Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

**ОТЧЕТ**

по учебной практике

на кафедре Прикладной Математики и Кибернетики

Вариант 5

Выполнил:

студент гр. ИП-71 \_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_Мартасов И.О.

«24» мая 2019г.

Руководитель практики

доцент каф. ПМиК \_\_\_\_\_\_\_\_\_\_\_\_\_/Разинкина Т.Э./

«24» мая 2019г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2019г.

**Содержание**

1. Условие задачи …...............................................................................................3

2. Описание алгоритма............................................................................................4

3. Код программы……...………………………. ...................................................5

4. Результаты .......................…................................................................................9

**Условие задачи**

Оценка «удовлетворительно»:

Граф задан его матрицей смежности. Требуется определить количество компонент связности этого графа. При этом должны быть конкретно перечислены вершины, входящие в каждую компоненту связности. Выбор алгоритма поиска компонент связности – произвольный. Например, приветствуется использование одного из видов обхода (поиск в глубину или поиск в ширину). Вход программы: число вершин графа и матрица смежности. Выход: разбиение множества вершин на подмножества, соответствующие компонентам связности.

Оценка «хорошо»:

В дополнение к предыдущему, пользователю должна быть предоставлена возможность редактировать исходную матрицу, т.е. изменять исходный граф без выхода из программы. Предусмотреть также возможность изменения количества вершин.

Оценка «отлично»:

Заданный граф рассматривать как ориентированный. Выполнять поиск компонент сильной связности.

**Описание алгоритма**

При запуске программы пользователю предлагается ввести количество вершин для построения матрицы смежности. Максимальное возможное количество вершин - 50. Инициализированная матрица всегда заполняется нулями.

После этого, на экране рисуется сама матрица и меню, состоящее из пунктов:

1 – Изменение количества вершин

2 – Вывод компонент связности

3 – Изменение элементов матрицы

4 – Выход

Пункт 1 позволяет пользователю изменить количество вершин в матрице. Если пользователь изменял матрицу, новая матрица сбросит все добавленные ранее ребра.

Пункт 2 считает количество компонент связности графа и выводит разбиение вершин на компоненты. Поиск компонент связности был сделан при помощи поиска в глубину. Сначала проверяется, была ли посещена вершина. Если нет, то количество компонент связности увеличивается, эта вершина помечается как посещенная и от неё происходит поиск в глубину. Для разбиения множества используется массив cc, в который записывается номер компоненты, к которой принадлежит вершина. Обход графа всегда начинается с первой вершины. По окончанию поиска на экран выводится количество компонент связности и разбиение вершин на компоненты, имеющее вид:

“i вершина - i компонента”.

Пункт 3 позволяет пользователю изменять исходную матрицу путём выбора строки и столбца, в которой расположена вершина. Номера строк и столбцов начинаются с единицы.

Пункт 4 закрывает программу.

**Код программы**

#include <iostream>

using namespace std;

void DFS(int \*\* graph, int n, bool \* visited, int comps\_col, int \* cc, int ver)

{

visited[ver] = true;

cc[ver] = comps\_col;

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

{

if ((graph[ver][i]) && (!visited[i]))

{

DFS(graph, n, visited, comps\_col, cc, i);

}

}

}

void solve(int \*\* graph, int n, int \* cc, bool \* visited, int comps\_col)

{

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

{

visited[i] = false;

}

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

{

if(!visited[i])

{

comps\_col++;

DFS(graph, n, visited, comps\_col, cc, i);

}

}

cout << "Количество компонент связности - " << comps\_col << endl;

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

{

cout << i + 1 << " вершина - " << cc[i] << " компонента " << endl;

}

}

int \*\* fillMatrix(int n)

{

int \*\* graph = new int \* [n];

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

{

graph[i] = new int[n];

for (int j = 0; j < n; j++)

{

graph[i][j] = 0;

}

}

return graph;

}

void printMatrix(int \*\* graph, int n)

{

cout << "Матрица смежности графа: " <<endl;

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

{

for (int j = 0; j < n; j++)

{

cout << graph[i][j] << " ";

}

cout << endl;

}

}

void fixmatrix(int n, int \*\* graph)

{

int z1, z2;

m3:cout << "Строка - ";

cin >> z1;

cout << "Столбец - ";

cin >> z2;

if((z1 < 1) || (z1 > n) || (z2 < 1) || (z2 > n))

{

cout << "Введите данные заново" << endl;

goto m3;

}

if(z1 == z2)

{

cout << "Петли недопустимы!Введите данные заново" << endl;

goto m3;

}

z1--;

z2--;

graph[z1][z2] = 1 - graph[z1][z2];

graph[z2][z1] = graph[z1][z2];

}

int main()

{

int comps\_col = 0;

int \*\* graph;

bool \* visited;

int \* cc;

int n, g, z;

m:cout << "Количество вершин - ";

cin >> n;

if((n <= 1) || (n > 50))

{

cout << "Неверные данные!" << endl;

goto m;

}

visited = new bool[n];

cc = new int[n];

graph = fillMatrix(n);

while(g != 4)

{

system("CLS");

printMatrix(graph, n);

cout << "1 - Изменение количества вершин" << endl;

cout << "2 - Вывод компонент связности" << endl;

cout << "3 - Изменение элементов матрицы" << endl;

cout << "4 - Выход" << endl;

cin >> g;

switch(g)

{

case 1:

goto m;

case 2:

solve(graph, n, cc, visited, comps\_col);

system("PAUSE");

break;

case 3:

fixmatrix(n, graph);

break;

default:

break;

}

}

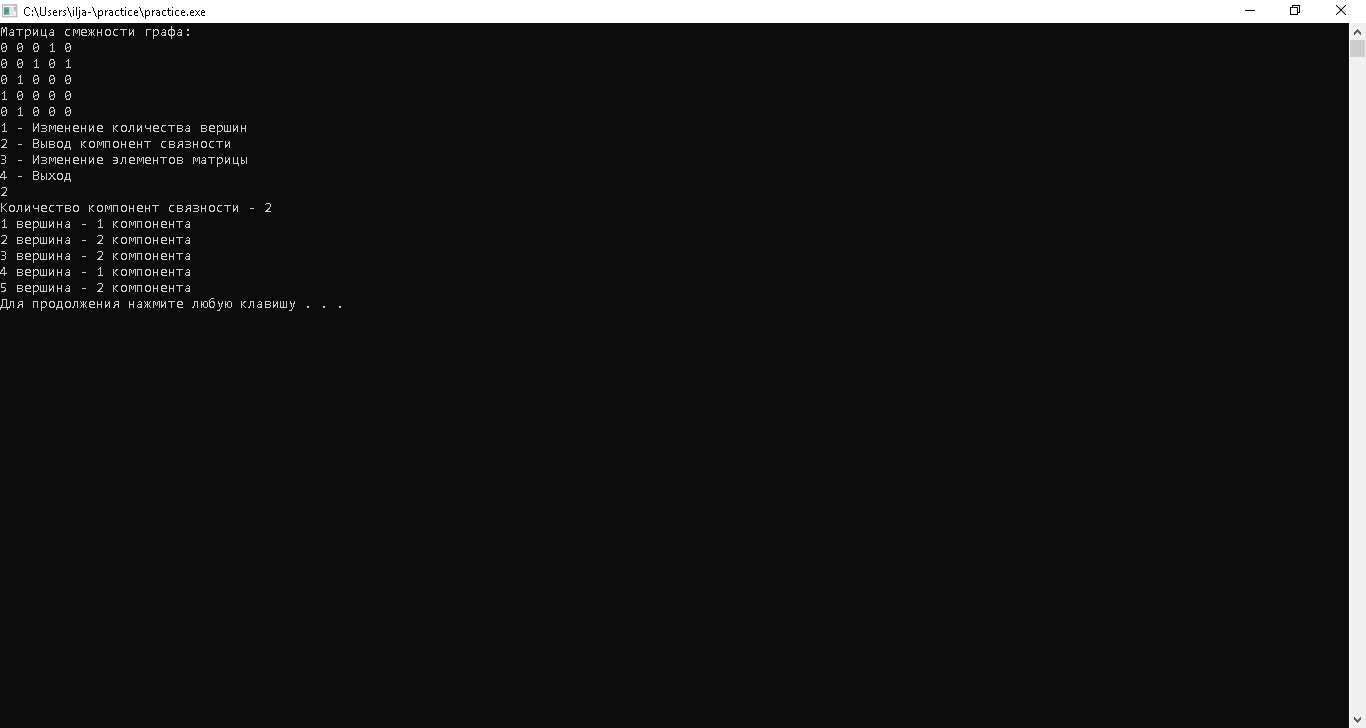
return 0;

**Результаты**

**1.Матрица размером 5x5**

**2.Процесс изменения матрицы**

**3.Измененная матрица**

**4.Вывод количества компонент связности и разбиение вершин**