Nume: Mateut Emilian-Gavril

Grupa 214

Structuri de date si algoritmi

Proiect

**Cerinta proiect:**

2. TAD Dictionar cu chei multiple (MultiMap) – implementare folosind o tabela de dispersie / rezolvare coliziuni prin liste intrepatrunse

**1.Enunt:**

Receptie hotel. Sa se implementeze o aplicatie care sa permita repartizarea clientilor in camerele unui hotel, sa inregistreze plecarea clientilor, sa afiseze clientii dintr-o camera anume, sa afiseze camerele care sunt pline ( camerele au K locuri )

-Multi-dictionarul este o alegere buna deoarece o camera (cheie) poate avea mai multi clienti (valori);

-Tabela de disperise este o alegere buna, deoarece e eficienta pentru reprezentarea dictionarelor si a multi-dictionarelor.

**2.Specificare TAD multi-dictionar + Interfata**

**a.Multi-dictionar**

Domeniu Multi-Dictionar:

D={md|md este un Multi-Dictionar cu elemente de tipul (c,v); c apartine lui TCheie, iar v apartine lui Tvaloare};

Operatii Multi-Dictionar:

Creeaza(md);

Adauga(md,c,v);

Sterge(md,c);

Apartine(md,c);

Dim(md);

getIterator(md);

Distruge(md);

* Creeaza(md)

Pre: -

Post:md apartine D

Descriere: se creeaza un Multi-Dictionar vid

* Adauga(md,c,v)

Pre:md apartine D, c apartine TCheie, v apartine TValoare

Post:md’=md+(c,v)

Descriere: adauga elementul(c,v) la Multi-Dictionarul md

Returneaza true daca a adaugat sau false daca nu

* Sterge(md,c)

Pre:md apartine D, c apartine TCheie

Post:md’=md-(c,v)

Descriere: sterge cheia c si toate valorile asociate acesteia din Multi-Dictionarul md

Returneaza true daca a sters, false daca nu

* Apartine(md,c)

Pre:md apartine D, c apartine TCheie

Post:-> 1, c apartine md

0, c nu apartine md

Descriere:functia apartine cauta cheia c in Multi-Dictionarul md

* Dim(md)

Pre:md apartine D

Post:->|md|

Descriere: functia dim returneaza dimensiunea Multi-Dictionarului md

* vid(d)

pre:d apartine D

post**:** vid =adevărat în cazul în care d e dicţionarul vid, fals în caz contrar

* chei(d, cm)

pre: d apartine *D*

post: cm apartine *M, c*m este mulţimea cheilor din dicţionarul d

descriere: returneaza numarul cheilor

* valori(d, c, cv)

pre: d apartine D

post: cv apartine C, c este colecţia valorilor cheii c

descriere: returneaza numarul valorilor

* iterator(md)

Pre:md apartine D

Post:->iterator pe md

Descriere:functia getIterator returneaza un iterator pe Multi-Dictionarul md

* Distruge(md)

Pre:md apartine D

Post:-

Descriere:distruge Multi-Dictionarul md

**b.Iterator**

Domeniu Iterator:

Di={it|it este un iterator pe un Multi-Dictionar}

Operatii Iterator:

Creeaza(it,md)

Prim(it)

Valid(it)

Urmator(it)

Curent(it)

* Creeaza(it,md)

Pre:md apartine D

Post:it apartine Di

Descriere:se creeaza un iterator pe Multi-Dictionarul md

* Prim(it)

Pre:it apartine Di, it –iterator pe md

Post:refera primul element al Multi-Dictionarului iterat

Descriere:se seteaza iteratorul pe primul element din Multi-Dictionar

* Valid(it)

Pre:it apartine Di, it –iterator pe md

Post:->true, iteratorul refera spre un element valid

False, altfel

Descriere:functia verifica daca iteratorul refera spre un element valid din Multi-Dictionar

* Urmator(it)

Pre:it apartine Di, it –iterator pe md

Post:it refera urmatorul element din md

Descriere:subalgoritmul muta iteratorul pe urmatorul element din md

* Curent(it)

Pre: it apartine Di, it –iterator pe md

Post:e apartine Telement, e elementul curent referit de it

Descriere:elementul curent referit de it din md

**3.Alegerea reprezentarii, operatiile, complexitati**

**a.Alegerea reprezentarii\***

Multi-dictionar:

m:intreg

c:TCheie[0..m-1] // presupunem ca e intreg

v:TValoare[0..m-1]

urm: (0..m)[0..m-1]

primliber:0..m

+dispersie:TFunctie

( dispersie(d,c) = c mod d.m )

Iteraror:

d:multi-dictionar

crt:intreg

\*Desi in implementare am alocat totul dinamic, pseudocod am considerat static

**b.operatii**

operatii iterator:

subalgoritm **creeaza**(it,d) este { ϴ(1) }

it.d <-d

it.crt <- 0

sfsubalgoritm

subalgoritm **prim**(it) este { ϴ(1) }

it.crt = 0;

cattimp(d.c[crt] != -1)

it.crt <-it.crt+1

sfcattimp

sfarsitsubalgoritm

Subalgoritm **urmator**(it)este {ϴ(1)}

repeta

it.crt <- it.crt+1

panacand(d.c[it.crt] = -1 si it.crt < d.m)

sfarsitsubalgoritm

functia **valid**(it) este {ϴ(1)}

daca it.crt < d.m si d.c[it.crt] <> -1

valid <-adevarat

valid <- fals

sfarsitfunctie

functia **getCurent**() {ϴ(1)}

getCurent <- it.crt

sfarsit functie {returneaza pozitia curenta a iteratorului}

functia **getCheie**( i intreg < [d].m) {ϴ(1)}

getCheie<-d.c[i]

sfarsitfunctie {returneaza cheia de pe o anumita pozitie}

functia **getValoare**( i intreg < [d].m) {ϴ(1)}

getValoare<-d.v[i]

sfarsitfunctie {retureaza valoarea de pe o anumita pozitie}

functia **getUrm**( i intreg < [d].m) {ϴ(1)}

getUrm<-d.urm[i]

sfarsitfunctie {returneaza numarul de pe o anumita pozitie din urm}

subalgoritm **cheie**(i,c) {ϴ(1)}

c <- d.c[it.crt]

sfarsitsubalgoritm { c primeste cheia curenta }

subalgoritm **valoare**(i,v) {ϴ(1)}

v <- d.v[it.crt]

sfarsitsubalgoritm { v primeste valoarea curenta }

Operatii multi-dictionar

Subalgoritm **creeaza**(d) este {ϴ(m)}

d.primliber <- 0

pentru i <-0,d.m-1 executa

d.c[i] <- -1

d.urm[i] <- -1

d.v[i] <- ””

sfarsitpentru

sfarsitsubalgoritm

subalgoritm actualizarePrimLiber(d) este {O(m)}

cattimp d.primliber <= d.m si d.c[primliber] <> -1

d.primliber <- d.primliber+1

sfarsitcattimp

sfarsitsubalgoritm

functia adauga(d,c,v) returneaza adevarat daca s-a realizat adaugarea, fals daca nu

functie **adauga**(d,c,v) {O(m)}

rez <- dispersie(d.c)

daca d.c[rez] = -1 atunci

d.c[rez] <- c

daca d.primliber = rez atunci

actualizarePrimLiber(d)

sfdaca

altfel cattimp rez <> -1 si (d.c[rez] <> c sau d.v[rez] <> v) executa

prec <- rez

rez <-d.urm[rez]

sfcattimp

daca rez <> -1 atunci

adauga <-fals

altfel daca d.primliber = d.m atunci

@ depasire

Altfel

d.c[d.primliber] <- c

d.v[d.primliber] <- v

d.urm[prec] <- d.primliber

actualizarePrimLiber(d)

sfdaca

sfdaca

sfdaca

sfarsitfunctie

functia sterge(d,c,v) returneaza true daca s-a efectuat stergerea, false daca nu

functia **sterge**(d,c,v) este {O(m)}

Subalgoritmul stergere(t, c) este: i <-d.dispersie(c) j <- -1 cat timp (i <> -1) si ((d.c[i] <> c) sau (d.v[i] <> v)) executa j <- i i <- d.urm[i] sfcattimp daca (i = -1) atunci return fals {cheie inexistenta}

altfel gata <- fals repeta prec <- i p <- d.urm[i]

cat timp (p <> -1) si (d.dispersie(d.c[p]) <> i) executa prec <- p p <-d.urm[p]

sfcattimp

daca (p = -1) atunci gata = adev altfel d.c[i] = d.c[p]

d.v[i] = d.v[p]

j <- prec

i <- p sf-daca pana cand gata daca (j <> -1) atunci d.urm[j] <- d.urm[i] sf-daca d.c[i] <- -1 d.v[i] <- ””

d.urm[i] <- -1 daca i < d.primLiber atunci d.primLiber = i sf-daca sf-daca

Sf-Stergere

Functia cauta(d,c,v) returneaza potizia pe care a gasit elementul, sau -1

functie **cauta**(d,c,v) este {O(m)}

rez <- d.dispersie(c)

cattimp rez <> -1 si ( d.c[rez] <> c sau d.v[rez] <> v) executa

rez <- d.urm[rez]

sfcattimp

cauta <- rez

sfarsitfunctie

functie **dim**() este {ϴ(m)}

nr <- 0

pentru i <- 0,d.m executa

daca d.c[ i ] <> -1 atunci

nr <- nr+1

sfdaca

sfpentru

dim <-nr

sfarsitfunctie

functie **vid**() este {O(m)}

pentru i <- 0,d.m executa

daca d.c[ i ] <> -1 atunci

vid <- adevarat

sfdaca

sfpentru

vid <- fals

sfarsitfunctie

**caz favorabil ϴ(1)**

**caz mediu ϴ(m)**

**caz defavorabil ϴ(m)**

**general (O(m))**

subalgoritm **iterator**(d) este { ϴ(1)}

iterator <- Iterator(d)

sfarsitsubalgoritm

int **chei**( d, cm ) este { ϴ(m)}

nr <- 0

pentru i <- 0,m executa

daca d.c[i] <> -1 atunci

ok <- 1

j <- 0

cattimp (j < nr si ok <> 0)

daca (d.c[j] == d.c[i])

ok <- 0

sfdaca

j <- j+1

sfcattimp

daca ok <> 0

ch[nr] <- c[i];

nr < - nr+1

sfdaca

sfdaca

sfpentru

chei <- nr

sfarsitfunctie

{ o cheie poate aparea de mai multe ori in tabela, dar o socotim numai o data la numaratoare}

int **valori**(d, c, cv) este { ϴ(m)}

nr <- 0

pentru i <- 0,m executa

daca (d.c[i] == c)

cv[nr] <- d.v[i]

nr <- nr+1

sfdaca

sfpentru

valori <-nr

sfarsitfunctie

**4.Proiectarea aplicatiei**

In folderul domain:

Clasa MultiMap

Clasa MultiMapIterator

Subalgoritm testmultimap()

In folderul repository:

Clasa Hotel care contine MultiMap si MultiMapIterator

In folderul ui:

Clasa Ui care contine Hotel

In app.cpp:

Main()

testMultiMap()

hotelrepo <- creeaza(Hotel)

ui <- creeaza (Ui(hotelrepo))

ui.showmenu()

distruge(ui)

sfarsitmain

**6.Diagrama**

Main menu

UI.showMenu

Ui.execMenu

oameniCamera

add

cameraOcupata

pline

leave

sterge

valorile

cheile

exist

cauta

valorile

cheile

adauga

Algoritmii din casutele ingrosate sunt in repository, in clasa Hotel, iar ultimele “frunze” sunt din interfata multidictionarului MultiMap (in domain). Iar in partea de jos, MultiMapIterator, apelate **direct din ui.execMenu**, prin repo

prim

urmator

getCurent

getUrm

getCheie

getUrm

getValoare

valid