SDA - Seminar 6 - TD

Conținut:

- Iterator DO reprezentat sub formă de TD, cu rezolvare coliziuni prin liste independente
- Dicționar reprezentat sub formă de TD, cu rezolvare coliziuni prin liste întrepătrunse

1. **Iterator pe Dicționar Ordonat (DO) reprezentat sub formă de tabelă de dispersie (TD), cu rezolvare coliziuni prin liste independente.**

- Presupunem că:
    - Memorăm doar cheile
    - Avem chei numere naturale
    - Ordinea impusă cheilor este “<” (cheile fiind distincte)

Exemplu:

- Presupunem că dorim să adăugăm în dicționar următoarele 9 chei: 5, 28, 19, 15, 20, 33, 12, 17, 10
- , iar dicționarul este reprezentat pe o TD cu
    - m = 9 locații

    și optăm pentru
    - dispersia prin diviziune

În cazul dispersiei prin diviziune, funcția de dispersie d este definită astfel:

✓ d(c) = c mod m

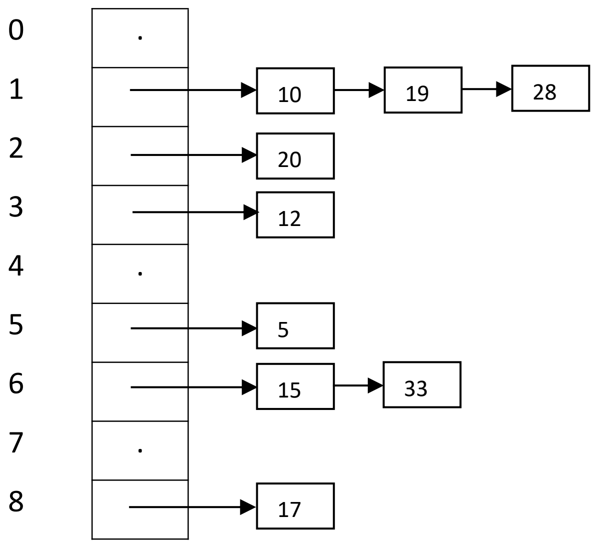
- Valorile de dispersie pentru cele 9 chei care se doresc a fi adăugate în DO:

c	5	28	19	15	20	33	12	17	10
d(c)	5	1	1	6	2	6	3	8	1

- **Observație**: Secvența d(c) formată de valorile de dispersie conține valori care se repetă. Acestea corespund coliziunilor. Având în vedere că discutăm în contextul unei TD cu rezolvare coliziuni prin liste independente, cheile aparținând unei coliziuni vor fi memorate în aceeași listă (independent de cheile având alte valori de dispersie).

- TD cu ajutorul căreia se reprezintă Dicționarul Ordonat, ulterior inserării cheilor exemplificate (5, 28, 19, 15, 20, 33, 12, 17, 10):

TD:



Observație: Cheile se inserează în listele independente astfel încât să se respecte relația impusă Dicționarului Ordonat. Cu alte cuvinte, toate listele independente vor fi (independent) ordonate.

Ordinea de parcurgere a cheilor din DO care se așteaptă a fi oferită de Iterator:

✓ Elementele ar trebui să fie parcurse de Iterator în această ordine: 5, 10, 12, 15, 17, 19, 20, 28, 33, adică în ordine crescătoare.

 Cum se obține această secvență ordonată a cheilor pornind de la TD ilustrată anterior?

✓ Se observă că lista ordonată a cheilor se obține prin interclasarea tuturor listelor independente (cu ajutorul cărora se memorează cheile care intră în coliziune).
Așadar, **implementarea Iteratorului pe DO se poate reduce la interclasarea prealabilă** (în constructor) **a listelor independente, într-o nouă listă,** ordonată, **urmată de parcurgerea ei .**

- **Reprezentare:**

<u>NodT:</u>
c: TCheie *(presupunând că memorăm doar cheile; altfel, adăugăm v:TValoare)*
urm: ↑NodT

<u>DicționarOrdonat:</u>
m: Intreg *(dimensiunea tabelei)*
l : (↑NodT)[] *(vectorul de liste independente)*
d: TFuncție: TCheie -> {0,1,…,m-1} *(funcția de dispersie)*
R: Relație: TCheie x TCheie -> {A, F} *(relația de ordine impusă cheilor)*

<u>IteratorDicționar:</u>
d: DicționarOrdonat
l: ↑NodT *(pointer la primul nod al listei ordonate obținută prin interclasare)* (sau, alternativ, TListă)
curentNod: ↑NodT *(cursorul iteratorului = pointer la nodul curent din lista ob)*

Constructorul Iteratorului:

```
subalgoritm creeaza(it, d):
     it.d ← d
     interclaseazaListe (d, it.l)
     it.curentNod ← it.l (sau it.l.prim, depinzând de tipul ales,
al rezultatului operației interclasează)
sf_subalgoritm
```

- ✓ Operațiile *valid, următor, element* vor avea complexitate constantă: Θ(1).

- ✓ *interclaseazăListe* poate interclasa listele independente ordonate :

1. pe rând, adică interclasând prima listă cu a 2-a, după care lista rezultată din interclasarea primelor două liste cu cea de-a 3-a ș.a.m.d.

2. <u>Concomitent</u>, adică interclasând toate listele deodată <u>folosind un ansamblu</u>

- ✓ Complexitatea interclasării:

$$\left.\begin{matrix} \text{TD cu m poziții} \\ \text{Dicționar cu n elemente} \end{matrix}\right\} \Rightarrow \text{ nr mediu de elemente într} - \text{o listă: } \frac{\text{n}}{\text{m}} = \alpha \text{ (factor de încărcare)}$$

1. <u>Interclasare prima lista cu a 2-a, etc.:</u>

- lista1 + lista2 => lista12 ----> α + α = 2α (operații elementare)
- lista12 + lista3 => lista123 ----> 2α + α = 3α
- lista123 + lista4 => lista1234 ----> 3 α + α = 4 α
- …

=> Se obțin, în total : $2\alpha + 3\alpha + \ldots + m\alpha \approx \left.\begin{array}{r} \frac{m*(m+1)}{2}\alpha \\ \alpha = \frac{n}{m} \end{array}\right\} \rightarrow \frac{m\,(m+1)}{2}\ \frac{n}{m} \Rightarrow\ \in\ \theta\,(n*m) \approx\ \theta(n)$

(presupunând m – constant)

2. <u>Toate listele deodată, folosind un ansamblu:</u>

- Punem (pointerul la) primul nod din fiecare listă într-un ansamblu minimal (*min-heap*).
- Scoatem (pointerul la) nodul conținând cheia minimă și adăugăm în ansamblu (pointerul la) următorul nod al listei căreia îi aparține nodul scos (dacă există)
- Ansamblul va conține maxim *k* elemente în orice moment, *k* fiind numărul de liste, 1 ≤ *k* ≤ m => înălțimea ansamblului e $O(\log_2 k)$

Observație: Ansamblul va conține k elemente atâta timp cât mai sunt elemente de parcurs în fiecare dintre cele k liste care se interclasează. Pe măsură ce listele se epuizează (parcurgerea lor, în cadrul interclasării, se finalizează), numărul de elemente memorate în ansamblu scade.

Așadar, deducem complexitatea interclasării ca fiind:

- $O(n \log_2 m)$, dacă k > 1
- Θ(n), dacă k = 1 (interclasarea se reduce la copierea singurei liste)

Observăm că valoarea lui k este mai mică sau egală cu m (avem cel mult atâtea liste independente câte locații în TD) => $\log_2$ k e constant, deci obținem o complexitate liniară pentru interclasare.

2. **Dicționar reprezentat pe TD, în varianta de rezolvare a coliziunilor prin liste întrepătrunse**

- Presupunem, din nou, că:
    - Memorăm doar cheile (doar pentru ilustrarea exemplului)
    - Avem chei numere naturale

Exemplu:

- Presupunem că dorim să adăugăm în dicționar următoarele 7 chei: 5, 18, 16, 15, 13, 31, 26
- , iar dicționarul este reprezentat pe o TD cu
    - m = 13 locații

    și optăm pentru
    - dispersia prin diviziune

Valorile de dispersie pentru cele 7 chei care se doresc a fi adăugate în Dicționar?

c	5	18	16	15	13	31	26
d(c)	5	5	3	2	0	5	0

➢ Adăugăm cheia 5

- Valoarea de dispersie pentru 5 este 5
- Slotul având indexul 5 este liber
- Adăugăm cheia 5 în slotul având indexul 5
- primLiber = 0 (rămâne neschimbat)

	0	1	2	3	4	5	6	7	8	9	10	11	12
c						5							
urm	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

➢ Adăugăm cheia 18

- Valoarea de dispersie pentru 18 este 5
- Slotul având indexul 5 este ocupat
- Prin urmare, adăugăm 18 pe slotul având indexul primLiber = 0
- primLiber = ~~0~~1 (actualizăm valoarea primului slot liber)
- Având în vedere că 18 se dispersează pe indexul 5, începând de la indexul 5 trebuie să ajungem, urmând legăturile, la cheia 8 (deci la indexul 0). Cu alte cuvinte, adăugăm cheia

18 (de pe indexul 0) la finalul înlănțuirii începând de pe indexul 5 (egal cu valoarea ei de dispersie). Aceasta se rezumă la urm[5]=0.

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18					5							
urm	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1

- Adăugăm cheia 16

  - Valoarea de dispersie pentru 16 este 3
  - Slotul având indexul 3 este liber
  - Adăugăm cheia 16 în slotul având indexul 3
  - primLiber = ~~0~~1 (rămâne neschimbat)

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18			16		5							
urm	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1

- Adăugăm cheia 15

  - Valoarea de dispersie pentru 15 este 2
  - Slotul având indexul 2 este liber
  - Adăugăm cheia 15 în slotul având indexul 2
  - primLiber = ~~0~~1 (rămâne neschimbat)

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18		15	16		5							
urm	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1

- Adăugăm cheia 13

  - Valoarea de dispersie pentru 13 este 0
  - Slotul având indexul 0 este ocupat
  - Prin urmare, adăugăm 13 pe slotul având indexul primLiber = 1
  - primLiber = ~~01~~4 (cel mai mic index mai mare decât 1, al unui slot liber)
  - Având în vedere că 13 se dispersează pe indexul 0, începând de la indexul 0 trebuie să ajungem, urmând legăturile, la cheia 13 (deci la indexul 1). Cu alte cuvinte, adăugăm cheia 13 (de pe indexul 1) la finalul înlănțuirii începând de pe indexul 0 (egal cu valoarea ei de dispersie). Aceasta se rezumă la urm[0]=1.

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18	13	15	16		5							
urm	~~-1~~ 1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1

- Adăugăm cheia 31
  - Valoarea de dispersie pentru 31 este 5
  - Slotul având indexul 5 este ocupat
  - Prin urmare, adăugăm 31 pe slotul având indexul primLiber = 4
  - primLiber = ~~014~~6 (cel mai mic index mai mare decât 4, al unui slot liber)
  - Având în vedere că 31 se dispersează pe indexul 5, începând de la indexul 5 trebuie să ajungem, urmând legăturile, la cheia 31 (deci la indexul 4). Cu alte cuvinte, adăugăm cheia 31 (de pe indexul 4) la finalul înlănțuirii începând de pe indexul 5 (egal cu valoarea ei de dispersie). Având în vedere că urm[5]=0, urm[0] = 1 și abia urm[1]=-1, aceasta se rezumă la urm[1]=4.

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18	13	15	16	31	5							
urm	~~-1~~ 1	~~-1~~ 4	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1

- Adăugăm cheia 26
  - Valoarea de dispersie pentru 26 este 0
  - Slotul având indexul 0 este ocupat
  - Prin urmare, adăugăm 26 pe slotul având indexul primLiber = 6
  - primLiber = ~~0146~~7 (cel mai mic index mai mare decât 6, al unui slot liber)
  - Având în vedere că 26 se dispersează pe indexul 0, începând de la indexul 0 trebuie să ajungem, urmând legăturile, la cheia 26 (deci la indexul 6). Cu alte cuvinte, adăugăm cheia 26 (de pe indexul 6) la finalul înlănțuirii începând de pe indexul 0 (egal cu valoarea ei de dispersie). Având în vedere că urm[0]=1, urm[1] = 4 și abia urm[4]=-1, aceasta se rezumă la urm[4]=6.

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18	13	15	16	31	5	26						
urm	~~-1~~ 1	~~-1~~ 4	-1	-1	~~-1~~ 6	~~-1~~ 0	-1	-1	-1	-1	-1	-1	-1

Observații:

- primLiber se actualizează considerând pozițiile libere de la stânga la dreapta, ceea ce înseamnă că spațiul liber nu este înlănțuit, ci se gestionează secvențial

- Într-o înlănțuire putem avea elemente care aparțin unor coliziuni diferite. De exemplu, coliziunea care începe pe poziția 5: 5 (valoare dispersie: 5) –> 18 (valoare dispersie: 5) –> 13 (valoare dispersie: 0) –> 31 (valoare dispersie: 5) –> 26 (valoare dispersie: 0) conține chei care se dispersează atât pe poziția 5, cât și pe poziția 0.

Reprezentarea TAD Dicționar, considerând TD în varianta de rezolvare a coliziunilor prin liste întrepătrunse?

✓ **Reprezentare Dicționare:**

TElement:
c: TCheie
v: TValoare

Dicționar:
m: Întreg (dimensiunea tabelei)
e: (TElement)[]
urm:{-1, 0,…, m-1}[]
primLiber: Întreg {0,...,m-1,m}
d: TFuncție: TCheie -> {0,…, m-1}

- **Observație:** În stabilirea domeniului valorilor pentru primLiber, s-a presupus că primLiber=m indică faptul că TD este plină.

```
subalgoritm creează (d):
     @ se inițiază funcție de dispersie d
     @ se inițiază m
     pentru i ← 0, m-1 execută
          d.e[i] ← NULL_TElement
          d.urm[i] ← -1
     sf_pentru
     d.primLiber ← 0
sf_subalgoritm
```

✓ Complexitate: Θ(m), m reprezentând dimensiunea tabelei

```
Funcția caută(d, c):
     i ← d.d(c) //Calculăm valoarea de dispersie pentru cheia c
     câttimp (i ≠ -1 și d.e[i].c ≠ c) execută //Cât timp mai sunt elemente în înlănțuirea începând de la indexul egal cu valoarea de dispersie și nu am găsit cheia căutată
          i ← d.urm[i] //ne deplasăm la următorul element din înlănțuire, urmând legăturile
     sf_câttimp
     dacă i = -1 atunci //în cazul în care cheia nu a fost găsită
          caută ← NULL_TValoare //semnalăm aceasta returnând NULL_TValoare
```

```
    altfel
        caută ← d.e[i].v //altfel, returnăm valoarea asociată ei
sf_funcție
```

✓ Complexitate: O(m), dar în medie (în ipoteza dispersiei uniforme simple) Θ(1)

Observație: Implementarea operației de adăugare este descrisă în materialul aferent cursului 10, prin urmare continuăm cu operația de **ștergere**.

Exemplificăm operația de ștergere pentru cheia 5.

Pornim de la TD în această stare:

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18	13	15	16	31	5	26						
urm	1	4	-1	-1	6	0	-1	-1	-1	-1	-1	-1	-1

, indexul primului slot liber fiind primLiber=7.

Observație:

- Nu putem elibera slotul 5 marcând, totodată, urm[5] cu -1 deoarece am pierde legăturile înspre 18 și 31, nemaiputându-le găsi prin operația *caută.*

Ideea ștergerii (pentru exemplul considerat; descrierea generică este prezentată în materialul aferent cursului 10):

- Căutăm elemente, în înlănțuirea începând de la poziția de pe care ștergem (în cazul nostru, 5) care se dispersează pe această poziție (adică pe poziția 5).
    a. Dacă nu există astfel de elemente, ștergem elementul precum am șterge un nod dintr-o listă simplu înlănțuită, refăcând legăturile
    b. Dacă există un astfel de element, atunci mutăm elementul pe poziția de unde dorim să ștergem și repetăm procesul de ștergere pentru poziția de unde am mutat elementul.

Așadar:

- Dorim să ștergem cheia 5, care este pe poziția 5

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18	13	15	16	31	5	26						
urm	1	4	-1	-1	6	0	-1	-1	-1	-1	-1	-1	-1

- Căutăm primul element care se dispersează pe poziția 5. Din 5, urmând legătura (urm[5]=0), ajungem la cheia 18, care se dispersează întocmai pe poziția 5.
- Așadar, mutăm (copiem) 18 pe poziția 5

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	18	13	15	16	31	~~5~~ 18	26						
urm	1	4	-1	-1	6	0	-1	-1	-1	-1	-1	-1	-1

- În continuare, dorim să ștergem cheia 18 de pe poziția 0 (de unde am identificat-o, aducând-o pe poziția 5)
- Căutăm primul element care se dispersează pe poziția 0. Din 0, urmând legătura (urm[0]=1), ajungem la cheia 13, care se dispersează întocmai pe poziția 0.
- Așadar, mutăm (copiem) 13 pe poziția 0

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	~~18~~ 13	13	15	16	31	~~5~~ 18	26						
urm	1	4	-1	-1	6	0	-1	-1	-1	-1	-1	-1	-1

- În continuare, dorim să ștergem cheia 13 de pe poziția 1 (de unde am identificat-o, aducând-o pe poziția 0)
- Căutăm primul element care se dispersează pe poziția 1 (adică pe poziția de unde dorim să ștergem). Din 1, urmând legătura (urm[1]=4), ajungem la cheia 31 (de pe poziția 4), care nu se dispersează pe poziția 1. Din 4, urmând legătura (urm[4]=6), ajungem la cheia 26 (de pe poziția 6), care, din nou, nu se dispersează pe poziția 1. Cheia 26 este, însă, ultima cheie din înlănțuire, având în vedere că urm[6]=-1. Prin urmare, deducem că nu există o cheie care se dispersează pe poziția 1. Așadar, ștergem poziția 1 din înlănțuire (precum am șterge dintr-o LSI), modificând legăturile (urm[0]<--urm[1] = 4)

	0	1	2	3	4	5	6	7	8	9	10	11	12
c	~~18~~ 13		15	16	31	~~5~~ 18	26						
urm	~~1~~ 4	~~4~~ -1	-1	-1	6	0	-1	-1	-1	-1	-1	-1	-1

Având în vedere că am eliberat un index mai mic decât primLiber=7, primLiber devine 1 (primLiber=~~7~~1).

```
subalgoritm sterge(d, c) este

    i ← d.d(c)//calculăm valoarea de dispersie pentru cheia c
    j ← -1 //rețin precedentul lui i, întrucât, când ștergem, ne este necesar "nodul" anterior, pentru refacerea legăturii
    //în următoarea structură repetitivă, parcurgem tabela pentru a verifica dacă poziția i are vreun anterior
    k ← 0
    câttimp (k < d.m și j = -1) execută
        dacă d.urm[k] = i atunci
            j ← k
        altfel
            k ← k+1
    sf_câttimp
    //în următoarea structură repetitivă, localizăm cheia care trebuie ștearsă, actualizând și precedentul
    câttimp i ≠ -1 și d.e[i].c ≠ c execută
        j ← i
        i ← d.urm[i]
    sf_câttimp
    //verificăm existența cheii
    dacă i= -1 atunci
        @cheia nu există
    altfel //în caz afirmativ,
        //căutăm, în înlănțuirea începând de pe poziția i o primă altă cheie care se dispersează în i (poziția de șters)
        gata ← fals //folosim o variabila booleană care devine adevărat când nu se dispersează nimic în i  (poziția de șters)
        repetă
            p ← d.urm[i] //prima poziție verificată
            pp ← i //anteriorul poziției p
            câttimp p ≠ -1 și d.d(d.e[p].c) ≠ i execută //cât timp mai avem chei în înlănțuire și nu am găsit o cheie care se dispersează pe poziția i, avansăm în înlănțuire, urmând legăturile, cu poziția curentă și cu cea anterioară ei
                pp ← p
                p ← d.urm[p]
            sf_câttimp
            dacă p = -1 atunci
                gata ← adevarat //marcăm situația în care nu există o cheie care se dispersează în i (poziția de șters)
            altfel //altfel
                d.e[i] ← d.e[p] //aducem cheia (perechea) de pe poziția p pe poziția i
                j ← pp //poziția anterioară noii poziții de șters devine pp
                i ← p //iar noua poziție de șters devine p (poziția de pe care am adus cheia pe fosta poziție de șters)
            sf_dacă
        până_când gata
```

```
        //nemaifiind chei care se dispersează pe poziția de șters, ștergem această poziție din înlănțuire
        dacă j ≠ -1 atunci //în cazul în care are un precedent, refacem legătura de la poziția precedentă (în înlănțuire) la următoarea poziție  (în înlănțuire)
            d.urm[j] ← d.urm[i]
        sf_dacă
        //și marcăm faptul că poziția i a fost eliberată
        d.e[i] ← NULL_TElement
        d.urm[i] ← -1
        //în cazul în care am eliberat o poziție mai mică decât primLiber, actualizăm valoarea primei poziții libere, dat fiind că gestionăm spațiul liber de la stânga la dreapta.
        dacă d.primLiber > i atunci
            d.primLiber ← i
        sf_dacă
    sf_dacă
sf_subalgoritm
```