|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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**

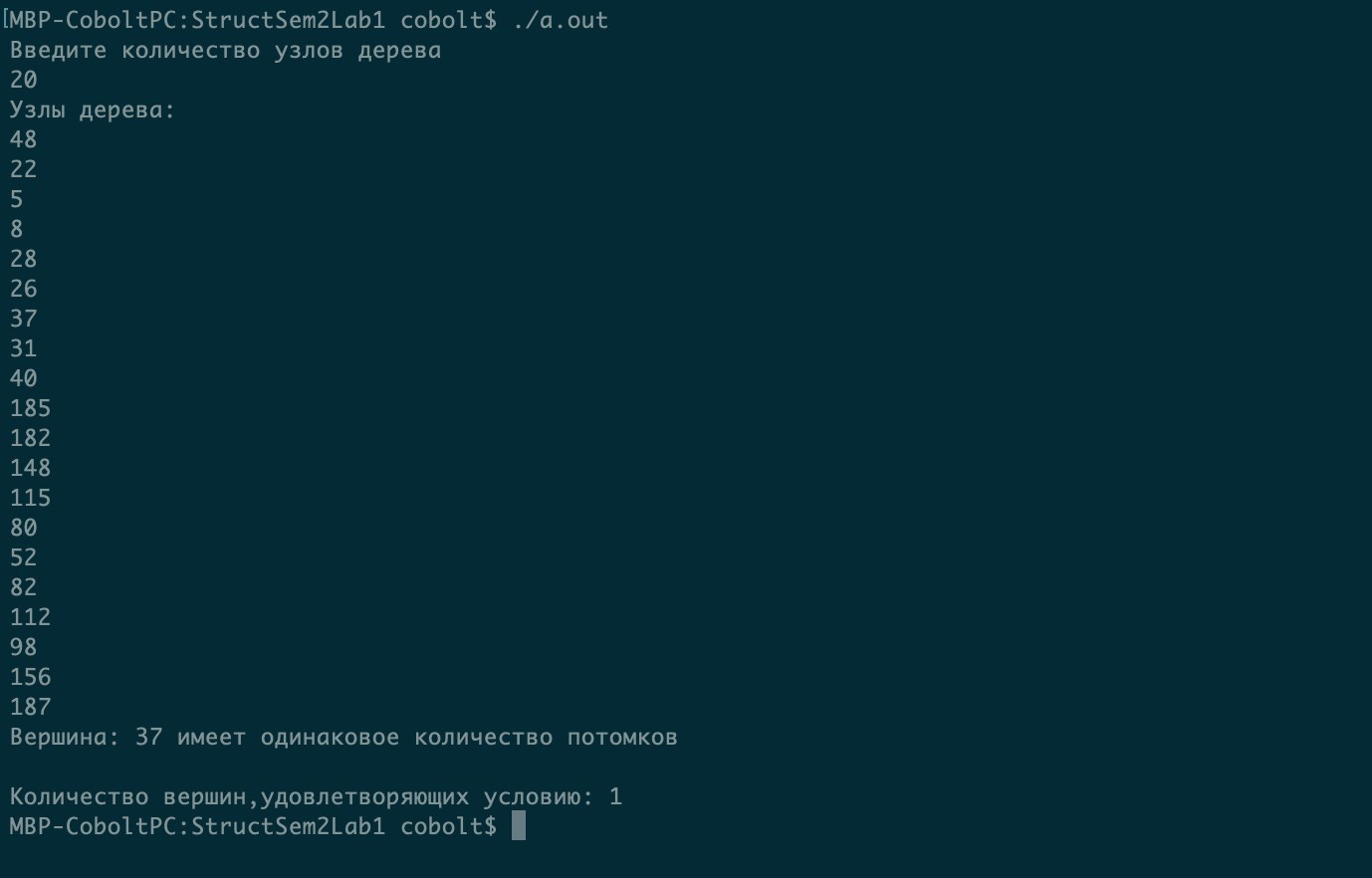
Задание 19:

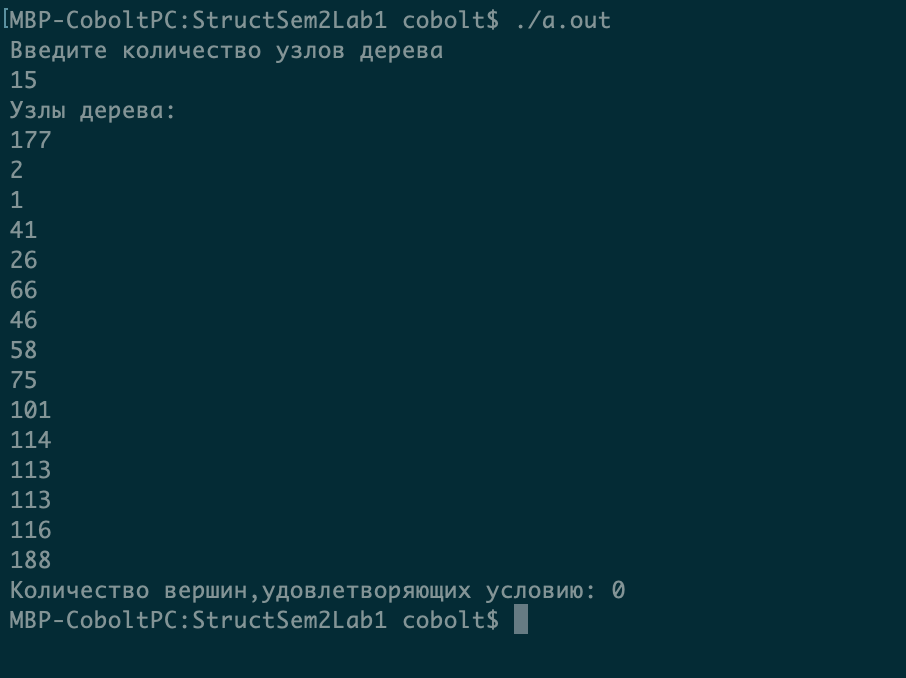
Вычислите количество вершин, у которых равны или высоты поддеревьев, или количество потомков в правом и левом поддеревьях.

**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 <locale.h>

#include <time.h>

#include <iomanip>

#include <vector>

using namespace std;

void printGraph(int\*\*& graph, const int& countOfCities)

{

cout << "№|";

for (int row = 0; row < countOfCities; row++)

cout << setw(5u) << row + 1;

cout << endl;

for (int row = 0; row < countOfCities; row++)

{

cout << row + 1 << "|";

for (int col = 0; col < countOfCities; col++)

cout << setw(5u) << graph[row][col];

cout << endl;

}

}

// Генерация значений графа

void regenGraph(int\*\*& graph, const int& countOfCities)

{

// Выделение памяти для графа

graph = new int\* [countOfCities];

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

graph[i] = new int[countOfCities];

// Заполнение нулями

for (int row = 0; row < countOfCities; row++)

for (int col = 0; col < countOfCities; col++)

graph[row][col] = 0;

// Генерация значений

for (int row = 0; row < countOfCities; row++)

{

graph[row][row] = 0;

if (row == 0)

{

graph[row][countOfCities - 1] = graph[countOfCities - 1][row] = 35 + rand() % 80;

graph[row][row + 1] = graph[row + 1][row] = 20 + rand() % 70;

}

if (row != countOfCities - 1)

graph[row][row + 1] = graph[row + 1][row] = 20 + rand() % 70;

}

}

// Алгоримт Дейкстры

auto Dijkstra(int\*\* &graph, const int &countOfCities, const int &start)

{

int distanceIndex, u, m = start;

int\* distanceArr = new int[countOfCities];

int\* from = new int[countOfCities];

bool\* visitedArr = new bool[countOfCities];

vector<int> result;

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

{

distanceArr[i] = INT\_MAX;

visitedArr[i] = false;

from[i] = -1;

}

distanceArr[start] = 0;

for (int count = 0; count < countOfCities - 1; count++)

{

int min = INT\_MAX;

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

if (!visitedArr[i] && distanceArr[i] <= min)

{

min = distanceArr[i];

distanceIndex = i;

}

u = distanceIndex;

visitedArr[u] = true;

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

if (!visitedArr[i] && graph[u][i] && distanceArr[u] != INT\_MAX && distanceArr[u] + graph[u][i] < distanceArr[i])

{

from[i] = u;

distanceArr[i] = distanceArr[u] + graph[u][i];

}

}

cout << "Стоимость пути из начальной вершины до остальных:\n";

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

if (distanceArr[i] != INT\_MAX) {

cout << "Из узла " << m+1 << " в узел " << i + 1 << " = " << distanceArr[i] << endl;

result.push\_back(distanceArr[i]);

}

else

cout << "Из узла " << m+1 << " в узел " << i + 1 << " маршрут недоступен" << endl;

return result;

}

// Построить дороги между городами

void buildRoads(int\*\*& graph, const int& countOfCities, vector<int>& distances, const int& selectedCity)

{

if (countOfCities >= 5)

{

int buildedRoads = 0;

for (int col = 0; col < countOfCities; col++)

{

if (graph[selectedCity][col] == 0 && col != selectedCity) {

graph[selectedCity][col] = graph[col][selectedCity] = distances[col] / 2;

buildedRoads++;

}

if (buildedRoads == 3)

break;

}

cout << "Расстояния между городами с учетом новых дорог\n";

printGraph(graph, countOfCities);

distances = Dijkstra(graph, countOfCities, selectedCity);

}

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(nullptr));

int\*\* graph = 0;

int countOfCities;

int selectedCity;

cout << "Введите колиечество городов:\n";

cin >> countOfCities;

cin.ignore();

regenGraph(graph,countOfCities);

printGraph(graph,countOfCities);

cout << "Введите город:\n";

cin >> selectedCity;

cin.ignore();

auto distances = Dijkstra(graph, countOfCities,selectedCity-1);

buildRoads(graph,countOfCities,distances,selectedCity-1);

return 0;

}