# Structuri de date și algoritmi

Tipul abstract de date  **CoadaCuPrioritati – implementare folosind o lista simplu inlantuita cu inlanturile reprezentate pe**

**tablou.**

## Nume: Vasilut Lucian

## Grupa: 217

# Enunțul problemei

Să se dezvolte o aplicație care manageriază acordarea burselor pentru studenti. Aplicația trebuie să fie capabilă să efectueze operațiile de adăugare a unui student in lista ,eliminarea celui mai prioritar student , afisare primilor k studenti din lista , precum si afisarea studentului cu cea mai mare medie.

# **COADA CU PRIORITATI**– specificare și interfață

## CP = { cp | cp este o coada cu prioritati cu elemente (e,p) de tip TElement x Tprioritate }

## Operațiile din interfața Cozii cu prioritati:

***creează***(cp,R)

***{ creeaza o coada cu prioritati vida}***

pre: - R: T Prioritate x T Prioritate , R relatia de ordine

post: cp , cp = .

***adaugă***(cp,e,p)

***{ se adauga un element in coada cu prioritati }***

pre: cp Є *CP, e* Є ***Telement , p*** Є TPrioritate

post: cp’ Є *CP*, cp’ = cp (e,p)

***șterge***(cp,e,p)

***{ se sterge elementul `cel mai prioritar`}***

pre: cp Є *CP, cp*

post: e Є *Telement* , p Є Tprioritate , e este `cel mai prioritar` element din cp, p e prioritatea sa , cp’ Є *CP ,* cp’ = cp (e,p)

@ arunca exceptia daca coada este vida

***element(cp,e,p)***

***{ se acceseaza elementul `cel mai prioritar` }***

pre: cp Є *CP, cp*

post: cp’ = cp , e Є ***TElement , p*** Є ***Tprioritate***

***e este `cel mai prioritar` element din cp , p e prioritatea sa***

***@ arunca exceptie daca coada cu prioritati e vida***

***vidă***(cp)

pre: cp Є *CP*

post: **vidă**

***plina(cp)***

pre: cp Є *CP*

post: plina

***distruge***(cp)

***{destructor}***

pre: cp Є *CP*

post: cp a fost `distrusa` ( spatiul de memorie alocat a fost eliberat)

# Stabilirea reprezentării **cozii cu prioritati**

***Coada cu prioritati***

size: ***Întreg*** { capacitatea listei }

currentIndex: ***Întreg***

v: ***Elev[0 .. 100]*** { elementele memorate de lista }

next\_Elem: ***Intreg[0 .. 100] { legaturile dintre elementele din lista }***

*Elev:*

*nume: String*

*prenume: String*

*priority: Float*

## Operațiile din interfața **cozii cu prioritati:**

subalgoritm creeaza ( cp ) este: *{* θ*(1) }*

*size = 0*

*sfarsit subalgoritm*

subalgoritm adauga( cp , e ) este : *{ O(n) , unde n este dimensiunea listei }*

*prev*  0

current cp.currentIndex

*daca* size = 0 atunci

cp.size cp.size + 1

cp.v[cp.size] e

cp.next\_Elem[cp.size] -1

cp.currentIndex cp.size

@paraseste subalgoritmul

*sfarsit daca*

*daca getPriority(cp.v[cp.currentIndex]) < getPriority(e) atunci*

*cp.size cp.size + 1*

*cp.v[cp.size]* e

cp.next\_Elem[cp.size] cp.currentIndex

cp.currentIndex cp.size

@paraseste subalgoritmul

*sfarsit daca*

cat timp curent -1 si getPriority(cp.v[current]) >= getPriority(e) executa

prev current

current cp.next\_Elem[current]

sfarsit cat timp

daca current -1 atunci

next current

cp.size cp.size + 1

cp.v[cp.size] e

cp.next\_Elem[cp.size] next

cp.next\_Elem[prev] cp.size

altfel

cp.size cp.size + 1

cp.v[cp.size] e

cp.next\_Elem[prev] cp.size

cp.next\_Elem[cp.size] -1

sfarsit daca

sfarsit subalgoritm

subalgoritm pop( e ) este: *{* θ*(n) }*

daca cp.size > 0 atunci

current cp.currentIndex

cp.currentIndex cp.next\_Elem[cp.currentIndex]

daca current < cp.currentIndex atunci

cp.currentIndex cp.currentIndex – 1

sfarsit daca

x cp.v[current] (x e de tip Elev )

pentru icurrent,cp.size executa

cp.v[i] cp.v[i+1]

cp.next\_Elem[i] cp.next\_Elem[i+1]

sfarsit pentru

pentru i 1 , cp.size executa

daca cp.next\_Elem[i] >= current atunci

cp.next\_Elem[i] cp.next\_Elem[i]--

sfarsit pentru

cp.size cp.size - 1

e x

sfarsit subalgoritm

subalgoritm element(e) este: *{* θ*(1) }*

current cp.currentIndex

x = cp.v[current] ( x este un element de tip elev )

e x

sfarsit subalgoritm

functie Size() este : { θ*(1) }*

*Size* cp.size

sfarsit functie

functie curentIndex() este: { θ*(1) }*

*curentIndex cp.currentIndex*

*sfarsit functie*

*functie plina() este: {* θ*(1) }*

daca cp.size = 100 atunci

plina = 1

altfel

plina = 0

sfarsit daca

sfarsit functie

Implementarea aplicației

In implementarea aplicatiei am utilizat clase precum Repository , Console ,Elev , iar legaturile intre clase sunt realizate in main.cpp .

Elev:

nume , prenume: sir

priority : real

Repository:

pr\_queue:CP

Console:

repo:Repository

Implementarea operatiilor din Repository

subalgoritmul Add(e) este: { O(n) , unde n este dimensiunea listei }

add( e ) { add() e din interfata CP }

sfarsit subalgoritm

subalgoritmul Pop(e) este: { θ(n) , unde n este dimensiunea listei }

pop(e) { pop() e din interfata CP }

sfarsit subalgoritm

functie getSize() este: { θ(1) }

siz Size() { Size() e din interfata CP }

sfarsit functie

subalgoritm Element(e) este: { θ(1) }

element(e) { element() e din interfata CP }

sfarsit subalgoritm

subalgoritm getAll() este : { θ(n) , unde n este dimensiunea listei }

current curentIndex() { curentIndex() e din interfata CP }

cat timp current -1 executa

@afiseaza elementele cozii

current pr\_queue.next\_Elem[current] { next\_Elem e din interfata CP }

sfarsit cat timp

sfarsit subalgoritm

subalgoritm getAll2( k ) este : { θ(k) , unde k reprezinta primele k elemente ale listei }

current curentIndex() { curentIndex() e din interfata CP }

rz 1

cat timp current -1 si rz <= k executa

@afiseaza elementele cozii

current pr\_queue.next\_Elem[current] { next\_Elem e din interfata CP }

sfarsit cat timp

sfarsit subalgoritm

Implementarea operatiilor din Console

subalgoritm creeaza( ui , r ) este : {

ui.repo

sfarsit subalgoritm

subalgoritm distruge(ui) este : { }

@ elibereaza spatiul alocat

sfarsit subalgoritm

subalgoritm **printMenu**(ui) este:

tipareste “ Student grant application ”

tipareste “A) Adauga un student in lista burselor”

tipareste “D) Sterge studentul cel mai prioritar "

tipareste “S) Afiseaza primii k studenti in ordinea mediilor ”

tipareste "F) Afiseaza cel mai bun student "

tipareste “X) Exit”

sfarsit subalgoritm

subalgoritm **run**(ui) este: { complexitate , nu se cunoaste numarul de pasi }

ok 1

cat timp ok = 1 executa

printMenu(ui)

tipareste “Dati optinea dumneavoastra”

citeste opt

daca opt = ‘A’ atunci

add(ui)

sfarsit daca

altfel

daca opt = ‘D’ atunci

pop()

sfarsit daca

altfel

daca opt = ‘S’ atunci

tipareste " Introdu numarul de studenti pe care vrei sa-i afisezi "

citeste k

daca k > repo.getSize() atunci

tipareste " Nu exista in lista atatia studenti!! "

sfarsit daca

altfel

@stergem ecranul

repo.getAll2(k)

sfarsit daca

altfel

daca opt = 'F' atunci

daca repo.getSize() > 0

element(ui);

altfel

tipareste " In lista nu exista nici un student "

sfarsit daca

sfarsit daca

altfel

daca opt = 'X' atunci

@break

altfel

tipareste “Invalid option”

sfarsit subalgoritm

subalgoritm element(ui) este : { }

@ stergem ecranul

x repo->element();

tipareste “Studentul cu cea mai mare medie este: “

@ afisarea caracteristicilor studentului gasit

sfarsit subalgoritm

subalgoritm pop(ui) este : { }

daca repo.getSize() > 0 atunci

@stergem ecranul

x repo.pop()

repo->getAll()

altfel

tipareste " Nu avem un student prioritar!!! "

sfarsit daca

sfarsit subalgoritm

subalgoritm add(ui) este : { caz mediu O(n) }

tipareste “Introdu numele studentului “

citeste nume

tipareste “Introdu prenumele studentului “

citeste prenume

tipareste “ Introdu media studentului( 0-2 zecimale ); “

citeste media

@ creaza un elev de tip Elev ( x )

@ curata ecranul

repo.add(x)

repo.getAll()

sfarsit subalgoritm

# Class Diagram

