Nume: Bucșa Ecaterina
Grupa: 211

Nr. problemă: 6

Documentație

– TAD: Dicționar ordonat - implementare folosind un arbore binar de căutare–

Enunț: Noul director al Școlii Generale ”Ion Creangă” dorește să facă câteva modificări în ceea ce privește modul de asignare al notelor pentru elevi. El s-a gândit că un catalog electronic ar fi mult mai practic, atât pentru profesori, cât și pentru elevi. Astfel el a recurs la o persoana din domeniu pentru realizarea unei aplicații. Pentru fiecare materie trebuie să existe urmatoarele funcționalități:

Cerințe:

a) Înscrierea(adăugarea) unui elev la acea materie .
b) Căutarea după nume a unui elev.
c) Ștergerea unui elev.
d) Numărul total de elevi din clasă.
e) Afișarea numelor tuturor elevilor din clasă.
f) Afișarea elevilor împreună cu nota obținută.

Justificare: Problema se pliază perfect cu TAD Dicționar Ordonat deoarece într-un catalog elevii sunt ordonați alfabetic după nume, care este unic și reprezintă cheia. În orice moment se dorește ca în catalog să fie această ordine, indiferent dacă pe parcursul semestrului o să se mai transfere elevi la această școală sau nu, iar Dicționarul Ordonat ne oferă posibilitatea ca tot timpul să existe această relație.

❖ TAD – Dicționar Ordonat

➢ Specificare

Def : Dicționarul ordonat este un container în care elementele sunt perechi de forma (cheie,valoare). Cheile sunt unice, elementele dicționarului fiind ordonate pe baza cheii.

Domeniu:

DO = {do | do este un dicţionar ordonat cu elemente e = (c, v), c de tip TCheie = TComparibil**,** v de tip TValoare}

➢ Interfață

- **creează(do,R)**
  pre: R- relație
  post:do ∈ DO, do este dicţionarul ordonat vid (fără elemente)

- **adaugă(do, c, v)**
  pre: do ∈ DO**,** c ∈ **TCheie,** v ∈ **TValoare,**
  post: do' ∈ DO**,** do'= do + (c, v) (se adaugă în dicţionarul ordonat perechea (c, v))

- **caută(do, c)**
  pre: do ∈ DO**,** c ∈ **TCheie**
  post: returnează v ∈ **TValoare**, valoarea asociată cheii c, aruncă excepție în cazul în care nu există niciun element cu cheia c in do

- **şterge(do, c)**
  pre: do ∈ DO**,** c ∈ **TCheie**
  post: elementul cu cheia c a fost șters din do, dacă există; aruncă excepție în caz contrar

- **dim(do)**
  pre: do ∈ DO
  post: dim= dimensiunea dicţionarului do (numărul de elemente) ∈ N*****

- **vid(do)**
  pre: do ∈ DO
  post: vid= adevărat în cazul în care do e dicţionarul ordonat vid
  fals în caz contrar

- **iterator(do, i)**
  {se creează un iterator pe dicţionarul ordonat do}
  pre: do ∈ DO
  post: i ∈ I, i este iterator pe dicţionarul ordonat do

- **distruge(do)**
  pre: do ∈ DO
  post: dicţionarul ordonat do a fost 'distrus' (spaţiul de memorie alocat a fost eliberat)

➢ Stabilirea reprezentării

DO:

răd: ↑Nod { pointer spre rădăcina arborelui }

R: Relație { R- relație de ordine definită pentru a genera ordinea elementelor }

Nod:

e: TElement { informația utilă din nod }

st,dr: ↑Nod { pointeri spre subarborele stâng, respectiv drept, ce începe din nodul curent }

TElement:

c:TCheie

v:TValoare

UI:

do:DO

DOExcepție:

mesaj: șir de caractere

❖ TAD Iterator { iterator în inordine pe dicționar ordonat }

➢ Specificare

Domeniu:

I = { i | i este un iterator pe un containerul dicționar ordonat având elemente de tip TElement }

➢ Interfață

- **creeazăIt(i,do)**
  pre: do este un dicționar ordonat
  post: i ∈ I, s-a creat iteratorul i pe containerul do

- **element(i, e)**
  pre: i ∈ I, *curent* este valid(referă un element din container)
  post: *e* ∈ TElement, *e* este elemental current din iterație(elementul din container referit de *curent*)

- **valid(i)**
  pre: i ∈ I, *curent* este valid

  post: valid= { adevărat, dacă *curent* referă o poziție validă din container
  fals, altfel

- **următor(i)**
  pre: i ∈ I, *curent* este valid
  post: *curent'* referă 'următorul' element din container față de cel referit de *curent*

➢ Stabilirea reprezentării

IteratorDo:

curent: ↑Nod {pointer spre nodul curent }

do: DO { dicționarul ordonat pe care se va face iterarea }

s: Stivă { se va folosi o Stivă care va conține ↑Nod și care are în interfață operații specifice: *creează(s), vidă(s), adaugă(s, e), element(s, e), șterge(s, e)*. }

❖ Operații Dicționar Ordonat Excepție

Funcția creeazăDOExcepție(s) este ***{ θ(1)}***

pre: s: șir de caractere

post: ex ∈ DOExcepție

ex.mesaj ← s

SfFuncția

Funcția getMsg(doExp) este ***{ θ(1)}***

pre: doExp ∈ DOExcepție

getMsg ← doExp.mesaj

SfFuncția

❖ Operații TElement

Subalgoritm creeazăTElement(e,c,v) este ***{ θ(1)}***

pre: c ∈ TCheie, v ∈ TValoare

post: e ∈ TElement este elementul cu cheia c si valoarea v

e.c ← c

e.v ← v

SfSubalgoritm

Funcția getCheie(e) este ***{ θ(1)}***

pre: e ∈ TElement

post: returnează cheia elementului e

getCheie ← e.c

SfFuncția

Funcția getValoare(e) este ***{ θ(1)}***

pre: e ∈ TElement

post: returnează valoarea asociată elementului e

getValoare ← e.v

SfFuncția

Funcția cmp(e1,e2) este ***{ θ(1)}***

pre: e1, e2 ∈ TElement

post: cmp primește adevărat dacă cheia lui e1 este "<" decat cheia lui e2

Dacă e1.c < e2.c atunci

cmp ← adevărat

sfDacă

cmp ← fals

SfFuncția

❖ Operații Nod

Subalgoritm creeazăNod(e) este ***{ θ(1)}***

pre: e ∈ TElement

post: returnează ↑Nod

aloca(p)

[p].e ← e

[p].st ← NIL

[p].dr ← NIL

creeazăNod ← p

SfSubalgoritm

Funcția element(nod) este ***{ θ(1)}***

pre: nod: ↑Nod

post: se returneaza elementul nodului curent

element ← [nod].e

SfFuncția

Funcția stanga(nod) este ***{ θ(1)}***

pre: nod: ↑Nod

post: se returnează rădăcina subarborelui stâng al nodului curent

stanga ← [nod].st

SfFuncția

Funcția dreapta(nod) este ***{ θ(1)}***

pre: nod: ↑Nod

post: se returnează rădăcina dubarborelui drept al nodului curent

dreapta ← [nod].dr

SfFuncția

❖ Operații Dicționar Ordonat

Subalgoritm creează(do,R) este ***{ θ(1)}***

pre: R – relație; relația după care vor fi memorate elementele

post: do ∈ DO, do este dicţionarul ordonat vid (fără elemente)
do.rad ← NIL
do.R ← R

SfSubalgoritm

Funcția adaugă_recursiv(rad,nod) este ***O(h)***

pre: rad,nod: ↑Nod

post:

Dacă rad=NIL atunci
    rad=nod
altfel
    Dacă R([nod].e , [rad].e) atunci
        [rad].st = adaugă_recursiv([rad].st, nod)
    altfel
        [rad].dr = adaugă_recursiv([rad].dr, nod)

sfDacă

sfDacă

adaugă_recursiv ← rad

SfFuncția

Funcția caută_recursiv(rad,c) este ***O(h)***

pre: rad: ↑Nod , c ∈ TCheie

Complexitate caz favorabil: elementul cu cheia c se afla pe prima poziție ***=>θ(1)***

Complexitate caz defavorabil: elementul cu cheia c se afla în unul dintre nodurile frunză sau nu se găsește în arbore: T(n)=$\sum_{i=1}^{h} 1 = h \in \boldsymbol{\theta(h)}$

Complexitate caz mediu: $T(n) = \sum_{i=1}^{h} \frac{1}{h} * i = \frac{h+1}{2} \in \boldsymbol{\theta(h)}$

Dacă rad=NIL sau [rad].e.c = c atunci

caută_recursiv ← rad

altfel

Dacă c < [rad].e.c atunci

caută_recursiv([rad].st, c)

altfel

caută_recursiv([rad].dr, c)

sfDacă

sfDacă

SfFuncția

Funcția **șterge_recursiv(rad,c) este** **O(h)**

pre: rad: ↑Nod – rădăcina unui subarbore , c ∈ TCheie

post: se șterge nodul cu cheia egală cu c din subarborele de rădăcină p și se returnează rădăcina noului subarbore

Dacă rad=NIL atunci { s-a ajuns la subarbore vid }

caută_recursiv ← NIL

```
altfel
    Dacă c < [rad].e.c  atunci    { se șterge din subarborele stâng }
        [rad].st=șterge_recursiv([rad].st, c)
        șterge_recursiv ← rad
    altfel
        Dacă c > [rad].e.c atunci     { se șterge din subarborele drept }
            [rad].dr=șterge_recursiv([rad].dr, c)
            șterge_recursiv ← rad
        altfel  {am ajuns la nodul care trebuie șters}
            Dacă [rad].st ≠ NIL și [rad].dr ≠ NIL atunci        { nodul
                are și subarbore stâng și subarbore drept }
                temp ← minim([rad].dr)
                { se mută cheia minimă în rad }
                [rad].e ← [temp].e
                { se șterge nodul cu cheia minimă din subarborele
                drept }
                [rad].dr ← șterge_recursiv([rad].dr, [rad].e.c)
                șterge_recursiv ← rad
            altfel
                temp ← rad
                Dacă [rad].st = NIL atunci { nu există subarbore
                stâng }
                    repl ← [rad].dr
                altfel  { nu există subarbore drept }
                    repl ← [rad].st
                sfDacă
                { dealocă spațiul de memorare pentru nodul care
                trebuie șters }
                dealocă(temp)
                șterge_recursiv ← repl
```

sfDacă

sfDacă

sfDacă

sfDacă

SfFuncția

Funcția minim(p) este ***O(h)***

pre: p: ↑Nod , p≠ NIL

post: returnează adresa nodului cu cheia minima din subarborele de rădăcină p

CâtTimp [p].st ≠ NIL execută

p ← [p].st

SfCâtTimp

minim ← p

SfFuncția

Funcția dim_recursiv(rad,dm) este ***O(h)***

pre: rad: ↑Nod

post: returnează numărul de elemente

Dacă rad = NIL atunci

dim_recursiv ← 0

dim_recursiv ← 1 + dim_recursiv([rad].st, dm +1) + dim_recursiv([rad].dr, dm + 1)

SfFuncția

Subalgoritm adaugă(do,c,v) este ***O(h)***

creeazăTElement(elem,c,v)

nod ← creeazăNod(elem)

do.rad ← adaugă_recursiv(do.rad, nod)

@aruncă excepție dacă mai există un element cu cheia c

SfSubalgoritm

Funcția caută(do,c) este O(h)

rez ← caută_recursiv(do.rad, c)

Dacă rez =NIL atunci

@aruncă excepție dacă mai există un element cu cheia c

sfDacă

caută ← [rez].e.v

SfFuncția

Funcția șterge(do,c) este O(h)

caută(c) { aruncă excepție dacă nu există element cu cheia c }

do.rad ← caută_recursiv(do.rad, c)

SfFuncția

Funcția dim(do) este O(h)

dim ← dim_recursiv(do.rad, 0)

SfFuncția

Funcția vid(do) este { θ(1)}

Dacă do.rad = NIL atunci

vid ← adevărat

SfDacă

vid ← fals

SfFuncția

Funcția iterator(pDo) este { θ(1)}

pre: pDo –adresa dicționarului ordonat : ↑DO

iterator ← creeazăIt(pDo)

SfFuncția

❖ Operații Iterator

Funcția creeazăIt(do) este ***{ θ(1)}***

pre: do ∈ DO

itDo.do ← do

itDo.curent ← do.rad

creează(itDo.s)

creeazăIt ← itDo

SfFuncția

Funcția valid(it) este ***{ θ(1)}***

{ validitatea iteratorului }

valid ← (it.curent ≠ NIL) sau ┐ vidă(it.s)

SfFuncția

Funcția element(it) este

{ elementul curent al iteratorului }

CâtTimp it.curent ≠ NIL execută { se adaugă în stivă ramura stângă a elementului current }

adaugă(it.s, it.curent)

it.curent ← [it.curent].st

SfCâtTimp

șterge(it.s,it.curent) { se șterge nodul din vârful stivei }

element ← [it.curent].e

SfFuncția

Subalgoritm următor(it) este ***{ θ(1)}***

{ deplasează iteratorul }

it.curent ← [it.curent].dr

SfSubalgoritm

❖ Operații UI

Subalgoritm creeazăUI(ui,do) este ***{ θ(1)}***

ui.do ← do

SfSubalgoritm

Subalgoritm uiAdaugăElement(ui) este ***O(h)***

@afișează(Introduceți numele elevului:)

@citește nume { nume ∈ TCheie }

@ afișează(Introduceți nota:)

@citește nota { nota ∈ TValoare }

ui.do.adaugă(nume, nota)

@afișează(Elev adăugat cu succes!)

SfSubalgoritm

Subalgoritm uiCautaElementDupaCheie(ui) este ***O(h)***

@afișează(Introduceți numele elevului:)

@citește nume { nume ∈ TCheie }

nota ← ui.do.caută(nume) { nota ∈ TValoare }

@afișează nota

SfSubalgoritm

Subalgoritm uiȘtergeElement (ui) este ***O(h)***

@afișează(Introduceți numele elevului pe care doriți să îl eliminați din catalog:)

@citește nume { nume ∈ TCheie }

ui.do.șterge(nume)

@afișează(Elev eliminat cu succes!)

SfSubalgoritm

Subalgoritm uiDimensiune (ui) este ***O(h)***

@afișează(Numarul elevilor din catalog este:)

nr ← ui.do.dim() { nr: Întreg }

@ afișează nr

SfSubalgoritm

Subalgoritm uiAfișeazăChei (ui) este ***{ θ(n); n-numarul de elemente din arbore}***

Dacă ui.do.vid() atunci

@afișează(Nu există elevi în catalog!)

altfel

@afișează(Elevii din clasă sunt:)

it ← ui.do.iterator()

CâtTimp it.valid() execută

@afișează it.element().getCheie()

it.urmator()

SfCâtTimp

SfSubalgoritm

Subalgoritm uiAfișeazăValori (ui) este ***{ θ(n); n-numarul de elemente din arbore}***

Dacă ui.do.vid() atunci

@afișează(Nu există elevi adaugați în catalog!)

altfel

@afișează(Notele elevilor din clasă sunt:)

it ← ui.do.iterator()

CâtTimp it.valid() execută

@afișează it.element().getValoare()

it.urmator()

SfCâtTimp

SfSubalgoritm

Subalgoritm uiAfișeazăElemente (ui) este { θ(n); n-numarul de elemente din arbore}

Dacă ui.do.vid() atunci

@afișează(Nu există elevi adaugați în catalog!)

altfel

@afișează(Elevii din această clasă împreună cu notele lor sunt:)

it ← ui.do.iterator()

CâtTimp it.valid() execută

@afișează it.element()

it.urmator()

SfCâtTimp

SfSubalgoritm

Subalgoritm run (ui) este

cmd ← -1 { cmd: Întreg }

nrcmds ← 7

ok ← 1

CâtTimp ok = 1 execută

```
@încearcă
    @afișează (1. Adaugă un elev în catalog)
    @afișează (2. Afișează elevii împreună cu notele lor)
    @afișează (3. Elimină un elev din catalog)
    @afișează (4. Afișează numărul elevilor din calsă)
    @afișează (5. Afișează numele tuturor elevilor din clasă)
    @afișează (6. Afișează toate notele din catalog)
    @afișează (7. Afișează nota unui elev)
    @afișează (Introduceți comanda: )
    @citește cmd
    Dacă cmd >= 0 și  cmd <= nrcmds atunci
        Dacă cmd = 0
            ok ← 0
        altfel
            case cmd
            case 1 : ui.uiAdaugăElement()
            case 2 : ui.uiAfișeazăElemente()
            case 3 : ui.uiȘtergeElement()
            case 4 : ui.uiDimensiune()
            case 5 : ui.uiAfișeazăChei()
            case 6 : ui.uiAfișeazăValori()
            case 7 : ui.uiCautăElementDupăCheie()
        SfDacă
    altfel
        @afișează( Comandă invalidă! )
@prindeExcepție  e { e : DOExcepție }
    @afișează e.getMsg()
SfCâtTimp
```

SfSubalgoritm

Funcția main() este

creează(d, cmp)

creeazăUI(ui , d)

ui.run()

main ← 0

SfFuncția