**UNIVERSITATEA BABEŞ-BOLYAI**

**PROIECT STRUCTURI DE DATE ŞI ALGORITMI**

**TAD Mulţime – implementare folosind o tabelă de dispersie / rezolvare coliziuni prin adresare deschisă**

Student: Deneş Gabriela-Damaris Profesor îndrumător:

Dr. Gabriela-Czibula

Grupa: 211

1. **Enunţul problemei**

La concursul anual de programare, comitetul de evaluare înregistrează într-o listă scorul

obţinut de fiecare participant. Despre fiecare participant se înregistrează următoarele

informaţii: Nume complet, Scor. Fiecare participant trebuie să rezolve 10 probleme, maximul

de puncte pe fiecare problemă este 10.

Creaţi o aplicaţie care permite comitetului de organizare să:

a) Adauge participanţi;

b) Şteargă participanţi;

c) Afişeze participanţii care au un scor mai mare decât o valoare dată.

1. **Specificarea şi interfaţa Tipului Abstract de Date**
   1. **TAD Iterator Mulţime**

**Domeniu**

I = {i|I – iterator pe m ∈ M}, mulţimea tuturor iteratorilor pe mulţimi

**Operaţii (**interfaţa TAD-ului Iterator**)**

Următor (i)

**Pre**: i ∈ I, i iterator pe m

**Post**: m′ ∈ M,m′ = m ∪ {e}

m-mulţimea elementelor iterate

e- un element neiterat

Valid (i)

**Pre**: i ∈ I, i iterator pe m

**Post:** →{Adevărat, dacă i se referă la un element valid din m

Fals, altfel

Element (i, e)

Pre: i ∈ I, i iterator pe m, valid(i)

**Post:** e = elementul curent referit de i

* 1. **TAD Mulţime**

**Domeniu**

M = {m|m = {ei|i = 1, n, n ∈ N, ei de tip TElement }}, mulțimea tuturor

mulțimilor cu elemente de tip TElement. TElement are o operație hash, care

reprezintă o funcția de dispersie.

**Operaţii**

Adaugă (m,e)

**Pre**: m ∈ M, e ∈ TElement

**Post**: m′ ∈ M,m′ = m ∪ {e}

Şterge (m,e)

**Pre**: m ∈ M, e ∈ TElement

**Post**: m′ ∈ M,m′ = m \ {e}

Caută (m,e)

**Pre**: m ∈ M, e ∈ TElement

**Post**: →{ Adevărat, daca e ∈ m

Fals, altfel

Dim (m)

**Pre**: m ∈ M

**Post**: → |m|

Vidă (m)

**Pre**: m ∈ M

**Post**: →{Adevărat, daca m = ∅

Fals, altfel

1. **Reprezentarea mulţimii**

Mulțimea va fi reprezentată pe o tabelă de dispersie („hashtable”). Aceasta

este un container ı̂n care se stochează perechi de cheie-valoare. Printr-o funcție

de dispersie, d′ : H → N, vor fi calculați indicii corespunzători fiecărei chei și

elementul va fi inserat pe poziția respectivă.

Pentru că această funcție nu este injectivă, pot apărea elemente care sunt

ı̂n coliziune, adică d′(x1) = d′(x2). Pentru rezolvarea coliziunilor se folosește

metoda adresării deschise. În acest caz se folosește o funcție de dispersie

modificată, d(c, i) = (d′(c) + i · c1 + i2 · c2)%m. Aceasta permite returnării

unei serii de valori pentru cheia dată, ı̂n funcție de parametrul i. Astfel, dacă

o locație este plină, se mărește parametrul i și se ı̂ncearcă la noua locație returnată

de funcția de dispersie, până când se găsește o poziție goală sau este

parcursă toată tabela.

* 1. **Mulţime**

1. **Subalgoritm** adaugă (m,e) - Complexitate: θ(1)

i←0

găsit←fals

**repetă**

poz←t.disp(e,i)

**dacă** t.e[poz] = 0 **atunci**

t.e[poz]←e

gasit←true

break;

**altfel dacă** t.e[poz] = e **atunci**

break;

**sf dacă**

i←i+1

**până când** i <> t.cap şi not găsit

**dacă** i <> t.cap **atunci**

t.dim←t.dim+1

**sf dacă**

**sf subalgoritm**

1. **Subalgoritm** şterge (m,e) -Complexitate: θ(1)

i*←*0

găsit*←*false

**repetă**

poz*←*t.disp(e,i)

**dacă** t.e[poz] = e **atunci**

t.e[poz]*←*NIL

t.dim*←*t.dim-1

gasit*←*true

break;

**sf dacă**

i*←*i+1

**până când** t.e[poz] = 0 sau i*←*t.cap sau not găsit

**sf subalgoritm**

1. **Funcţie** caută (m,e) -Complexitate: θ(1)

i*←*0

găsit*←*false

**repetă**

poz*←*t.disp(e,i)

**dacă** t.e[poz] = e **atunci**

găsit*←*true

break;

**altfel**

i*←*i+1

**sf dacă**

**până când** t.e[poz] = 0 sau i*←*t.cap

**sf funcţie**

1. **Funcţie** dim (m) -Complexitate: θ(1)

dim←t.dim

**sf functie**

1. **Funcţie** vidă (m) -Complexitate: θ(1)

vidă←t.dim = 0

**sf funcție**

* 1. **Iterator Mulţime**

1. **Subalgoritm** următor (it) -Complexitate: θ(1)

it.poz←it.poz+1

**cât timp** it.poz < it.m.cap şi it.m.e[it.poz]=NILL

it.poz←it.poz+1

**sf cât timp**

**sf subalgoritm**

1. **Funcţie** valid (it) -Complexitate: θ(1)

**dacă** it.poz < it.m.cap si it.m.e[it.poz]<>NILL

**atunci** valid←true

**altfel**

valid←false

**sf dacă**

**sf funcţie**

1. **Funcţie** element (it,e) -Complexitate: θ(1)

element←it.m.e[it.poz]

**sf funcţie**

* 1. **Calculare complexitate pentru adăugare**

În cazul favorabil, prima poziție ı̂ncercată este goală, deci bucla repetă se execută

o singură dată, așa că *T*(*n*) *∈ θ*(1).

În caz nefavorabil, tabela este plină, așa că bucla repetă se execută de n ori.

*T*(*n*) *∈ θ*(*n*).

Fie n numărul de elemente din mulțime și m capacitate mulțimii. Notăm

cu *α* = *n*/*m* factorul de ı̂ncărcare al mulțimii.

Fie X numărul de poziții ı̂ncercate pentru adăugare. X ∈ {1, 2, ..., n + 1}.

Probabilitatea ca prima poziție ı̂ncercată să fie goală este . Probabilitatea

ca a doua poziție ı̂ncercată să fie goală este . Probabilitatea ca a i-a

poziție ı̂ncercată să fie goală este

În cazul mediu, α constant, T(n) ∈ θ(1).

Global, complexitatea este de O(n).

1. **Proiectarea aplicaţiei**
2. **Algoritm** ADMINISTRARE-PARTICIPANŢI

**cât timp** execută = Adevărat execută

@ Citește opțiunea utilizatorului

**dacă** @ Utilizatorul a ales să adauge participanţi atunci

adaugaParticipant()

**sf dacă**

**dacă** @ Utilizatorul a ales să afişeze participanţi atunci

afişeazăParticipant()

**sf dacă**

**dacă** @ Utilizatorul a ales să şteargă participanţi atunci

ştergeParticipant()

**sf dacă**

**dacă** @ Utilizatorul a ales să filtreze participanţi atunci

filtreazăParticipanţi()

**sf dacă**

**dacă** @ Utilizatorul a ales să iasă din program **atunci**

execută*←*Fals

**sf dacă**

**sf cât timp**

2. **Subalgoritm** ADAUGĂ-PARTICIPANT - Complexitate: θ(1)

@ Citește de la tastatură numeParticipant

@ Citește de la tastatură scor adaugă(scor, numeParticipant)

**sf subalgoritm**

1. **Subalgoritm** AFIŞEAZĂ(m) -Complexitate: θ(n)

iterator(m,it)

**cât timp** valid(it) **execută**

element(it,e)

@ Afișează e

**sf cât timp**

**sf subalgoritm**

4. **Subalgoritm** ŞTERGE-PARTICIPANT

@ Citește de la tastatură numeParticipant

**sf subalgoritm**

1. **Subalgoritm** FILTREAZĂ-PARTICIPANŢI

**5. Bibliografie**

1. NICULESCU V., CZIBULA G., *Structuri* *fundamentale de date. O perspectivă*

*orientată-obiect*. Editura Casa Cărții de Știință, Cluj-Napoca,2011

2. TENENBAUM, A. M., LANGSAM, Y; AUGENSTEIN, M. J., *Data Structures Using*

*C*, Prentice Hall, 1990