Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

Кафедра программного

обеспечения и администрирования

информационных систем

Направление подготовки

математическое обеспечение и

администрирование

информационных систем

Форма обучения очная

**Отчет**

**по лабораторной работе №10**

«Деревья»

Выполнил:

студент группы 213 Тихонов Е.Е.

Проверил:

профессор кафедры ПОиАИС Кудинов В.А.

Курск, 2022

**Цель работы**: Изучить принцип хранения множества значений в дереве. Научиться программно, реализовывать основные операции над деревьями.

**Задание**

*Задача 1*. Определите, является ли заданный граф бинарным деревом.

*Задача 2*. Найдите количество ребер в остовом дереве заданного графа, полученном обходом графа в глубину.

**Блок схемы алгоритмов для решения задач**

Блок схема алгоритма поиска бинарного дерева при помощи обхода в глубину для решения задачи 1 представлена на рисунке 1.

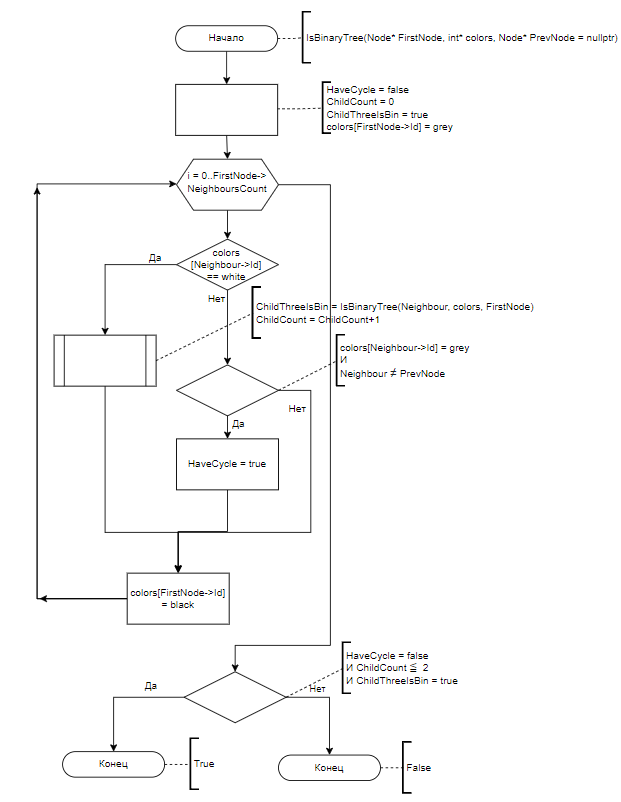


Рисунок 1 - Блок схема алгоритма для поиска бинарного дерева

Блок схема алгоритма поиска количество ребер в остовом дереве заданного графа при помощи обхода в ширину для решения задачи 2 представлена на рисунке 2.

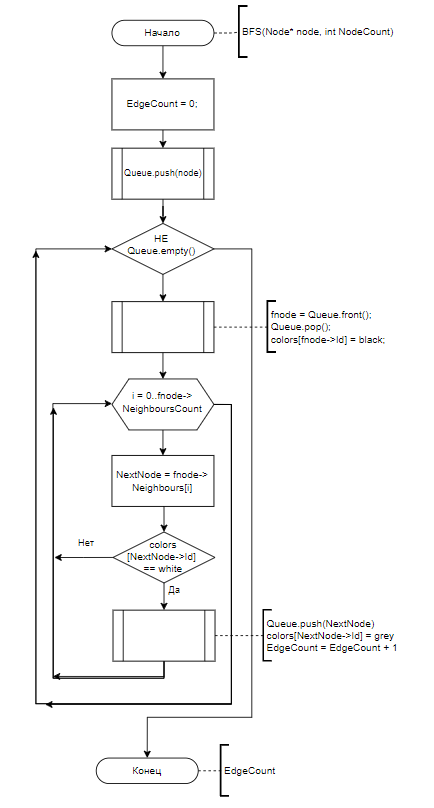


Рисунок 2 - Блок схема алгоритма для поиска количество ребер в остовом дереве

**Код программы**

#include <iostream>

#include <string>

#include <queue>

using namespace std;

class Node

{

public:

int Id; //номер вершины

int Data; //ключ вершины

Node\*\* Neighbours; //соседние(смежные) вершины

int NeighboursCount;

int MaxNeighboursCount;

Node(int Id, int Data = 0, int MaxNeighboursCount = 10)

{

this->Id = Id;

this->Data = Data = 0;

this->Neighbours = new Node \* [MaxNeighboursCount];

this->NeighboursCount = 0;

this->MaxNeighboursCount = MaxNeighboursCount;

}

void AddNeighbor(Node\* Neighbor, bool oriented = false) // добавить смежную вершину

{

if (NeighboursCount == MaxNeighboursCount)

{

cout << "Для вершины " << this->Id << " достигнут лимит соседей" << endl;

}

else

{

Neighbours[NeighboursCount] = Neighbor;

NeighboursCount++;

if (oriented == false)

{

Neighbor->AddNeighbor(this, true);

}

}

}

};

enum Color { white, grey, black };

bool IsBinaryTree(Node\* FirstNode, int\* colors, Node\* PrevNode = nullptr)

{

bool HaveCycle = false;

int ChildCount = 0;

bool ChildThreeIsBin = true;

colors[FirstNode->Id] = grey;

for (int i = 0; i < FirstNode->NeighboursCount; i++)

{

Node\* Neighbour = FirstNode->Neighbours[i];

if (colors[Neighbour->Id] == white)

{

ChildThreeIsBin = IsBinaryTree(Neighbour, colors, FirstNode);

ChildCount++;

}

else if (colors[Neighbour->Id] == grey && Neighbour != PrevNode)

{

HaveCycle = true;

}

colors[FirstNode->Id] = black;

}

if (HaveCycle == false && ChildCount <=2 && ChildThreeIsBin == true)

{

return true;

}

return false;

}

int BFS(Node\* node, int NodeCount)

{

int EdgeCount = 0;

int\* colors = new int[NodeCount];

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

{

colors[i] = white;

}

queue<Node\*> Queue;

Queue.push(node);

while (!Queue.empty())

{

Node\* fnode = Queue.front();

Queue.pop();

colors[fnode->Id] = black;

for (int i = 0; i < fnode->NeighboursCount; i++)

{

Node\* NextNode = fnode->Neighbours[i];

if (colors[NextNode->Id] == white)

{

Queue.push(NextNode);

colors[NextNode->Id] = grey;

EdgeCount++;

}

}

}

return EdgeCount;

}

int main()

{

setlocale(LC\_ALL, "Russian");

Node\* v1 = new Node(0);

Node\* v2 = new Node(1);

Node\* v3 = new Node(2);

Node\* v4 = new Node(3);

//2

//граф имеющий цикл(квадрат)

v1->AddNeighbor(v2);

v2->AddNeighbor(v3);

v3->AddNeighbor(v4);

v4->AddNeighbor(v1);

int\* colors = new int[4]{ white, white, white, white };

if (IsBinaryTree(v1, colors))

{

cout << "граф является бинарным деревом" << endl;

}

else

{

cout << "граф не является бинарным деревом" << endl;

}

//граф не имеющий цикл(ломаная)

Node\* n1 = new Node(0);

Node\* n2 = new Node(1);

Node\* n3 = new Node(2);

Node\* n4 = new Node(3);

Node\* n5 = new Node(4);

n1->AddNeighbor(n2);

n1->AddNeighbor(n3);

n2->AddNeighbor(n4);

n2->AddNeighbor(n5);

colors = new int[5]{ white, white, white, white, white };

if (IsBinaryTree(n1, colors))

{

cout << "граф является бинарным деревом" << endl;

}

else

{

cout << "граф не является бинарным деревом" << endl;

}

Node\* m1 = new Node(0);

Node\* m2 = new Node(1);

Node\* m3 = new Node(2);

Node\* m4 = new Node(3);

Node\* m5 = new Node(4);

m1->AddNeighbor(m2);

m2->AddNeighbor(m3);

m2->AddNeighbor(m4);

m2->AddNeighbor(m5);

colors = new int[5]{ white, white, white, white, white };

if (IsBinaryTree(m1, colors))

{

cout << "граф является бинарным деревом" << endl;

}

else

{

cout << "граф не является бинарным деревом" << endl;

}

//2

cout << "граф V(квадрат) с 4 ребрами имеет остовное дерево с "<< BFS(v1, 4) << " ребрами" << endl;

cout << "граф N(бинарное дерево) с 4 ребрами имеет остовное дерево с " << BFS(n1, 5) << " ребрами" << endl;

cout << "граф M(дерево) с 4 ребрами имеет остовное дерево с " << BFS(m1, 5) << " ребрами" << endl;

//изменим последнее дерево

m3->AddNeighbor(m4);

m4->AddNeighbor(m5);

cout << "граф M(измененный) с 6 ребрами имеет остовное дерево с " << BFS(m1, 5) << " ребрами" << endl;

}

**Тест программы**

Тест программы для решения задачи 1 и 2 представлен на рисунке 3.

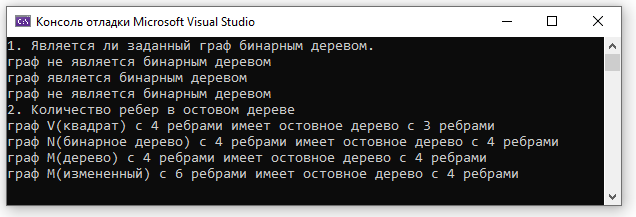


Рисунок 3 - Тест программы