←*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
câttimp (i≠-1) și (*c.ch***[**i**]** ≠*ch*) **execută**
j←i
i←*c.urm***[**i**]**
sfcâttimp
dacă i≠-1 **atunci** //am mai găsit cheia
@ cheie existentă
altfel
dacă *c.primLiber* ≤ *c.m*-1 **atunci** //tabela nu este plină
*c.ch***[***c.primLiber***]** ←*ch*

```
                    c.urm[j] ←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 înlănțuiă ⇒ **căutare cu succes**, altfel **căutare fără succes**
- exemplu
  - căutăm **35 (cu succes)**: 5→15→20→**35**
  - căutăm **45 (fără succes)**: 5→15→20→35→ 30→ **-1**

ȘTERGERE

- pp. că vrem să ștergem cheia *ch*
- localizăm cheia
- exemplu
  - *ch*=5

    (5,**5**) →(0,**15**) →(1,**20**) →(4,**35**) →(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

IMPLEMENTAREA OPERAȚIILOR DE CĂUTARE ŞI ȘTERGERE LA SEMINARUL 6