|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт информационных технологий |
| Кафедра вычислительной техники |
|  |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1** | |
| **по дисциплине** | |
| «Структуры и алгоритмы обработки данных» | |
|  | |
| Выполнил студент группы ИКБО-05-18 | Токар И.И. |
| Принял | Расулов М.М. |

|  |  |  |
| --- | --- | --- |
| Лабораторные работы выполнены | «\_\_»\_\_\_\_\_\_\_201\_\_ г. |  |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. |  |

Москва 2020 г.

**Описание задания**

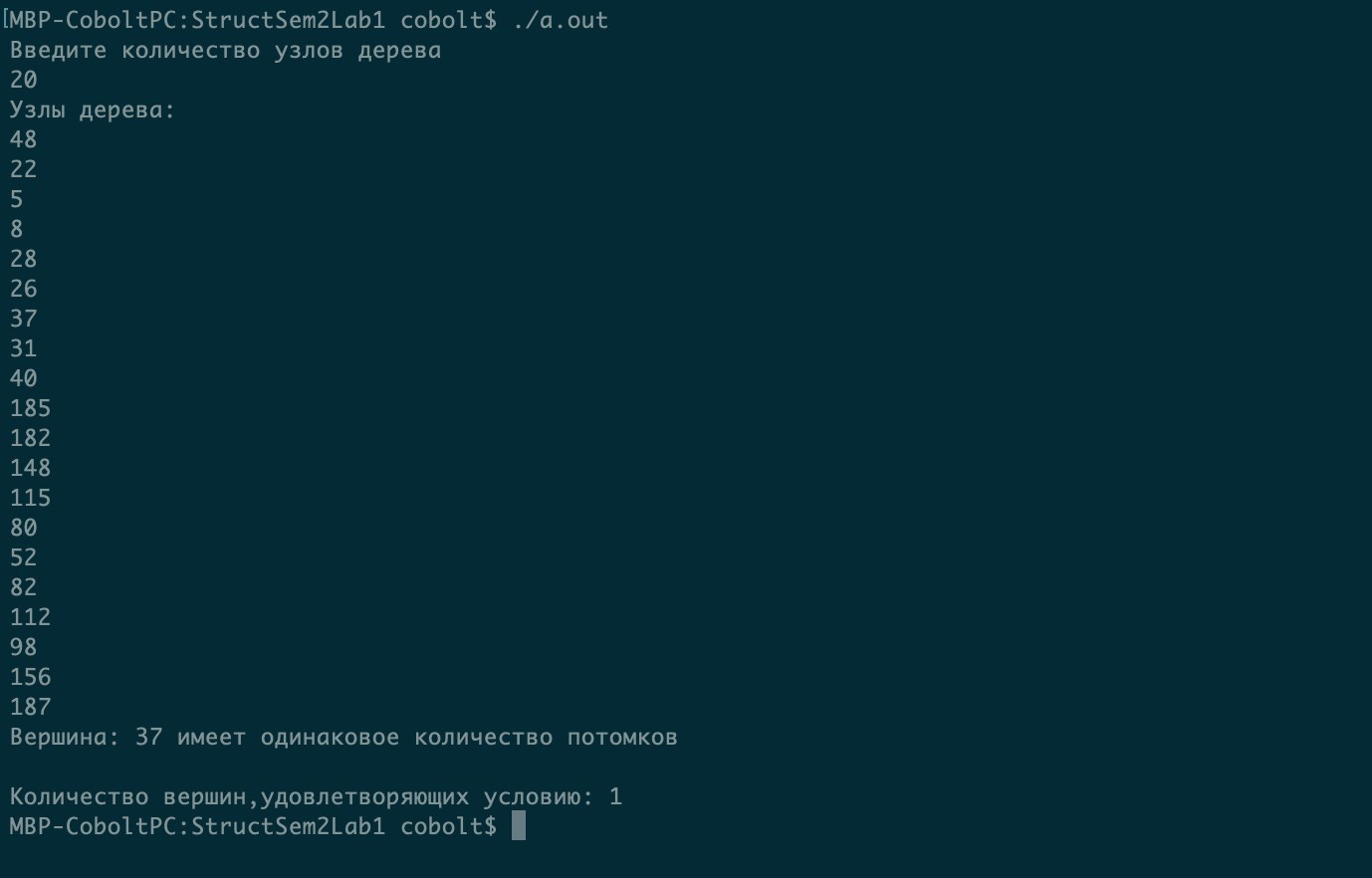
Задание 19:

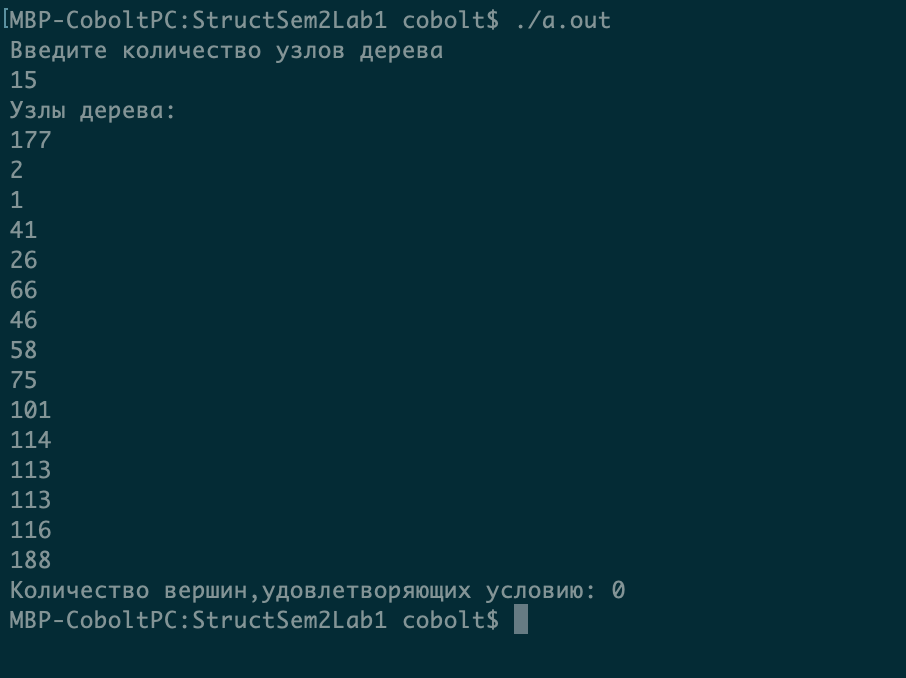
Вычислите количество вершин, у которых равны или высоты поддеревьев, или количество потомков в правом и левом поддеревьях.

**Описание хода решения заданий**

Чтобы найти такие элементы, необходимо реализовать две функции, подсчета высоты слева и справа, а также подсчета потомков в этих местах. Однако необходимо так же реализовать рекурсивный алгоритм обхода и функцию проверки результатов. Сначала активируется функция обхода, где каждый узел проверяется исходя из вышеупомянутых критериев, затем, если условия были выполнены, выводится информация и инкрементируется переменная подсчета нужных узлов. Затем идет следующий узел, пока не обойдем все дерево, но нужно заметить, что каждый узел представляется в функциях проверки как корень дерева, но в функции вычисления высоты поддеревьев мы смотрим только левые и правые поддеревья, то есть передаем ссылки на них, относительно корня.

**Тестирование**

**  
Рисунок 1 Тестирование программы**

****

**Выводы**

В данной лабораторной работе были получены практические и теоретические навыки по работе с деревьями и алгоритмами для деревьев. **Список используемой литературы**

1. Стивен**,**П. Язык программирования C лекции и упражнения / П. Стивен.— 5-е изд., перераб. и доп.— М.:Sams, 2013.— 960 с.
2. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре.— 4-е изд., перераб. и доп.— М.: Питер, 2004.— 924 с.
3. Седжвик, Р. Алгоритмы на C++ / Р. Седжвик.— М.:Вильямс, 2017.— 1056 с.
4. Кубенский А.А. Структуры и алгоритмы обработки данных: объектно-ориентированный подход и реализация на C++ / А.А. Кубенский.— М.: БХВ-Петербург, 2017.— 300 с.

**Приложения**

**Main.cpp**

#include <iostream>

#include "Tree.h"

#include <time.h>

#include <random>

#include <locale.h>

using namespace std;

int main()

{

srand(time(nullptr));

setlocale(LC\_ALL, "rus");

Tree intTree;

int countOfNodes;

cout << "Введите количество узлов дерева\n";

cin >> countOfNodes;

cin.ignore();

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

intTree.addNewNode(1 + rand() % 200);

cout << "Узлы дерева:\n";

intTree.printTree(intTree.getRoot());

intTree.treeWalk(intTree.getRoot());

intTree.getcountOfNeededNodes();

return 0;

}

**TreeNode.h**

class TreeNode

{

protected:

TreeNode\* leftNodePointer;

TreeNode\* rightNodePointer;

TreeNode\* parentPointer;

int data;

public:

TreeNode(const int& data)

{

this->data = data;

this->leftNodePointer = nullptr;

this->rightNodePointer = nullptr;

this->parentPointer = nullptr;

}

int getTreeNodeData()

{

return this->data;

}

void addLeftNode(TreeNode\*& node)

{

this->leftNodePointer = node;

}

void addRightNode(TreeNode\*& node)

{

this->rightNodePointer = node;

}

void addParentNode(TreeNode\*& node)

{

this->parentPointer = node;

}

TreeNode\* getLeftNode()

{

return this->leftNodePointer;

}

TreeNode\* getRightNode()

{

return this->rightNodePointer;

}

TreeNode\* getParentNode()

{

return this->leftNodePointer;

}

};

**Tree.h**

#include "TreeNode.h"

#include <iostream>

#include <algorithm>

using namespace std;

class Tree

{

private:

TreeNode\* root;

int countOfLeftNodes;

int countOfRightNodes;

int leftHeight;

int rightHeight;

int neededNodes;

public:

Tree()

{

this->root = nullptr;

this->countOfLeftNodes = 0;

this->countOfRightNodes = 0;

this->leftHeight = 0;

this->rightHeight = 0;

this->neededNodes = 0;

}

// Операции с корнем

TreeNode\* getRoot()

{

return this->root;

}

void setRoot(TreeNode\* newRoot)

{

this->root = newRoot;

}

// Получение количества элементов слева и справа

int getCountOfLeftNodes()

{

return this->countOfLeftNodes;

}

int getCountOfRightNodes()

{

return this->countOfRightNodes;

}

void getcountOfNeededNodes()

{

cout << "Количество вершин,удовлетворяющих условию: " << this->neededNodes << endl;

}

void addNewNode(const int& data)

{

TreeNode\* newNode = new TreeNode(data);

TreeNode\* rootBuffer1 = root;

TreeNode\* rootBuffer2 = nullptr;

// Поиск места узла

while (rootBuffer1 != nullptr)

{

// Запоминаем родителя

rootBuffer2 = rootBuffer1;

if (data < rootBuffer1->getTreeNodeData())

rootBuffer1 = rootBuffer1->getLeftNode();

else

rootBuffer1 = rootBuffer1->getRightNode();

}

newNode->addParentNode(rootBuffer2);

// Добавление узла

if (this->root == nullptr)

this->root = newNode;

else

{

if (data < rootBuffer2->getTreeNodeData())

rootBuffer2->addLeftNode(newNode);

else

rootBuffer2->addRightNode(newNode);

}

}

void printTree(TreeNode\* node)

{

if (node == nullptr)

return;

cout << node->getTreeNodeData() << endl;

printTree(node->getLeftNode());

printTree(node->getRightNode());

}

void countNodes(TreeNode\* node, TreeNode\* rootBuffer, string& direction)

{

if (node == nullptr)

return;

if (rootBuffer->getTreeNodeData() < node->getTreeNodeData())

direction = "right";

if (direction == "left" && node->getTreeNodeData() != rootBuffer->getTreeNodeData())

this->countOfLeftNodes++;

if (direction == "right" && node->getTreeNodeData() != rootBuffer->getTreeNodeData())

this->countOfRightNodes++;

countNodes(node->getLeftNode(), rootBuffer, direction);

countNodes(node->getRightNode(), rootBuffer, direction);

}

int calculateHeight(TreeNode\* node)

{

if (node == nullptr || (node->getLeftNode() == nullptr && node->getRightNode() == nullptr))

return 0;

int left = calculateHeight(node->getLeftNode());

int right = calculateHeight(node->getRightNode());

return left > right ? left + 1 : right + 1;

}

void treeWalk(TreeNode\* node)

{

if (node == nullptr)

return;

findNodesWhereSubsEqual(node);

treeWalk(node->getLeftNode());

treeWalk(node->getRightNode());

}

void findNodesWhereSubsEqual(TreeNode\* node)

{

TreeNode\* rootBuffer = node;

TreeNode\* nodeBuffer = node;

string direction = "left";

static int countNeededNodes = 0;

countNodes(node, rootBuffer, direction);

this->leftHeight = calculateHeight(node->getLeftNode());

this->rightHeight = calculateHeight(node->getRightNode());

// Вывод

if (this->countOfLeftNodes == this->countOfRightNodes && this->countOfLeftNodes != 0)

{

cout << "Вершина: " << rootBuffer->getTreeNodeData() << " имеет одинаковое количество потомков" << endl << endl;

this->neededNodes++;

}

else if (this->leftHeight == this->rightHeight && this->leftHeight != 0 && this->rightHeight != 0)

{

cout << "Вершина: " << nodeBuffer->getTreeNodeData() << " имеет равные величины высот правого и левого поддеревьев\n"

<< "Высота левого поддерева: " << this->leftHeight + 1 << endl

<< "Высота правого поддерева: " << this->rightHeight + 1 << endl << endl;

this->neededNodes++;

}

this->countOfLeftNodes = this->countOfRightNodes = 0;

this->leftHeight = this->rightHeight = 0;

}

};