Міністерство освіти і науки України  
Черкаський державний технологічний університет

Кафедра програмного забезпечення автоматизованих систем

**Звіт**

Про виконання лабораторної роботи №4

з дисципліни “Алгоритми і структура даних”

|  |  |
| --- | --- |
|  | Виконав:  Студент 2-го курсу  Групи ПЗС-1944  Білозор Д.О.  Перевірив:  Асистент Кафедри ПЗАС Олексюк В.В. |

Черкаси 2019

**Лабораторна робота № 4**

**Тема роботи:** Бінарні дерева.

**Мета роботи:** Набути навиків роботи з бінарними деревами.

**Завдання(Варіант 7):**

Визначте, чи є два дерева дзеркальним відображенням один одного.

**­Теоретичні відомості**

Двійкове (або Бінарне) дéрево пóшуку (англ. binary search tree, BST) в інформатиці — двійкове дерево, в якому кожній вершині x зіставлене певне значення val[x].

Бінарні дерева пошуку набагато ефективніші в операціях пошуку, аніж лінійні структури, в яких витрати часу на пошук пропорційні O(n), де n — розмір масиву даних, тоді як в повному бінарному дереві цей час пропорційний в середньому O(log2n) або O(h), де h — висота дерева (хоча гарантувати, що h не перевищує log2n можна лише для збалансованих дерев, які є ефективнішими в алгоритмах пошуку, аніж прості бінарні дерева пошуку).

Найпоширенішою операцією, яка виконується з бінарним деревом пошуку, є пошук в ньому певного ключа. Крім того, бінарні дерева пошуку підтримують такі запити, як пошук мінімального і максимального елемента, а також попереднього і наступного.

Процедура пошуку починається з кореня дерева і проходить вниз по дереву. Для кожного вузла х на шляху вниз його ключ key[x] порівнюється з переданим як параметр ключем k. Якщо ключі однакові, пошук завершується. Якщо k менше key[х], пошук триває в лівому піддереві х; якщо більше — то пошук переходить в праве піддерево. Ту ж процедуру можна записати ітеративно, «розгортаючи» рекурсію в цикл while.

Лістинг програми

#include <iostream>

#include <vector>

using namespace std;

typedef vector<int> T\_inums;

///////////////////////////////////////////////////////////////////////////////////

struct T\_node

{

int data;

T\_node\* left;

T\_node\* right;

};

///////////////////////////////////////////////////////////////////////////////////

void make\_node(T\_node\*& rp\_node, int val)

{

rp\_node = new T\_node;

rp\_node->data = val;

rp\_node->left = 0;

rp\_node->right = 0;

}

///////////////////////////////////////////////////////////////////////////////////

void sort\_insert(T\_node\*& rp\_tree, int val)

{

if(rp\_tree == 0)

{

make\_node(rp\_tree, val);

}

else if(val < rp\_tree->data)

{

sort\_insert(rp\_tree->left, val);

}

else

{

sort\_insert(rp\_tree->right, val);

}

}

///////////////////////////////////////////////////////////////////////////////////

void sort\_insert(T\_node\*& rp\_tree, T\_inums inums)

{

for(T\_inums::const\_iterator it = inums.begin(); it != inums.end(); ++it)

{

sort\_insert(rp\_tree, \*it);

}

}

///////////////////////////////////////////////////////////////////////////////////

void reverse\_sort\_insert(T\_node\*& rp\_tree, int val)

{

if(rp\_tree == 0)

{

make\_node(rp\_tree, val);

}

else if(val < rp\_tree->data)

{

reverse\_sort\_insert(rp\_tree->right, val);

}

else

{

reverse\_sort\_insert(rp\_tree->left, val);

}

}

///////////////////////////////////////////////////////////////////////////////////

void reverse\_sort\_insert(T\_node\*& rp\_tree, T\_inums inums)

{

for(T\_inums::const\_iterator it = inums.begin(); it != inums.end(); ++it)

{

reverse\_sort\_insert(rp\_tree, \*it);

}

}

///////////////////////////////////////////////////////////////////////////////////

bool are\_specularly\_similar(T\_node\* p\_treeA, T\_node\* p\_treeB)

{

if(p\_treeA == 0 && p\_treeB == 0)

{

return true;

}

if(p\_treeA == 0 || p\_treeB == 0)

{

return false;

}

if(p\_treeA->data != p\_treeB->data)

{

return false;

}

return are\_specularly\_similar(p\_treeA->left, p\_treeB->right)

&& are\_specularly\_similar(p\_treeA->right, p\_treeB->left);

}

///////////////////////////////////////////////////////////////////////////////////

int main()

{

setlocale(LC\_CTYPE, "ukr");

T\_inums inums;

inums.push\_back(2);

inums.push\_back(7);

inums.push\_back(1);

inums.push\_back(8);

inums.push\_back(2);

T\_node\* p\_tree\_l = 0;

sort\_insert(p\_tree\_l, inums);

T\_node\* p\_tree\_r = 0;

reverse\_sort\_insert(p\_tree\_r, inums);

if(are\_specularly\_similar(p\_tree\_l, p\_tree\_r))

{

cout << "Дерева дзеркально схожi.";

}

else

{

cout << "Дерева дзеркально не схожi.";

}

cout << endl;

if(are\_specularly\_similar(p\_tree\_l, p\_tree\_l))

{

cout << "Дерева дзеркально схожi.";

}

else

{

cout << "Дерева дзеркально не схожi.";

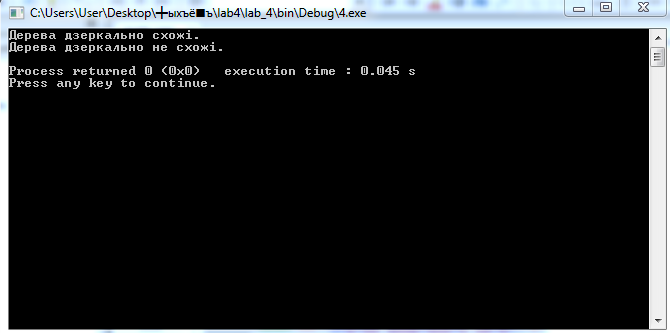
}

cout << endl;

return 0;

}

**Результат роботи**



**Висновок**:

Я набув навиків роботи з бінарними деревами, та визначив, чи є два дерева дзеркальним відображенням один одного.