Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №11

По дисциплине «Основы алгоритмизации и программирования»

На тему « Бинарные деревья»

Выполнил:

Студент 1 курса 8 группы

Лужецкий Владислав Константинович

Преподаватель: асс. Андронова М.В.

2024, Минск

Оглавление

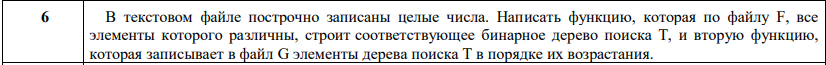
[Основное задание(6 Вариант): 2](#_Toc164294432)

[Дополнительное задание 1: 4](#_Toc164294433)

[Дополнительное задание 2: 7](#_Toc164294434)

[Дополнительное задание 3: 10](#_Toc164294435)

# Основное задание(6 Вариант):



//Заменил название файлов для удобства. Тут немного больше функций и в целом сделано «по-своему», но выполняет условие, где идёт чтение с файла input.txt и построение бинарного дерева поиска в файле output.txt

#include <iostream>

#include <fstream>

using namespace std;

struct node {

int value;

node\* left;

node\* right;

node(int key) {

value = key;

left = nullptr;

right = nullptr;

}

~node() {

if (left)

delete left;

if (right)

delete right;

}

};

// Проверяет, существует ли элемент со значением value в дереве с корнем root

bool exists(node\* root, int value) {

if (root == nullptr)

return false;

if (root->value == value)

return true;

if (value < root->value)

return exists(root->left, value);

else

return exists(root->right, value);

}

// Вставляет элемент со значением value в дерево с корнем root

node\* insert(node\* root, int value) {

if (exists(root, value))

return root;

if (root == nullptr)

return new node(value);

if (value < root->value)

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

else if (value > root->value)

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

return root;

}

// Вычисляет глубину дерева с корнем root

int depth(node\* root) {

if (root == nullptr)

return 0;

int leftDepth = depth(root->left);

int rightDepth = depth(root->right);

return max(leftDepth, rightDepth) + 1;

}

// Выводит значения элементов дерева, используя обход в глубину

void print(node\* root, ofstream& outputFileStream, int depth = 0) {

if (root == nullptr)

return;

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

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

outputFileStream << " ";

outputFileStream << "|--" << root->value << endl;

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

}

int main() {

setlocale(0, "");

// Имя файла для чтения целых чисел

string inputFile = "input.txt";

// Имя файла для записи элементов дерева

string outputFile = "output.txt";

// Создание корня дерева

node\* root = nullptr;

// Чтение целых чисел из файла и построение дерева

ifstream file(inputFile);

if (file.is\_open()) {

int num;

while (file >> num) {

root = insert(root, num);

}

file.close();

}

else {

cout << "Не удалось открыть файл для чтения." << endl;

return 1;

}

// Запись элементов дерева в порядке возрастания в файл

ofstream outputFileStream(outputFile);

if (outputFileStream.is\_open()) {

print(root, outputFileStream);

outputFileStream.close();

}

else {

cout << "Не удалось открыть файл для записи." << endl;

return 1;

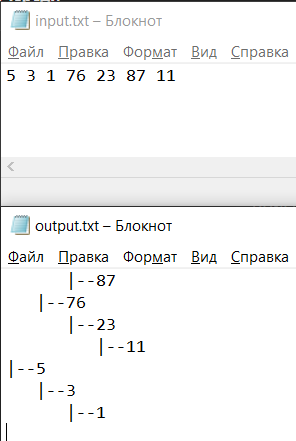
}

// Освобождение памяти, удаляя дерево

delete root;

return 0;

}



# Дополнительное задание 1:



#include <iostream>

using namespace std;

struct node {

int value;

node\* left;

node\* right;

node(int key) {

value = key;

left = nullptr;

right = nullptr;

}

~node() {

if (left)

delete left;

if (right)

delete right;

}

};

// Проверяет, существует ли элемент со значением value в дереве с корнем root

bool exists(node\* root, int value) {

if (root == nullptr)

return false;

if (root->value == value)

return true;

if (value < root->value)

return exists(root->left, value);

else

return exists(root->right, value);

}

// Вставляет элемент со значением value в дерево с корнем root

node\* insert(node\* root, int value) {

if (exists(root, value))

return root;

if (root == nullptr)

return new node(value);

if (value < root->value)

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

else if (value > root->value)

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

return root;

}

// Вычисляет глубину дерева с корнем root

int depth(node\* root) {

if (root == nullptr)

return 0;

int leftDepth = depth(root->left);

int rightDepth = depth(root->right);

return max(leftDepth, rightDepth) + 1;

}

// Выводит значения элементов дерева, используя обход в глубину

void print(node\* root, int depth = 0) {

if (root == nullptr)

return;

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

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

cout << " ";

cout << "|--" << root->value << endl;

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

}

int sumDescendants(node\* root, int target) {

// Базовый случай: если корень пустой, вернуть 0 (нет дочерних элементов).

if (root == nullptr)

return 0;

// Если значение корня равно целевому значению, вычислить сумму дочерних элементов и вернуть её.

if (root->value == target) {

int childrenSum = sumDescendants(root->left, target) + sumDescendants(root->right, target);

return childrenSum;

}

// Рекурсивно вычислить сумму дочерних элементов для левого и правого поддеревьев,

// а также сумму значений текущего корня, и вернуть их сумму.

int leftSum = sumDescendants(root->left, target);

int rightSum = sumDescendants(root->right, target);

return root->value + leftSum + rightSum;

}

int main() {

setlocale(0, "");

// Создание корня дерева

node\* root = nullptr;

// Ввод целых чисел и построение дерева

int num;

cout << "Введите целые числа (ввод завершится при вводе отрицательного числа):" << endl;

while (cin >> num && num >= 0) {

root = insert(root, num);

}

// Вывод элементов дерева

cout << "Элементы дерева в порядке возрастания:" << endl;

print(root);

int target;

cout << "Введите число, содержащееся в корне: ";

cin >> target;

int totalSum = sumDescendants(root, target);

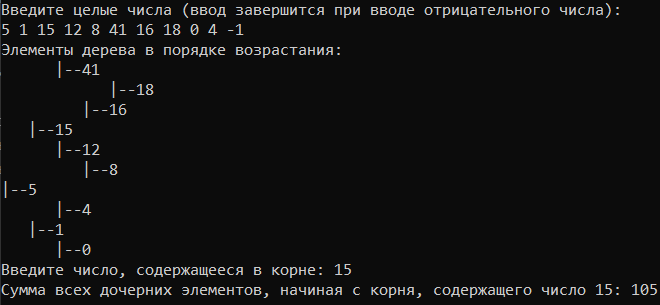
cout << "Сумма всех дочерних элементов, начиная с корня, содержащего число " << target << ": " << totalSum << endl;

// Освобождение памяти, удаляя дерево

delete root;

return 0;

}



# Дополнительное задание 2:



#include <iostream>

using namespace std;

struct node {

int value;

node\* left;

node\* right;

node(int key) {

value = key;

left = nullptr;

right = nullptr;

}

~node() {

if (left)

delete left;

if (right)

delete right;

}

};

// Проверяет, существует ли элемент со значением value в дереве с корнем root

bool exists(node\* root, int value) {

if (root == nullptr)

return false;

if (root->value == value)

return true;

if (value < root->value)

return exists(root->left, value);

else

return exists(root->right, value);

}

// Вставляет элемент со значением value в дерево с корнем root

node\* insert(node\* root, int value) {

if (exists(root, value))

return root;

if (root == nullptr)

return new node(value);

if (value < root->value)

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

else if (value > root->value)

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

return root;

}

// Вычисляет глубину дерева с корнем root

int depth(node\* root) {

if (root == nullptr)

return 0;

int leftDepth = depth(root->left);

int rightDepth = depth(root->right);

return max(leftDepth, rightDepth) + 1;

}

// Выводит значения элементов дерева, используя обход в глубину

void print(node\* root, int depth = 0) {

if (root == nullptr)

return;

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

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

cout << " ";

cout << "|--" << root->value << endl;

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

}

int sumTree(node\* root) {

if (root == nullptr)

return 0;

int sum = root->value;

sum += sumTree(root->left);

sum += sumTree(root->right);

return sum;

}

int main() {

setlocale(0, "");

// Создание корня дерева

node\* root = nullptr;

// Ввод целых чисел и построение дерева

int num;

cout << "Введите целые числа (ввод завершится при вводе отрицательного числа):" << endl;

while (cin >> num && num >= 0) {

root = insert(root, num);

}

// Вывод элементов дерева

cout << "Элементы дерева в порядке возрастания:" << endl;

print(root);

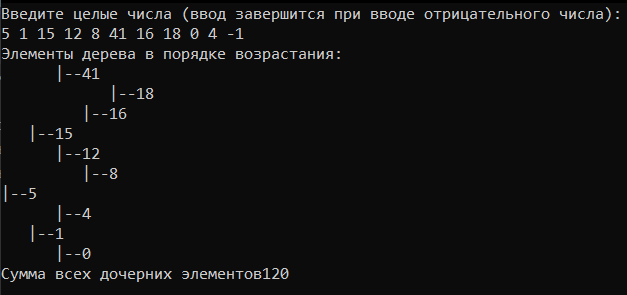
cout << "Сумма всех дочерних элементов" << sumTree(root) << endl;

// Освобождение памяти, удаляя дерево

delete root;

return 0;

}



# Дополнительное задание 3:



void countAndPrintBranches(node\* root, int level, int target) {

if (root == nullptr)

return;

if (level == 0) {

// Достигли целевого уровня, выводим значение текущего узла

if (root->value == target) {

cout << root->value << " ";

}

return;

}

// Рекурсивно вызываем функцию countAndPrintBranches() для всех дочерних элементов на уровне level - 1

countAndPrintBranches(root->left, level - 1, target);

countAndPrintBranches(root->right, level - 1, target);

}

int main() {

setlocale(0, "");

// Создание корня дерева

node\* root = nullptr;

// Ввод целых чисел и построение дерева

int num;

cout << "Введите целые числа (ввод завершится при вводе отрицательного числа):" << endl;

while (cin >> num && num >= 0) {

root = insert(root, num);

}

// Вывод элементов дерева

cout << "Элементы дерева в порядке возрастания:" << endl;

print(root);

int level;

cout << "Введите уровень: ";

cin >> level;

int target;

cout << "Введите число: ";

cin >> target;

cout << "Элементы ветвей на уровне " << level << ", содержащие число " << target << ":" << endl;

countAndPrintBranches(root, level, target);

// Освобождение памяти, удаляя дерево

delete root;

Ы

return 0;

}

