Bibliotecă de tipuri abstracte

Proiect de semestru – Cerc C

Documentație

Nume Prenume: Gabor George Cătălin

Data: 22.01.2017

Fac. / An / Grupa: CTI română/ II/ 30229

1. Introducere

Enunțul problemei:

Să se realizeze o bibliotecă c statică care să furnizeze principale operații pe urmatoarele tipuri de date abstracte: Vector, LinkedList, HashTable, Minheap, BinarySearchTree, BalancedBST.

Noțiuni:

Un tip de data abstract este o specificare a unui set de date de un anumit tip, împreună cu un set de operații ce pot fi executate cu aceste date. Implementare nu este vizibilă utilizatorul, însă acesta cunoaște setul de operații care pot fi efectuate și efectele lor.

Un tip generic este un tip care poate fi prelucrat de cod indiferent ce tipuri de bază sunt cuprinse in el.

Biblioteca statica este o colectie de fisiere obiect din care la etapa de compilare se preiau fragmente de cod(functii, clase sau alte resurse) si se adauga in aplicatia ce o foloseste.

2. Obiectivele proiectului

Operațiile furnizate sunt următoarele:

Vector : CreateVector, PrintVector, AddVectorItems, PutVectorItem, GetVectorItem, DeleteVectorItem, SearchVectorItem, SortVector, MergeVectors, DeleteVector;

LinkedList : CreateLinkedList, PrintLinkedList, AddLinkedListItem, PutLinkedListItem, GetLinkedListItem, DeleteLinkedListItem, SearchLinkedListItem, SortLinkedList, MergeLinkedLists, DeleteLinkedList;

HashTable: CreateHashTable, PrintHashTable, AddHashTableItem, DeleteHashTableItem, SearchHashTableItem, ReHashTable, DeleteHashTable;

MinHeap: CreateHeap, PrintHeap, AddHeapItem, GetHeapMin, DeleteHeapMin, DeleteHeapItem, MergeMinHeaps, DeleteHeap;

BinarySearchTree: CreateBST, PrintBST, PreorderBST, InorderBST, PostorderBST, AddBSTItem, SearchBSTItem, DeleteBSTItem, MergeBSTs, HightBST, DeleteBST;

BalancedBST: CreateBalancedBST, PrintBalancedBST, PreorderBalancedBST, InorderBalancedBST, PostorderBalancedBST, AddBalancedBSTItem, SearchBalancedBSTItem, DeleteBalancedBSTItem, MergeBalancedBSTs, HightBalancesBST, DeleteBalanedBST.

Tester: „runall" – rularea tuturor fisierelor de intrare, „nr" – rularea fisierului cu numarul „nr", „nr1 nr2"- rularea fisierelor cu numar cuprins intre „nr1" si „nr2".

3. Proiectare si implementare

Pentru implementarea structurilor cerute am folosit tipuri generice. Acestea furnizează următoarele operații : aloca_x, afisare_x, dezaloca_x, compare_x, citeste_x, copy_x, hash_x. Acestea sunt obligatorii pentru buna funcționare a testerului și a bibliotecii.

Tipurile de date abstracte cerute sunt implementate astfel:

Vector - este folosit un vector alocat dinamic, care își dublează dimensiunea când se umple;

LinkedList - este folosită inlăntuirea de noduri (fiecare nod retine o valoare și o referință la următorul nod);

HashTable - este folosit un vector alocat dinamic, care în momentul când factorul de umplere depașește 75% își mărește dimensiune la cel mai apropiat număr prim de dublul dimensiunii actuale. Este prezentă o functie de hash implicita ce produce un intreg pe baza elementului, numărului de octeți al tipului reținut in tabelă și dimensiunea tabelei. Aceasta poate fi inlocuită de o funcție definită de utilizator la crearea tabelei sau la operatia de ReHashTable;

MinHeap este implementat cu un tablou alocat dinamic, ce îsi dublează dimensiunea dacă se umple. Pentru a menține invariantul este folosită proprietatea: fii nodului părinte se află la pozițiile 2 * index_părinte + 1 și 2 * index_părinte + 2;

BinarySearchTree este implementat ca inlănțuire de noduri(fiecare nod reține o valoare, pointer la fiul stâng și pointer la fiul drept);

BalancedBST – implementat cu ajutorul unui arbore AA (inlăntuire de noduri in care fiecare nod are o valoare, un nivel , pointer la fiul stâng și pointer la fiul drept). Pentru a reechilibra arborele,

după operațiile de adăugare și ștergere a unui element, se utilizează algoritmul de inclinare si spargere(divizare).

Operațiile de „merge” sunt realizate folosind un vector alocat dinamic pe post de intermediar (se pun in acesta copii ale elementelor din sursa și doar daca s-au putut face toate copiile se adaugă în destinație), in acest fel se foloșește principiul „totul sau nimic”.

Apricația tester are rolul de a prelua, valida și efectua comenzile utilizatorului. Aceasta are implementate o serie de funcții generice (ex: create_new_struct) care pot lucra cu orice tip de dată abstract atata timp cât operația cerută se incadrează(ex: toate tipurile se pot afișa).

4. Manual de utilizare

Aplicatia tester ofera 2 tipuri : „persoana” si „intreg” ce pot fi utilizate. Insă utilizator poate defini și utliza alte tipuri, atâta timp cât sunt implementate toate operațiile cerute. De asemenea funcțiile generice ale testerului trebuie puțin modificate astfel încât să poată recunoaște aceste noi tipuri și să poată folosi funcțiile oferite de ele.

Teste-ul permite utilizatorul 3 opțiuni:

a.”runall” – se evaluează toate fișierele din directorul „input”;

b.”nr” – se evaluează fișierul cu numărul „nr” din directorul „input”(se poate introduce atît 1 cît și 01 sau 001, pentru a indica fișierul cu numărul 1);

c.”nr1 nr2” – se evaluează toate fișiere a căror număr este cuprins între „nr1” și „nr2” (se poate introduce atât 1 cât și 01 sau 001, pentru a indica fișierul cu numărul 1).

Orice altă comandă nu va fi execută, aplicația oferind utilizatorului un mesaj corespunzător.

5. Concluzii

Toate cerințele au fost realizate.

Cele mai dificile task-uri : realizarea tester-ului, implementarea HashTable, implementarea BalancedBST, validarea comenzilor și afișarea mesajelor corespunzătoare.

Cele mai plăcute task-uri : realizarea tester-ului, implementarea tipurilor abstracte cerute, utilizarea de tipuri generice.