Biblioteca de tipuri abstracte

Bia Beniamin,

16.01.2017

AC/I/30112

**1.Introducere**

Librarie implementeaza mai multe tipuri de structucturi de date de tip generic.Librarie este de forma unei librari statice.Biblioteca poate fi utila oricand vrem sa folosim o structura de date anume,obtinand timpi de rulare mai buni.De exemplu cand vrem sa tot cautam in structura,nu trebuie sa implementam strctura,putem folosi libraria

**2.Obiectivele proiectului**

1. Vector (implementat ca un tablou unidimensional alocat dinamic)

Operații a. CreateVector - creare instanță Vector

b. PrintVector – afișarea elementelor

c. AddVectorItems – adăugarea unui număr de elemente la finalul vectorului

d. PutVectorItem – adaugă un element la indexul specificat

e. GetVectorItem – returnează elementul de la indexul specificat

f. DeleteVectorItem – ștergerea elementului de la indexul specificat

g. SearchVectorItem – returnează indexul elementului care are valoarea căutată

h. SortVector – ordonarea crescătoare a elementelor

i. MergeVectors – combinarea a doi vectori: adăugarea la primul vector a elementelor din al doilea vector

j. DeleteVector – ștergerea elementelor și eliberarea memoriei

2. LinkedList (implementat ca o listă simplu înlănțuită)

Operații

a. CreateLinkedList - creare instanță LinkedList

b. PrintLinkedList – afișarea elementelor

c. AddLinkedListItem – adăugarea unui nou element la final

d. PutLinkedListItem – adaugă un element la pozitia specificată

e. GetLinkedListItem – returnează elemental de la poziția specificată

f. DeleteLinkedListItem – ștergerea și returnarea elementului cu valoarea specificată

g. SearchLinkedListItem – returnează elementul care are valoarea căutată

h. SortLinkedList – ordonarea crescătoare a elementelor

i. MergeLinkedLists – combinarea a două liste: adăugarea la prima listă a elementelor din a doua listă

j. DeleteLinkedList – ștergerea elementelor și eliberarea memoriei

3. HashTable (implementat folosind înlănțuire)

Operații

a. CreateHashTable - creare instanță HashTable

i. Se transmite ca parametru funcția de dispersie 1. Dacă e NULL se folosește o funcție implicită b. PrintHashTable – afișarea elementelor astfel încât să reflecte structura tabelei

c. AddHashTableItem – adăugarea unui nou element

i. Dacă factorul de umplere depășește 75% se face redimensionare (aprox. dublare)

d. DeleteHashTableItem – ștergerea elementului cu valoarea specificată

e. SearchHashTableItem – returnează dacă valoarea căutată este în tabela de dispersie

f. ReHashTable – crearea unei noi tabele de hashing și redistribuirea elementelor folosind o funcție de hashing specificată, urmată de ștergerea tabelei inițiale

g. DeleteHashTable – ștergerea elementelor și eliberarea memoriei

4. MinHeap (implementat ca un tablou alocat dinamic)

Operații

a. CreateHeap - creare instanță Heap

b. PrintHeap – afișarea elementelor sub formă de arbore

c. AddHeapItem – adăugarea unui nou element

d. GetHeapMin – returnează elementul minim

e. DeleteHeapMin – ștergerea și returnarea elementului minim

f. DeleteHeapItem – ștergerea elementului cu valoarea specificată

g. MergeMinHeaps – combinarea a două heapuri: adăugarea la primul heap a elementelor din al doilea heap, respectând invariantul

h. DeleteHeap – ștergerea elementelor și eliberarea memoriei

5. BinarySearchTree (implementat ca înlănțuire de noduri)

Operații

a. CreateBST - creare instanță BinarySearchTree

b. PrintBST – afișarea elementelor sub formă de arbore

c. PreorderBST – parcurgere și afișare în preordine

d. InorderBST – parcurgere și afișare în inordine

e. PostorderBST – parcurgere și afișare în postordine

f. AddBSTItem – adăugarea unui nou element la arbore

g. SearchBSTItem – returnează elementul care are valoarea căutată

h. DeleteBSTItem – ștergerea și returnarea elementului cu valoarea specificată

i. MergeBSTs – combinarea a doi arbori binari de căutare: adăugarea la primul arbore a elementelor din al doilea arbore, respectând invariantul

j. HightBST – returnează înălțimea subarborelui specificat

k. DeleteBST – ștergerea elementelor și eliberarea memoriei

6. BalancedBST (implementat folosind o abordare la alegere pentru reechilibrarea arborelui – AVL, AA, B, etc.)

Operații

a. CreateBalancedBST - creare instanță BalancedBST

b. PrinBalancedtBST – afișarea elementelor sub formă de arbore

c. PreorderBalancedBST – parcurgere și afișare în preordine

d. InorderBalancedBST – parcurgere și afișare în inordine

e. PostorderBalancedBST – parcurgere și afișare în postordine

f. AddBalancedBSTItem – adăugarea unui nou element la arbore

g. SearchBalancedBSTItem – returnează elementul care are valoarea căutată h. DeleteBalancedBSTItem – ștergerea și returnarea elementului cu valoarea specificată

i. MergeBalancedBSTs – combinarea a doi arbori binari de căutare echilibrați: adăugarea la primul arbore a elementelor din al doilea arbore, respectând invariantul

j. HightBalancedBST – returnează înălțimea subarborelui specificat

k. DeleteBalancedBST – ștergerea elementelor și eliberarea memori

ei

**3.Proiectare si implementare**

1.Vector-folosit ca o lista alocata dinamica

2.LinkedList-folosit ca o lista,si fiecare valoare are locatie urmatoarei

3.HashTable-implementat ca o lista inlantuita,initial are o functie de hash default,care poate fi schimbata

4.MinHeap este un tablou alocat dynamic

5.BinarySearch tree este un arbore binary nebalansat

6.BalancedBST-este un arbore care se autobalanseaza,orice operatie s-ar efectua,el ramane balansat avand timp de gasire a unui numar de o{logn}

**4.Manual de utilizare**

Pentru a fi utilizata librarie,tot ce trebuie sa faci este sa incluzi in proiectul personal din folderul Code\Dll fisierul dll impreuna cu fisierele header.Acum libraria este gata pentru a fi folosita.Ea poate primi comenzi fie sub forma de comanda text,fie sub forma de apelarea unor functii.

Pentru Vector:

Se include in partea de sus a fisierului,instructiunea:#include “Vector.h”

//creare instanta vector

void createVector(char\* name);

//comanda printare vector

void printVector(char\* name);

//adaugarea valorilor in instanta pasata ca parametru

void addVectorItems(command \*com);

//pune o valoare la o anumita pozitie

void putVectorItems(char\* name, int index, Item value);

//returneaza valoarea care se afla la pozitia indicata

Item getVectorItem(char\* name, int index);

//sterge un element la pozitia indicata

void deleteVectorItem(char\* name, int index);

//cauta o anumita valoare in vector

int searchVectorItem(char\* name, Item value);

//sorteaza valorile

void sortVector(char\* name);

//face merge la 2 instante

void mergeVectors(char \*inst1, char \*inst2);

//sterge instanta de vector specifficata

void deleteVector(char\* name);

Pentru LinkedList:

Se include in partea de sus a fisierului,instructiunea:#include “LinkedList.h”

//creare instanta vector

void createVector(char\* name);

//comanda printare vector

void printVector(char\* name);

//adaugarea valorilor in instanta pasata ca parametru

void addVectorItems(command \*com);

//pune o valoare la o anumita pozitie

void putVectorItems(char\* name, int index, Item value);

//returneaza valoarea care se afla la pozitia indicata

Item getVectorItem(char\* name, int index);

//sterge un element la pozitia indicata

void deleteVectorItem(char\* name, int index);

//cauta o anumita valoare in vector

int searchVectorItem(char\* name, Item value);

//sorteaza valorile

void sortVector(char\* name);

//face merge la 2 instante

void mergeVectors(char \*inst1, char \*inst2);

//sterge instanta de vector specifficata

void deleteVector(char\* name);

Pentru HashTable:

Se include in partea de sus a fisierului,instructiunea:#include “HashTable.h”

//creaza o instanta noua cu functia standard

void createHashTable(char\* name);

//creaza o instana noua cu o functie specificata

void createHashTableWithFunction(char\* name, unsigned long(\*hashFunction)(unsigned char\*));

//printeaza o lista sortata

void printHashTable(char\* name);

//adauga valori in instanta specificata

void addHashTableItems(command \*com);

//se executa rehash la tablou,primind o functie de dispersie

void reHashTable(char\* name, unsigned long(\*hashFunction)(unsigned char\*));

//sterge o valoare specificata

void deleteHashTableItem(char\* name, Item \*value, dataType type);

//cauta o valoare in tabel

int searchHashTableItem(char\* name, Item value, dataType type);

//sterge instanta specificata

void deleteHashTable(char\* name);

Pentru MinHeap:

Se include in partea de sus a fisierului,instructiunea:#include “MinHeap.h”

//creaza o instanta noua

void createHeap(char\* name);

//printeaza valorile din instanta specificata

void printHeap(char\* name);

//adauga elemente la instanta specificata

void addHeapItems(command \*com);

//returneaza valoarea minima din instana specificata

Item getHeapMin(char\* name);

//sterge valoarea cea mai mica

void deleteHeapMin(char\* name);

//sterge o valoare specificata daca este gasita

void deleteHeapItem(char\* name, Item \*value);

//face merge la doua instante

void mergeHeaps(char \*inst1, char \*inst2);

//sterge o instanta specificata

void deleteHeap(char\* name);

Pentru BinarySearchTree:

Se include in partea de sus a fisierului,instructiunea:#include “BinarySearchTree.h”

//creaza o instanta noua

void createBinarySearchTree(char\* name);

//printeaza valorile din instanta sub forma de arbore

void printBinarySearchTree(char\* name);

//adauga elementele in arbore

void addBinarySearchTreeItems(command \*com);

//printeaza elementele in preordine

void preorderBinarySearchTree(char\* name);

//printeaza elementele in postordine

void postorderBinarySearchTree(char\* name);

//printeaza elementele in inorder

void inorderBinarySearchTree(char\* name);

//sterge elementul care are valoarea egala cu ccea specificata

Item \*deleteBinarySearchTreeItem(char\* name, Item \*value);

//returneaza valoarea cautatata daca ii gasita

Item \*searchBinarySearchTreeItem(char\* name, Item \*value);

//face merge la 2 instante

void mergeBinarySearchTrees(char \*inst1, char \*inst2);

//sterge o instanta

void deleteBinarySearchTree(char\* name);

Pentru BalancedBinarySearchTree:

Se include in partea de sus a fisierului,instructiunea:#include “BalancedBST.h”

#include "Types.h"

//creaza o instanta noua

void createBalancedBST(char\* name);

//printeaza valorile din instanta sub forma de arbore

void printBalancedBST(char\* name);

//adauga elementele in arbore si face auto echilibrarea

void addBalancedBSTItems(command \*com);

//printeaza elementele in preordine

void preorderBalancedBST(char\* name);

//printeaza elementele in postordine

void postorderBalancedBST(char\* name);

//printeaza elementele in inorder

void inorderBalancedBST(char\* name);

//sterge elementul care are valoarea egala cu ccea specificata

Item \*deleteBalancedBSTItem(char\* name, Item \*value);

//returneaza valoarea cautatata daca ii gasita

Item \*searchBalancedBSTItem(char\* name, Item \*value);

//face merge la 2 instante

void mergeBalancedBSTs(char \*inst1, char \*inst2);

//sterge o instanta

void deleteBalancedBST(char\* name);

Pentru a folosi Testerul integrat,se deschide CommandPrompt,se navigeaza cu comanda cd la locatia proiectului,si acum se poate rula in urmatoarele moduri:

-Tester.exe denumireFisierIntrare pentu a rula un singur fisier de intrare.

-Tester.exe denumireFisier1 denumireFisier3 pentru a rula in intervalul specificat.Ex denumireFisier1, denumireFisier2, denumireFisier3

-Tester.exe runall -ruleaza toate fisierele cu extensia .in din directoriul Input

!!!Atentie,fisierele de intrare trebuie sa aiba extensia .in

5.Concluzii

Din proiectul propus,am realizat toate cerintele propuse.Pentru a verifica eventualele date eronate,am introdus un system de detectare a erorilor.Task-urile au fost in general ok,umpic de documentatie la balancedBST,dar a fost ok.Poate umpic cam lung,dar bine ca am reusit sa termin.