Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Логика и основа алгоритмизации в инженерных задачах»

на тему

«Реализация алгоритма выделения компонент

связности орграфа, используя поиск в глубину»

Выполнил студент группы 21ВВ2:

Волдырев В. С.

Приняли: к.т.н. доцент Юрова О.В.

д.т.н. профессор Митрохин М.А.

Пенза 2022

**Содержание**

Оглавление

[Реферат 3](#_Toc123031186)

[Введение 4](#_Toc123031187)

[1. Постановка задачи 5](#_Toc123031188)

[2. Теоретическая часть задания 6](#_Toc123031189)

[3. Описание алгоритма программы 8](#_Toc123031190)

[4. Описание программы 10](#_Toc123031191)

[5. Тестирование 18](#_Toc123031192)

[6. Ручной расчет задачи 21](#_Toc123031193)

[Заключение 22](#_Toc123031194)

[Список литературы 23](#_Toc123031195)

[Приложение А 24](#_Toc123031196)

# Реферат

Отчет 26 стр., 7 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, ДОСТИЖИМОСТЬ, ПОИСК В ШИРИНУ,

КОМПОНЕНТА.

Цель исследования – разработка программы, способная выделять компоненты связности орграфа, использую алгоритм поиска в глубину.

В работе рассмотрены правила поиска в ширину, на основе которых находится компонента связности орграфа. Установлено, что с помощью данного алгоритма можно выделить компоненту связности подграфа орграф, которого несвязен, слабо связан или сильно связан.

# Введение

Алгоритм поиска (или обхода) в глубину (англ. depth-firstsearch, DFS) позволяет построить обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины.

Отличие поиска в глубину от поиска в ширину заключается в том, что (в случае неориентированного графа) результатом алгоритма поиска в глубину является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины. Этим он принципиально отличается от поиска в ширину, где одновременно обрабатывается множество вершин, в поиске в глубину в каждый момент исполнения алгоритма обрабатывается только одна вершина.

С другой стороны, поиск в глубину не находит кратчайших путей, зато он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.

Если же граф ориентированный, то поиск в глубину строит дерево путей из начальной вершины во все доступные из нее.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2010, язык программирования – Си.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска в глубину, осуществляющий поиск компоненты сильной связности орграфа.

# 1. Постановка задачи

Требуется разработать программу, которая выделит компоненты связности орграфа, используя алгоритм поиска в глубину.

Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности орграфа, вид орграфа и все компоненты связности орграфа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №21

# 2. Теоретическая часть задания

# Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин X1, X2, ..., Xn. и множеством ребер , соединяющих между собой определенные вершины. Ребра из множества А ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.

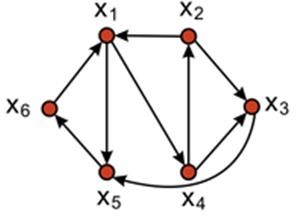


Рисунок 1 – Пример орграфа

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа. Остановимся на одном из двух стандартных методах такого перебора - поиск в глубину.

Пусть задан орграф G= (X, a) и вершина X1, с которой начинается обход. После посещения вершины X1, следующей за ней будет посещена смежная с X1 вершиной. Далее, эта процедура повториться для вершин смежных с вершинами из множества q, за исключением вершины X1, т.к. она уже была посещена. Так, продолжая обходить в глубину, алгоритм обойдет все доступные из X1 вершины орграфа G. Алгоритм прекращает свою работу после обхода всех вершин графа, либо в случае выполнения наличествующего условия.

Орграф называется сильно связным, если любые две его вершины сильно связаны. Две вершины X1 и X2 любого орграфа сильно связаны, если существует ориентированный путь из X1 в X2 и ориентированный путь из X2 в X1. Компонентами сильной связности орграфа называются его максимальные по включению сильно связные подграфы.

Любая вершина орграфа сильно связана сама с собой.

# 

# 3. Описание алгоритма программы

Для реализации алгоритма понадобится матрица (int\*\* G), массив посещенных или не посещенных вершин (int\* numV), также понадобиться очередь, для реализации алгоритма обхода в ширину, и вектор, в который будем добавлять посещенные вершины.

Реализация алгоритма начинается с инициализации матрицы либо случайными значениями, либо значениями, введенными с клавиатуры.

Затем, чтобы выявить компоненты связности орграфа производится обход в ширину обычной и транспонированной матрицы. Если, после обхода, в множестве присутствуют не все заявленные вершины, то поиск начинается с не посещенной ранее вершины. Такой цикл происходит, пока не будут посещены все вершины. После каждого обхода в ширину, программа демонстрирует пользователю, полученную компоненту.

Ниже представлен псевдокод функций: findComponents(), BFS()

**BFS():**

Q = очистка очереди;

Добавление вершины в очередь;

Отметка о посещении вершины;

Добавление вершины в вектор;

Пока (очередь не пуста)

Для (i = 0; i < кол-ва вершин; i++)

Если ((G[v][i] == 1 или G[i][v] == 1) и вершина не посещена)

Добавление в очередь вершины;

Отметка о посещении вершины;

Добавление вершины в вектор;

**findComponents():**

Для (i = 0; i < кол-ва вершин; i++)

Если вершина не посещена

BFS();

Вывод на экран компоненты;

Запись компоненты в файл;

Обнуление посещения вершин;

Очистка вектора;

# 4. Описание программы

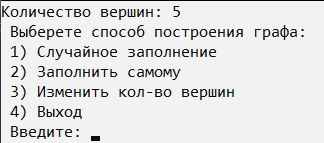
Для написания данной программы использован язык программирования С++. Язык программирования С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

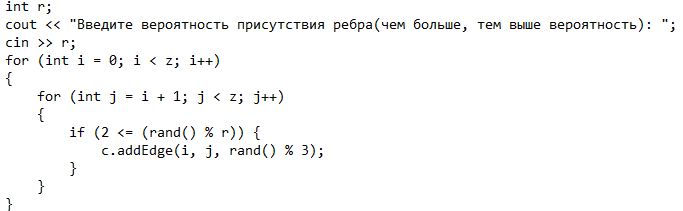
Работа программы начинается с ввода количества вершин.



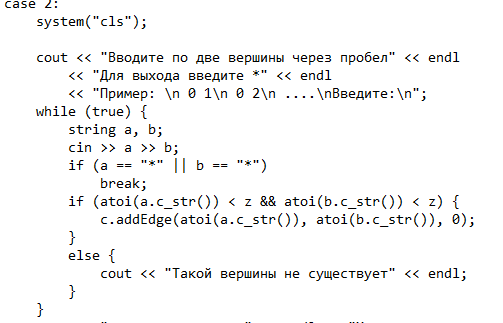
Потом перед пользователем стоит выбор в главном меню.



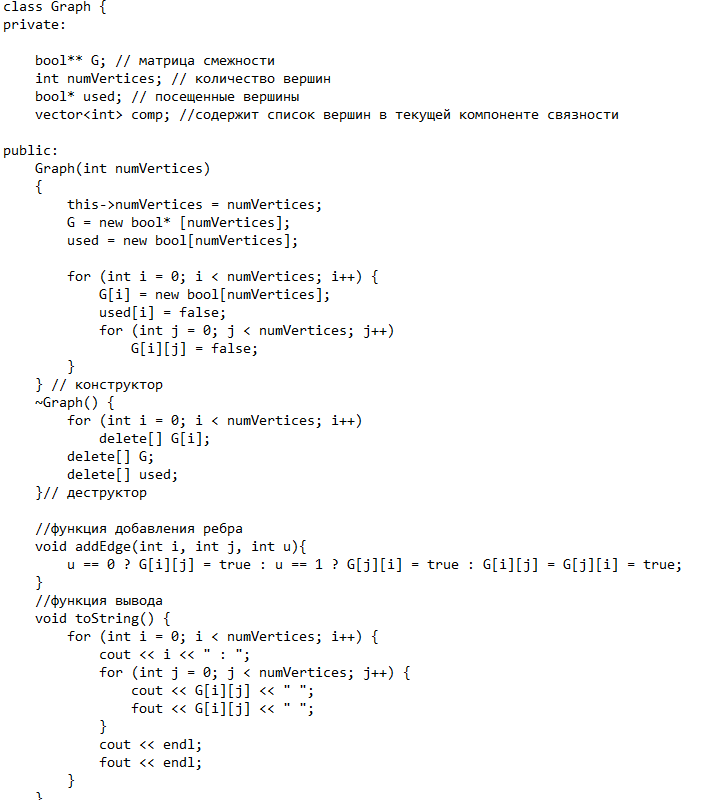
Если пользователь выбрал заполнения матрицы случайным образом ему придётся ввести вероятности присутствия ребра между вершинами.



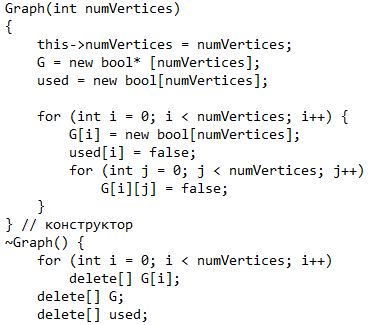
Если же пользователь выбрал заполнения матрицы самому ему придётся вводить поочередную связь между вершинами, пока он самостоятельно не выйдет из этого из цикла.



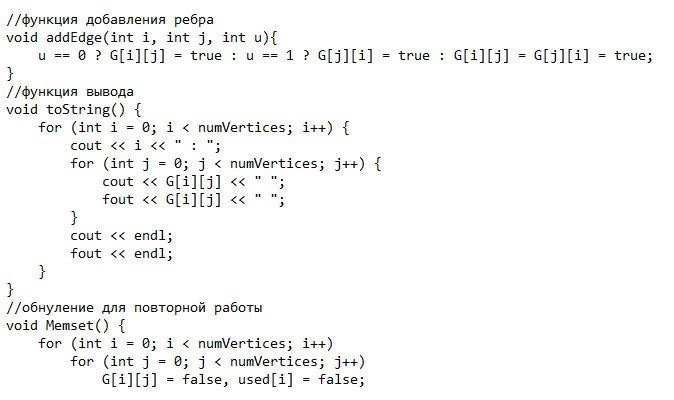
Для программной реализации алгоритма понадобиться класс Graph , который содержит в себе матрицу смежности и всю информацию о нем, а также функции для её вывода или очистки для повторной работы.



Конструктор заботится в программе за выделение динамической памяти матрице G(тип: bool) и присвоению переменной numVertices(тип: int) количества вершин, вводимы пользователем. Деструктор выполняет обратную операцию, очищает выделенную динамическую память по завершению программы.



Функция addEdgeMatrix(int, int) (рисунок 8) нужна для заполнения матрицы смежности о связях между вершинами. Функция toString() (рисунок 8) выполняет задачу вывода матрицы на экран. Функция Memset() (рисунок 8) обнуляет матрицу для повторной работы над ней, чтобы программа при повторном вводе выдавала верные значения.



Ниже на рисунках можно увидеть работу выше описанной программы.

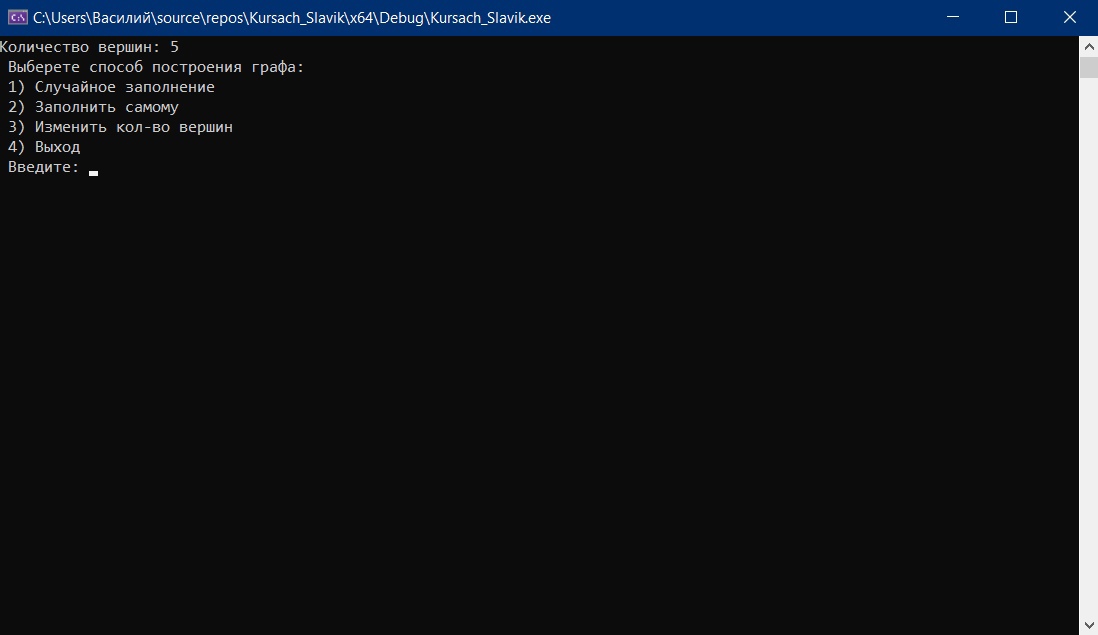


Рисунок 2 – Меню программы

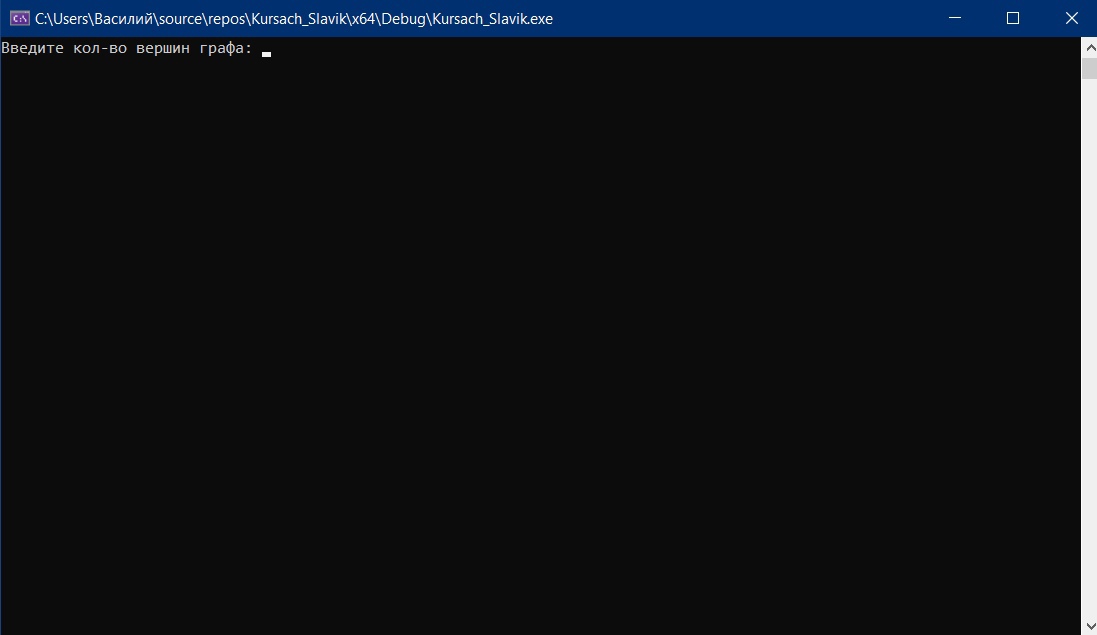


Рисунок 3 - Задание размера матрицы

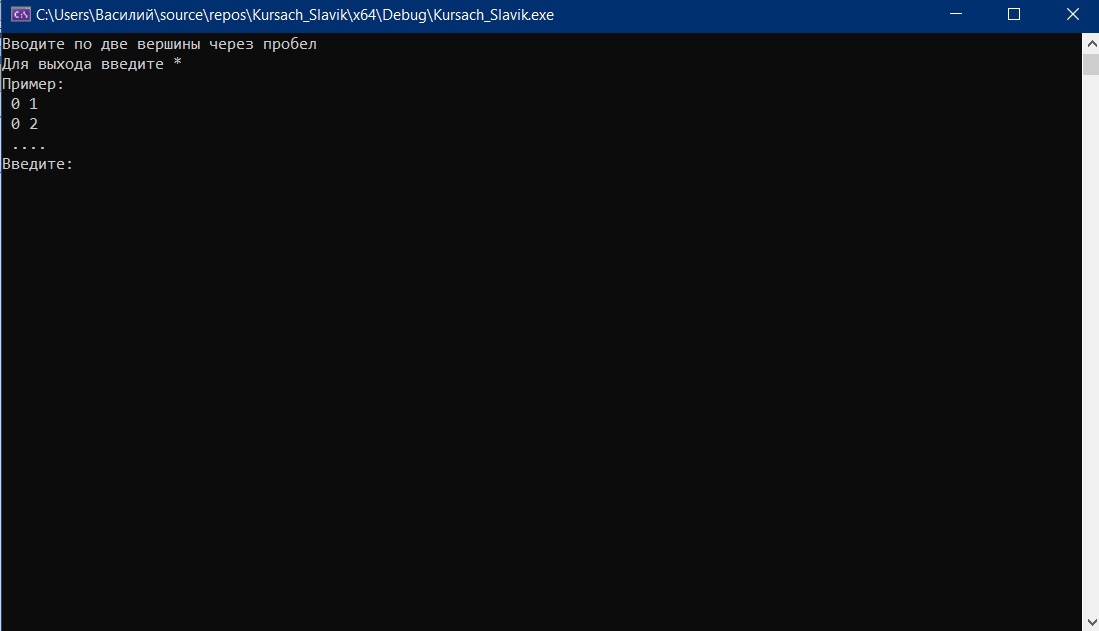


Рисунок 4 – Заполнение матрицы вручную

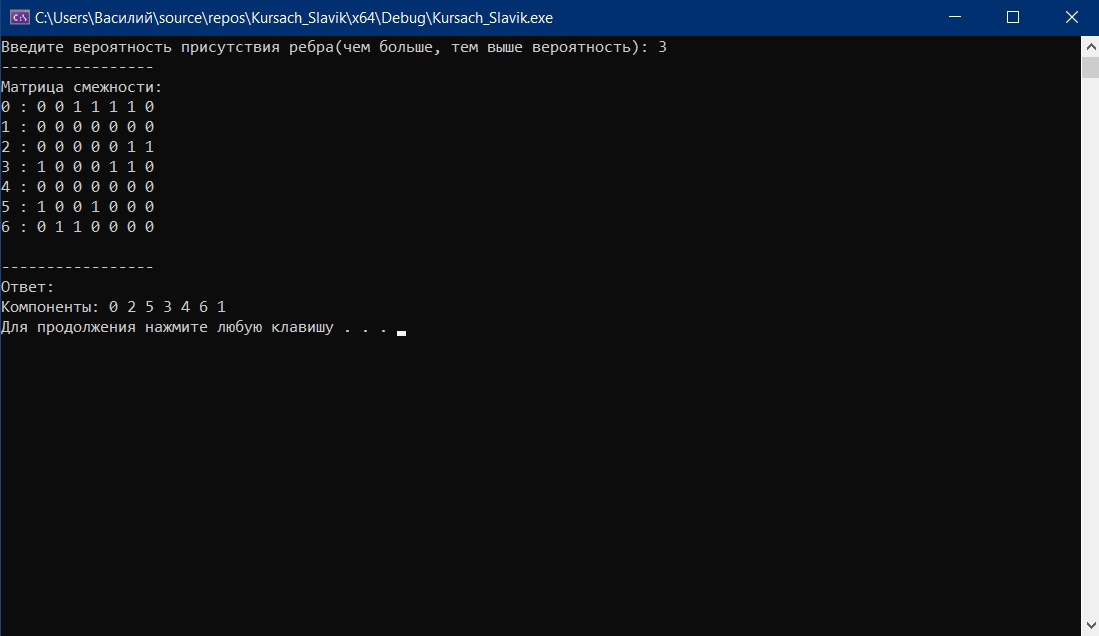


Рисунок 5 – Вывод работы программы на экран

# 5. Тестирование

Среда разработки MicrosoftVisualStudio 2010 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

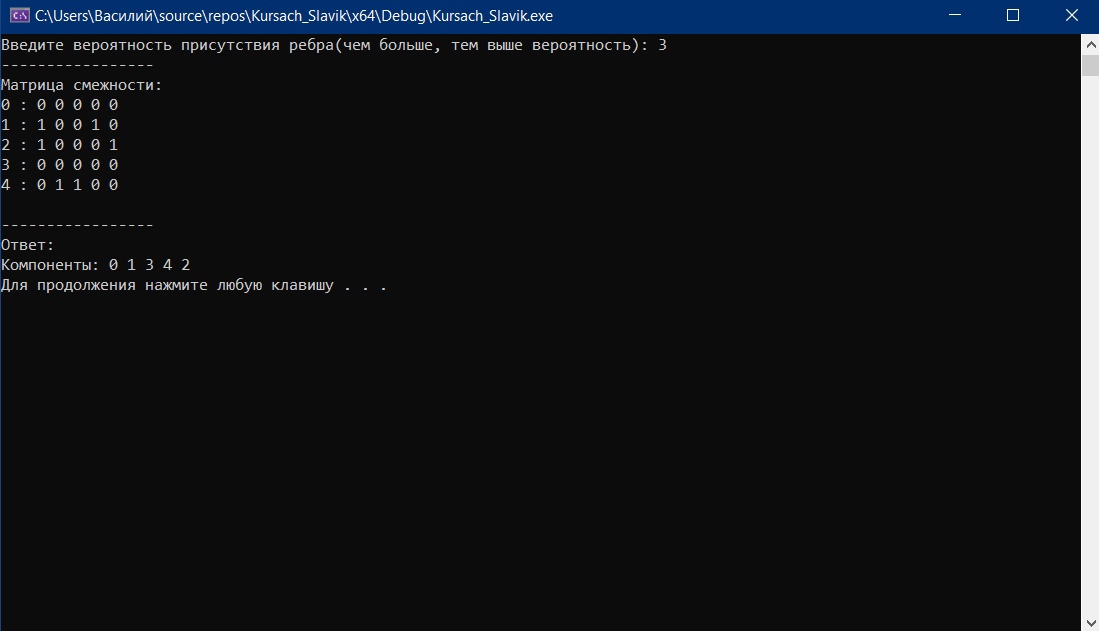
Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывод компонент связности орграфов. 

Рисунок 6 – Тестирование при вводе количество вершин = 5

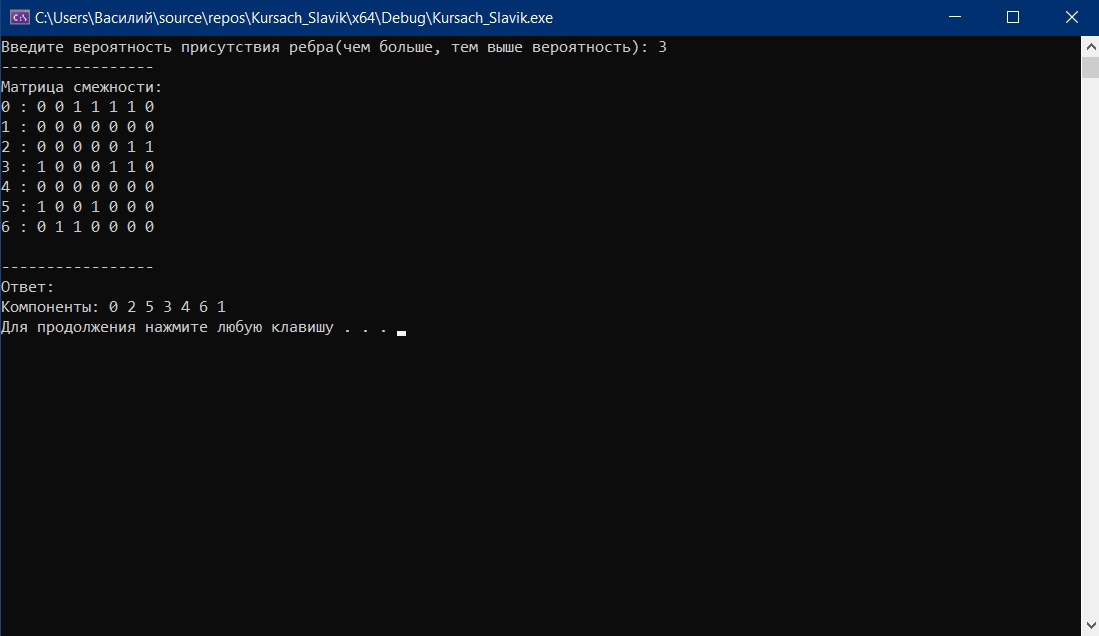


Рисунок 7 - Тестирование при вводе количество вершин = 7

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

# 6. Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 5 вершинами (рисунок 6).

Начинаем обход из 0 вершины в последующие. Проверяем, если есть путь из 0 в другие вершины, то дальше идем. В нашем случае идем в вершину 1 и 3. Аналогично проверяем и в вершине 2. Из вершины 2 идем в 0 и 4. Из вершины 4 идем в 1 и 2. .

В итоге у нас получилось компоненты связности орграфа {01342}.

Таким образом, можно сделать вывод, что программа работает верно.

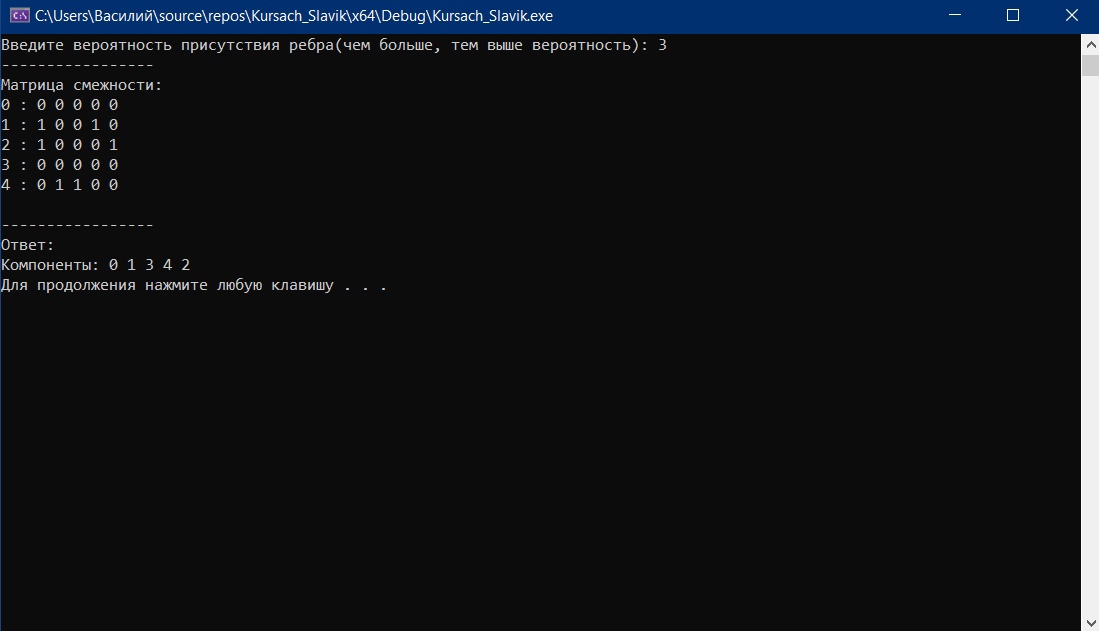


Рисунок 8 – Тестирование работы программы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска в глубину для поиска компонент сильной связности орграфа в Microsoft Visual Studio 2022.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания языка программирования Cи.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

* 1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.
  2. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
  3. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
  4. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
  5. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
  6. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

# Приложение А

**Листинг программы**

#include "stdafx.h"

#include <iostream>

#include <string>

#include <string.h>

#include <vector>

#include <fstream>

#include <algorithm>

#include <time.h>

using namespace std;

ofstream fout("answer.txt");

class Graph {

private:

bool\*\* G; // матрица смежности

int numVertices; // количество вершин

bool\* used; // посещенные вершины

vector<int> comp; //содержит список вершин в текущей компоненте связности

public:

Graph(int numVertices)

{

this->numVertices = numVertices;

G = new bool\* [numVertices];

used = new bool[numVertices];

for (int i = 0; i < numVertices; i++) {

G[i] = new bool[numVertices];

used[i] = false;

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

G[i][j] = false;

}

} // конструктор

~Graph() {

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

delete[] G[i];

delete[] G;

delete[] used;

}// деструктор

//функция добавления ребра

void addEdge(int i, int j, int u){

u == 0 ? G[i][j] = true : u == 1 ? G[j][i] = true : G[i][j] = G[j][i] = true;

}

//функция вывода

void toString() {

for (int i = 0; i < numVertices; i++) {

cout << i << " : ";

for (int j = 0; j < numVertices; j++) {

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

fout << G[i][j] << " ";

}

cout << endl;

fout << endl;

}

}

//обнуление для повторной работы

void Memset() {

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

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

G[i][j] = false, used[i] = false;

}

//обход в глубину

void DFS(int v);

//поиск компонентов связи

void find\_comps();

};

void Graph::DFS(int v) {

used[v] = true;

comp.push\_back(v); // добавление элемента в компонент

for (int i = 0; i < numVertices; ++i) {

if ((G[v][i] != 0 || G[i][v] != 0) && (!used[i]))

DFS(i);

}

}

void Graph::find\_comps() {

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

if (!used[i]) {

comp.clear();

DFS(i);

cout << "Компоненты:";

fout << "Компоненты:";

for(auto it=comp.begin(); it!=comp.end(); it++){

cout << " " << \*it, fout << " " << \*it;

}

//for (int j : comp)

// cout << " " << j, fout << " " << j;

cout << endl; fout << endl;

}

}

int main()

{

srand(time(NULL));

setlocale(0, "");

int z;

l:

system("cls");

cout << "Введите кол-во вершин графа: "; cin >> z;

Graph c(z);

string s;

while (true) {

system("cls");

cout << "Количество вершин: " << z

<< "\n Выберете способ построения графа: \n 1) Случайное заполнение \n"

<< " 2) Заполнить самому \n 3) Изменить кол-во вершин \n 4) Выход\n Введите: ";

cin >> s;

switch (atoi(s.c\_str()))

{

case 1:

system("cls");

int r;

cout << "Введите вероятность присутствия ребра(чем больше, тем выше вероятность): ";

cin >> r;

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

{

for (int j = i + 1; j < z; j++)

{

if (2 <= (rand() % r)) {

c.addEdge(i, j, rand() % 3);

}

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << endl << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

c.find\_comps();

c.Memset();

system("pause");

break;

case 2:

system("cls");

cout << "Вводите по две вершины через пробел" << endl

<< "Для выхода введите \*" << endl

<< "Пример: \n 0 1\n 0 2\n ....\nВведите:\n";

while (true) {

string a, b;

cin >> a >> b;

if (a == "\*" || b == "\*")

break;

if (atoi(a.c\_str()) < z && atoi(b.c\_str()) < z) {

c.addEdge(atoi(a.c\_str()), atoi(b.c\_str()), 0);

}

else {

cout << "Такой вершины не существует" << endl;

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << endl << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

c.find\_comps();

c.Memset();

system("pause");

break;

case 3:

goto l;

break;

case 4:

exit(0);

break;

}

system("cls");

}

fout.close();

c.~Graph();

return 0;

}