Беларускі дзяржаўны тэхналагічны ўніверсітэт

Факультэт інфармацыйных тэхналогій

Кафедра праграмнай інжэнерыі

Лабараторная работа №11

Па дысцыпліне «Асновы алгарытмізацыі і праграмавання»

На тэму «Бінарныя дрэвы»

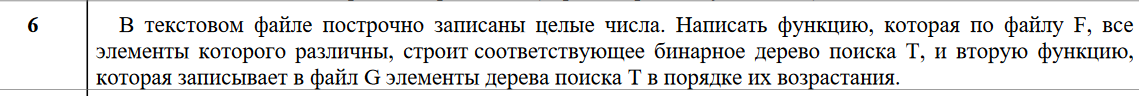
Выканала:

Студэнтка 1 курса 6 группы

Жучкевіч Кацярына Сяргееўна

Выкладчык: асс. Андронава М.В.

Мінск, 2024



#include <iostream>

#include <fstream>

#include <Windows.h>

using namespace std;

struct node {

int num;

node\* left;

node\* right;

node(int key) {

num = key;

left = nullptr;

right = nullptr;

}

~node() {

if (left)

delete left;

if (right)

delete right;

}

};

bool checkExistance(node\* root, int num) { //функцыя, якая праглядае, ці ёсць элемент з корнем ў дрэве

if (root == nullptr)//калі канчатак і праграма не знайшла...

return false;//тады адказ адмоўны

if (root->num == num)//а калі знайшлі

return true;//тады адказ не адмоўны

if (num < root->num)//і шукаем з дапамогай рэкурсіі

return checkExistance(root->left, num);

else

return checkExistance(root->right, num);

}

node\* insert(node\* root, int num) { //функцыя, якая ўстаўляе элемент ў дрэва

if (checkExistance(root, num))

return root;

if (root == nullptr)

return new node(num);

if (num < root->num)

root->left = insert(root->left, num);

else if (num > root->num)

root->right = insert(root->right, num);

return root;

}

int depth(node\* root) {//функцыя для таго, каб адпрацаваць карані

if (root == nullptr)

return 0;

int leftDepth = depth(root->left);//шукаем глыбіню

int rightDepth = depth(root->right);

return max(leftDepth, rightDepth) + 1;

}

void print(node\* root, ofstream& fout, int depth = 0) {//функцыя, якая выводзіць элементы, праглядаючы глыбіню дрэва

if (root == nullptr)

return;

print(root->right, fout, depth + 1);

for (int i = 0; i < depth; i++)

fout << "\t";

fout << ">>" << root->num << endl;

print(root->left, fout, depth + 1);

}

int main() {

SetConsoleOutputCP(1251);//каб літара "і" працавала))

SetConsoleCP(1251);

node\* root = nullptr;//ствараем корань

ifstream fout("F.txt");//адчыняем файл

if (fout.is\_open()) {//з якога будзем браць лічбы

int num;

while (fout >> num) {

root = insert(root, num);

}

}

else {

cout << "Памылка!" << endl;

return 1;

}

ofstream fin("G.txt");//адчыняем файл

if (fin.is\_open()) {//у які будзем капіраваць лічбы

print(root, fin);

}

else {

cout << "Памылка!" << endl;

return 1;

}

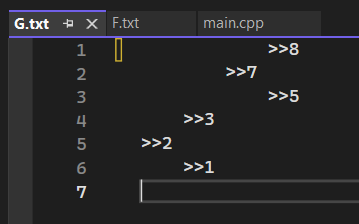
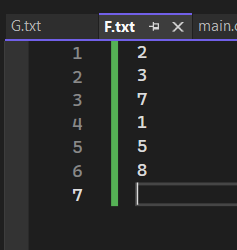
fout.close();//зачыняем файлы

fin.close();

delete root;

return 0;

}





#include <iostream>

#include <Windows.h>

using namespace std;

struct Tree //дерево

{

int data; //данные

Tree\* left, \* right;

};

Tree\* root = NULL;

float sum = 0;

Tree\* create(int data)//создание дерева

{

Tree\* newTree = new Tree;

newTree->data = data;

newTree->left = NULL;

newTree->right = NULL;

return newTree;

}

Tree\* insert(int data, Tree\* tree)//ввод элемента в дерево

{

if (tree == NULL)

{

return create(data);

}

if (tree->data > data)//поиск места для нового элемента

{

tree->left = insert(data, tree->left);

}

else if (tree->data < data)

{

tree->right = insert(data, tree->right);

}

return tree;

}

Tree\* delElem(int data, Tree\* tree)//удаление элемента дерева

{

if (tree->data > data)//поиск по дереву нужного элемента

{

tree->left = delElem(data, tree->left);

}

else if (tree->data < data)

{

tree->right = delElem(data, tree->right);

}

else//когда нашли элемент

{

if (tree->left = NULL)//если левая ветвь пуста

{

Tree\* survived = tree->right;

delete tree;

return survived;

}

else if (tree->right = NULL)//если правая ветвь пуста

{

Tree\* survived = tree->left;

delete tree;

return survived;

}

}

}

void output(Tree\* t, int level)//вывод дерева

{

if (t != NULL)

{

output(t->right, level + 1);

for (int i = 0; i < level; i++)

{

cout << " ";

}

cout << t->data << "\n";

output(t->left, level + 1);

}

}

void countSum(Tree\* tree, int& count, float& sum)//лічым суму элементаў

{

if (tree == NULL)

{

return;

}

sum = sum + tree->data;//шукаем суму

count++;

//а потым рэкурсіей знаходзім ўсю суму

countSum(tree->right, count, sum);//для правага

countSum(tree->left, count, sum);//для левага

}

float average(Tree\* tree) { //функцыя, ў якой умова задання

int count = 0;

countSum(tree, count, sum);

double temp = sum / count;//шукаем сярэдняе

return temp;

}

void main()

{

SetConsoleOutputCP(1251);//каб літара "і" працавала))

SetConsoleCP(1251);

int choice;

int ans;

for (;;)

{

cout << "1 - Зрабіць дрэва\n2 - Дадаць элемент\n3 - Выдаліць элемент\n4 - Вывад дрэва\n5 - Умова\n";

cin >> choice;

switch (choice)

{

case 1: cout << "\nУвядзіце элемент: "; cin >> ans; root = create(ans); break;

case 2: cout << "\nУвядзіце элемент: "; cin >> ans; insert(ans, root); break;

case 3: cout << "\nУвядзіце элемент: "; cin >> ans; delElem(ans, root); break;

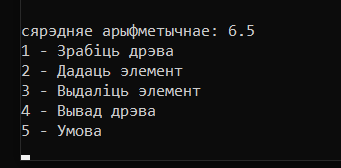
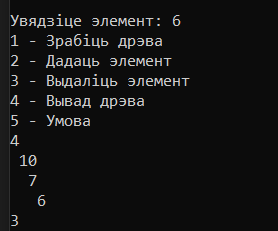
case 4: output(root, 0); break;

case 5: cout << "\n" << "сярэдняе арыфметычнае: " << average(root) << "\n"; break;

}

}

}





#include <iostream>

#include <Windows.h>

using namespace std;

struct Tree //дерево

{

int data; //данные

Tree\* left, \* right;

};

Tree\* root = NULL;

float sum = 0;

Tree\* create(int data)//создание дерева

{

Tree\* newTree = new Tree;

newTree->data = data;

newTree->left = NULL;

newTree->right = NULL;

return newTree;

}

Tree\* insert(int data, Tree\* tree)//ввод элемента в дерево

{

if (tree == NULL)

{

return create(data);

}

if (tree->data > data)//поиск места для нового элемента

{

tree->left = insert(data, tree->left);

}

else if (tree->data < data)

{

tree->right = insert(data, tree->right);

}

return tree;

}

Tree\* delElem(int data, Tree\* tree)//удаление элемента дерева

{

if (tree->data > data)//поиск по дереву нужного элемента

{

tree->left = delElem(data, tree->left);

}

else if (tree->data < data)

{

tree->right = delElem(data, tree->right);

}

else//когда нашли элемент

{

if (tree->left = NULL)//если левая ветвь пуста

{

Tree\* survived = tree->right;

delete tree;

return survived;

}

else if (tree->right = NULL)//если правая ветвь пуста

{

Tree\* survived = tree->left;

delete tree;

return survived;

}

}

}

void output(Tree\* t, int level)//вывод дерева

{

if (t != NULL)

{

output(t->right, level + 1);

for (int i = 0; i < level; i++)

{

cout << " ";

}

cout << t->data << "\n";

output(t->left, level + 1);

}

}

int checkRight(Tree\* tree){//функцыя, у якой шукаем колькасць правых элементаў

if (tree->right != NULL)

{

checkRight(tree->right);

sum++;

}

return sum;

}

void main()

{

SetConsoleOutputCP(1251);//каб літара "і" працавала))

SetConsoleCP(1251);

int choice;

int ans;

for (;;)

{

cout << "1 - Зрабіць дрэва\n2 - Дадаць элемент\n3 - Выдаліць элемент\n4 - Вывад дрэва\n5 - Умова\n";

cin >> choice;

switch (choice)

{

case 1: cout << "\nУвядзіце элемент: "; cin >> ans; root = create(ans); break;

case 2: cout << "\nУвядзіце элемент: "; cin >> ans; insert(ans, root); break;

case 3: cout << "\nУвядзіце элемент: "; cin >> ans; delElem(ans, root); break;

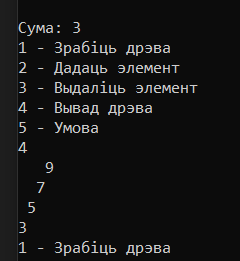
case 4: output(root, 0); break;

case 5: cout << "\n" << "Сума: " << checkRight(root) << "\n"; break;

}

}

}





#include <iostream>

#include <Windows.h>

using namespace std;

struct Tree //дерево

{

int data; //данные

Tree\* left, \* right;

};

Tree\* root = NULL;

float sum = 0;

Tree\* create(int data)//создание дерева

{

Tree\* newTree = new Tree;

newTree->data = data;

newTree->left = NULL;

newTree->right = NULL;

return newTree;

}

Tree\* insert(int data, Tree\* tree)//ввод элемента в дерево

{

if (tree == NULL)

{

return create(data);

}

if (tree->data > data)//поиск места для нового элемента

{

tree->left = insert(data, tree->left);

}

else if (tree->data < data)

{

tree->right = insert(data, tree->right);

}

return tree;

}

Tree\* delElem(int data, Tree\* tree)//удаление элемента дерева

{

if (tree->data > data)//поиск по дереву нужного элемента

{

tree->left = delElem(data, tree->left);

}

else if (tree->data < data)

{

tree->right = delElem(data, tree->right);

}

else//когда нашли элемент

{

if (tree->left = NULL)//если левая ветвь пуста

{

Tree\* survived = tree->right;

delete tree;

return survived;

}

else if (tree->right = NULL)//если правая ветвь пуста

{

Tree\* survived = tree->left;

delete tree;

return survived;

}

}

}

void output(Tree\* t, int level)//вывод дерева

{

if (t != NULL)

{

output(t->right, level + 1);

for (int i = 0; i < level; i++)

{

cout << " ";

}

cout << t->data << "\n";

output(t->left, level + 1);

}

}

int countLeaves(Tree\* tree){//функцыя, у якой шукаем колькасць элементаў

if (tree->left != NULL)

{

countLeaves(tree->left);

sum++;

}

if (tree->right != NULL)

{

countLeaves(tree->right);

sum++;

}

return sum;

}

void main()

{

SetConsoleOutputCP(1251);//каб літара "і" працавала))

SetConsoleCP(1251);

int choice;

int ans;

for (;;)

{

cout << "1 - Зрабіць дрэва\n2 - Дадаць элемент\n3 - Выдаліць элемент\n4 - Вывад дрэва\n5 - Умова\n";

cin >> choice;

switch (choice)

{

case 1: cout << "\nУвядзіце элемент: "; cin >> ans; root = create(ans); break;

case 2: cout << "\nУвядзіце элемент: "; cin >> ans; insert(ans, root); break;

case 3: cout << "\nУвядзіце элемент: "; cin >> ans; delElem(ans, root); break;

case 4: output(root, 0); break;

case 5: cout << "\n" << "Сума: " << countLeaves(root) << "\n"; break;

}

}

}

