

**Содержание**

Реферат……………………………………………………………………...6

[Введение 7](#_Toc91217240)

1. [Постановка задачи 8](#_Toc91217241)

2. [Теоретическая часть задания 9](#_Toc91217242)

3. [Описание алгоритма программы 11](#_Toc91217243)

4. [Описание программы 14](#_Toc91217244)

5. [Тестирование 16](#_Toc91217245)

6. [Ручной расчет задачи 23](#_Toc91217246)

[Заключение 24](#_Toc91217247)

[Список литературы 25](#_Toc91217248)

[Приложение А. Листинг программы. 26](#_Toc91217249)

**Реферат**

Отчет 29 страниц, 12 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, ДОСТИЖИМОСТЬ, ПОИСК В ГЛУБИНУ, КОМПОНЕНТА.

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

В работе рассмотрены правила поиска в глубину, на основе которых находятся все компоненты связности орграфа.

# Введение

# 

Поиск в глубину (англ. Depth-first search, DFS) — один из методов обхода графа. Стратегия поиска в глубину, как и следует из названия, состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину. Если после завершения алгоритма не все вершины были рассмотрены, то необходимо запустить алгоритм от одной из нерассмотренных вершин.В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – С++.

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

# Постановка задачи

# 

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

Исходный орграф в программе должен задаваться матрицей смежности. При старте программы пользователь должен выбрать каким образом будет генерироваться граф: случайно или загружаться из текстового файла. После чего программа должна выполнить поиск в глубину, при помощи которого найти все компоненты связности графа. И вывести исходную матрицу и полученный результат на экран и в файл.

# Теоретическая часть задания

Граф G (рис. 1) задается множеством вершин X1, X2, ..., X6. 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.

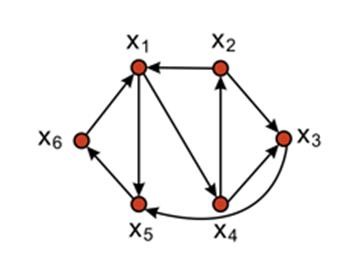


Рисунок 1 – Пример орграфа

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 обход вершин осуществляется по ребрам графа.

Под обходом понимается последовательное посещение (обработка) вершин графа в определённом порядке. Одним из двух часто использующихся способов обхода является обход в глубину, или DFS Его иногда также называют волновым, по аналогии с распространяющейся волной.

Суть DFS достаточно проста. Обход начинается с посещения определённой вершины (для обхода всего графа часто выбирается произвольная вершина). Затем алгоритм посещает все подвершины этой вершины. За ними – подвершины следующей вершины, и так далее.

Понятие компоненты связности вытекает из понятия связности графа. Попросту говоря, компонента связности - часть графа (подграф), являющаяся связной. Формально, компонента связности - набор вершин графа, между любой парой которых существует путь.

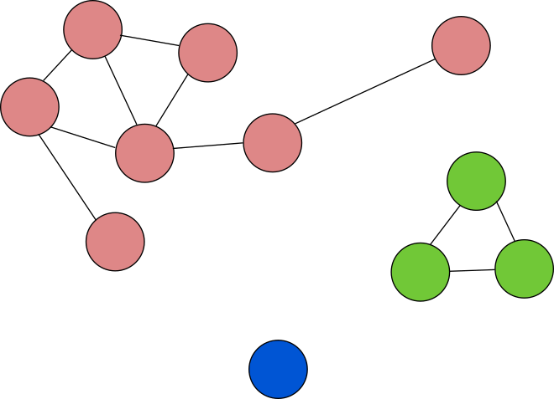


Рисунок 2 – Компоненты связности графа

Граф на рис. 2 содержит три компоненты связности, закрашенные разными цветами. Можно заметить, что даже одна вершина, изолированная от остального графа, составляет компоненту связности.

Общее понятие связности распространяется только на неориентированные графы. Для описания ориентированных графов используются понятия сильной и слабой связности.

Для поиска компонент связности используется обычный DFS практически без модификаций (можно использовать и BFS). При запуске обхода из одной вершины, он гарантированно посетит все вершины, до которых возможно добраться, то есть, всю компоненту связности, к которой принадлежит начальная вершина. Для нахождения всех компонент просто попытаемся запустить обход из каждой вершины по очереди, если мы ещё не обошли её компоненту ранее.

# Описание алгоритма программы

Для реализации программной реализации алгоритма нам понадобится три массива и одна очередь: двумерный массив типа int, G[n][n], содержащий матрицу смежности размера n, графа, вектор типа инт comp - содержащий компоненты связности графа, массив типа bool, used, размера n, содержащий информацию об использованных вершинах.

Алгоритм представляет собой цикл, который проверяет каждую вершину и делает обход в глубину для каждой не посещенной ранее вершины. В нашем случае он проверит первую вершину, и так как это первый запуск и ни одна вершина еще не была посещена, начнет он именно с нее. Вершина пометится посещенной в массиве used, далее по алгоритму программа перейдет к связанной с предыдущей или же если таковой нет, то к следующей вершине. Каждая посещенная вершина так же пометится true в массиве used, и параллельно с этим алгоритм занесет номера посещенных вершин в вектор компонент связности comp.

После того как в массиве used не остается не посещенных вершин, алгоритм закончит свою работу.

Ниже представлен псевдокод функций find\_comps и dfs.

Вход: G – матрица смежности графа, n – количество вершин графа.

Выход: номера вершин, составляющих компоненты связности графа G, на экране.

Функция find\_comps:

1. создать массив истинности used размера n

2. начало цикла

3. для i = 0, пока i < n, выполнять i++

4. used[n] = false

5. конец цикла

6. начало цикла

7. для i = 0, пока i < n, выполнять i++

8. если used[i] == false

9. начало условия

10. очистить comp

11. выполнить ф-ию dfs(G,n,i,used)

12. начало цикла

13. для size\_t k=0, пока k < comp.size(), выполнять k++

14. если k<1 и comp.size()>1

15. начало условия

16. вывести на экран comp[k]+1 с пробелом

17. записать в файл comp[k]+1 с пробелом

18. конец условия

19. если k>=1 и comp.size()>1

20. начало условия

21. вывести на экран comp[k]+1 с 🡪

22. записать в файл comp[k]+1 с 🡪

23. конец условия

24. конец цикла

25. перейти к следующей строке на экране

26. перейти к следующей строке в файле

27. конец условия

28. конец цикла

Функция dfs:

1. пометить used[v] как истинное
2. поместить v в comp
3. начало цикла

4. для i = 0, пока i < n, выполнять i++

5. если G[v][i] равно 1 и used[i] не истинно

6. начало условия

7. выполнить ф-ию dfs(G, n, i, used)

8. конец условия

9.конец цикла

Полный код находится в Приложении А.

# Описание программы

Для написания данной программы использован язык программирования C++. Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из множества функций: main, DFS\_No\_Rec, Init\_Matrix, Out\_Matrix, Input\_Matrix Delete\_Matrix, find\_comps.

Работа программы начинается с запроса на ввод пользователем количества вершин графа (размера матрицы смежности). Далее программа предоставляет пользователю выбор: сгенерировать матрицу выбранного размера случайным образом или загрузить значения матрицы того же размера из заранее предоставленного файла «input.txt» содержимое файла показано на рис. 3. После осуществления выбора, программа очищает экран, чтобы пользователю не мешали предыдущие надписи, не относящиеся к результатам работы. Затем выводит на экран матрицу смежности графа, сгенерированную в соответствии с выбором пользователя, и выполняет алгоритм поиска и вывода компонент связности в данном графе (рис. 8, 9).

Ниже предоставлен код главной функции программы (полный код программы предоставлен в прил. А)

void main()

{

setlocale(LC\_ALL, "Rus");

bool check = false;

int n;

cout << " Введите количество вершин в графе: ";

cin >> n;

int\*\* G = new int\* [n];//граф

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

{

G[i] = new int[n];

}

while (check == false)

{

cout << "\n Выберите действие: \n 1) Сгенерировать граф размера : " << n << ", псевдослучайным образом;\n 2) Выгрузить граф из файла;\n ";

int z = \_getch() - 48;

switch (z)

{

case 1:

cout << "1\n\n";

Init\_Matrix(G, n);

check = true;

break;

case 2:

cout << "2\n\n";

Input\_Matrix(G, n);

check = true;

break;

default:

cout << "\n Ошибка выбора. Пожалуйста, введите корректный номер варианта! \n\n";

break;

}

}

system("cls");

fout.open("output.txt", ofstream::app);

cout << " Матрица смежности графа: \n";

fout << " Матрица смежности графа: \n";

Out\_Matrix(G, n);

cout << " Компоненты связности: \n";

fout << " Компоненты связности: \n";

find\_comps(G, n);

fout << "\n=========================\n\n";

fout.close();

Delete\_Matrix(G, n);

system("pause>>void");

}

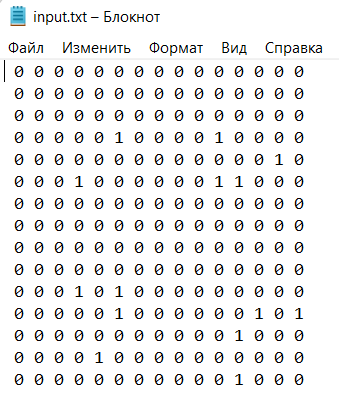


Рисунок 3 – Содержимое "input.txt"

# Тестирование

Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы. Тестирование осуществлялось по мере создания программы, при добавлении новых опций и закончилось итоговым тестом, по окончанию написания кода. В результате совокупности всех тестов было выявлено и устранено множество проблем.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и разного способа генерации значений матрицы смежности (случайная генерация или загрузка матрицы из файла.).

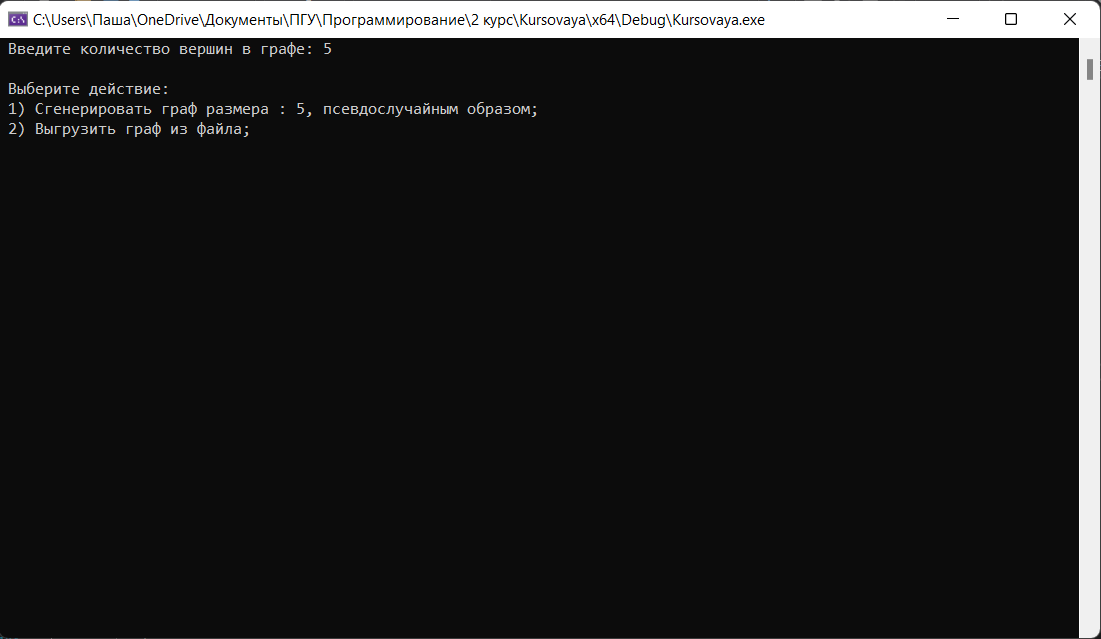


Рисунок 4 – Пользовательский интерфейс

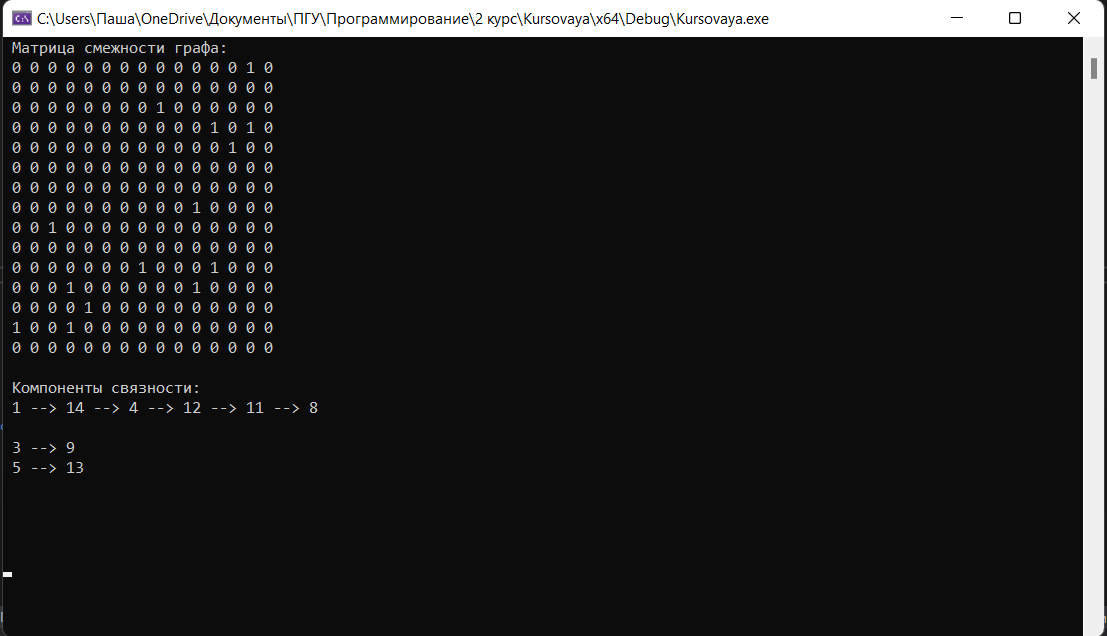


Рисунок 5 – Случайная генерация матрицы размера: 15

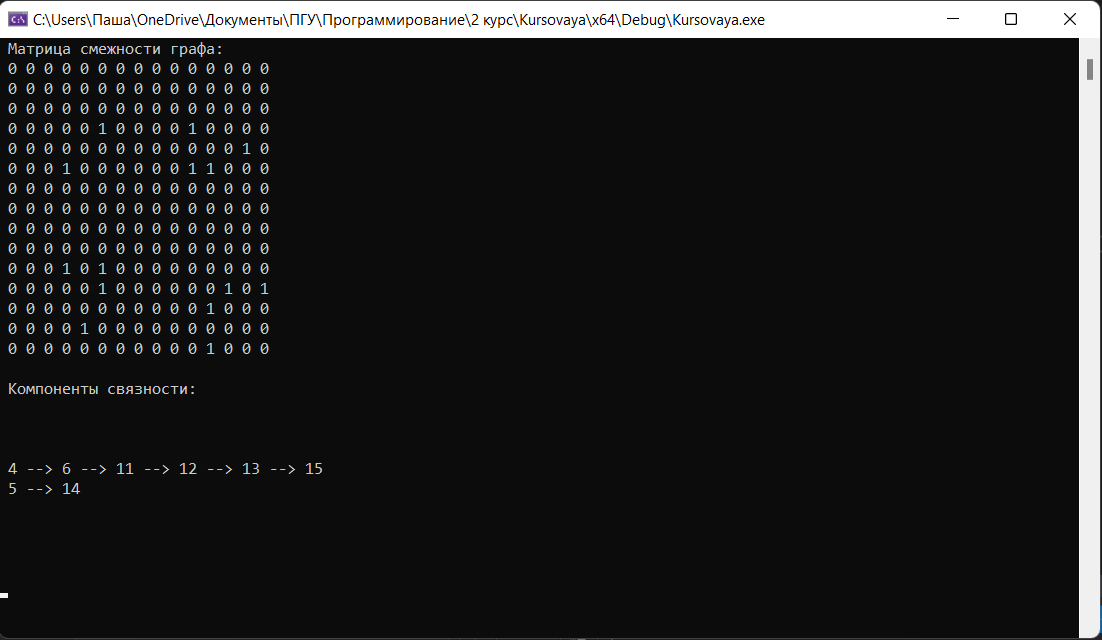


Рисунок 6 – Генерация матрицы размера 15, из файла

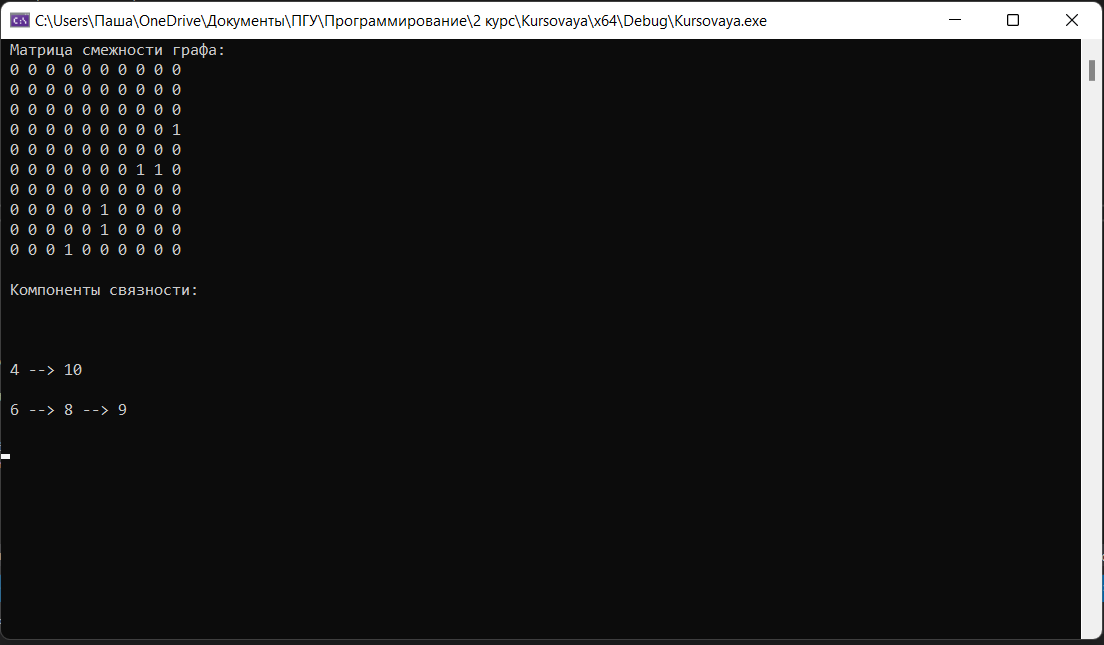


Рисунок 7 – Случайная генерация матрицы размера: 10

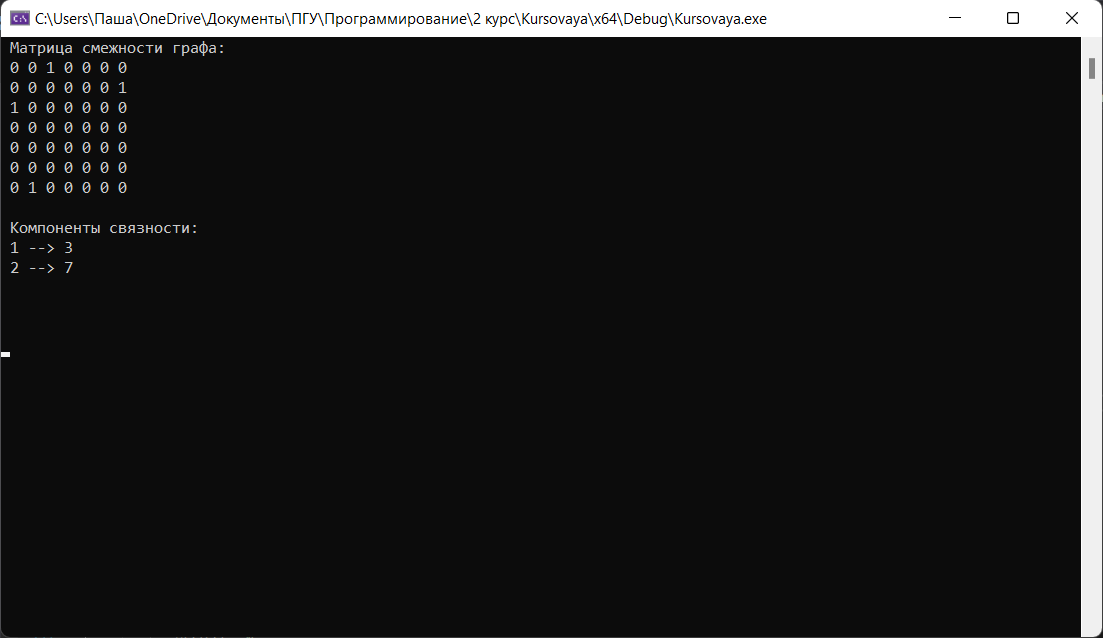


Рисунок 8 – Случайная генерация матрицы размера: 7

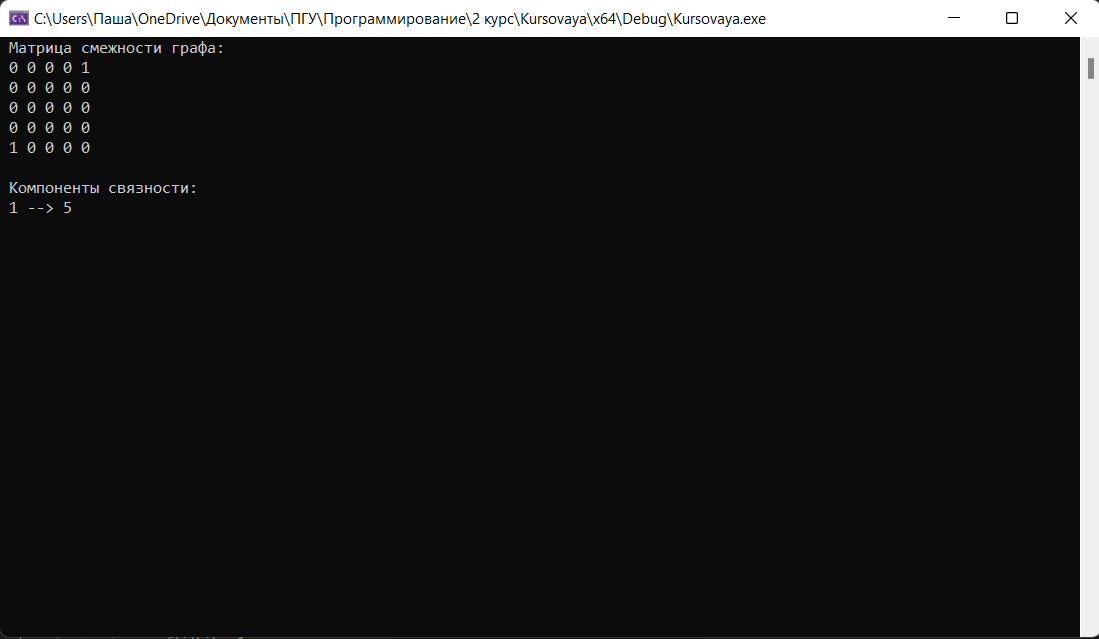


Рисунок 9 – Генерация матрицы размера 5, из файла

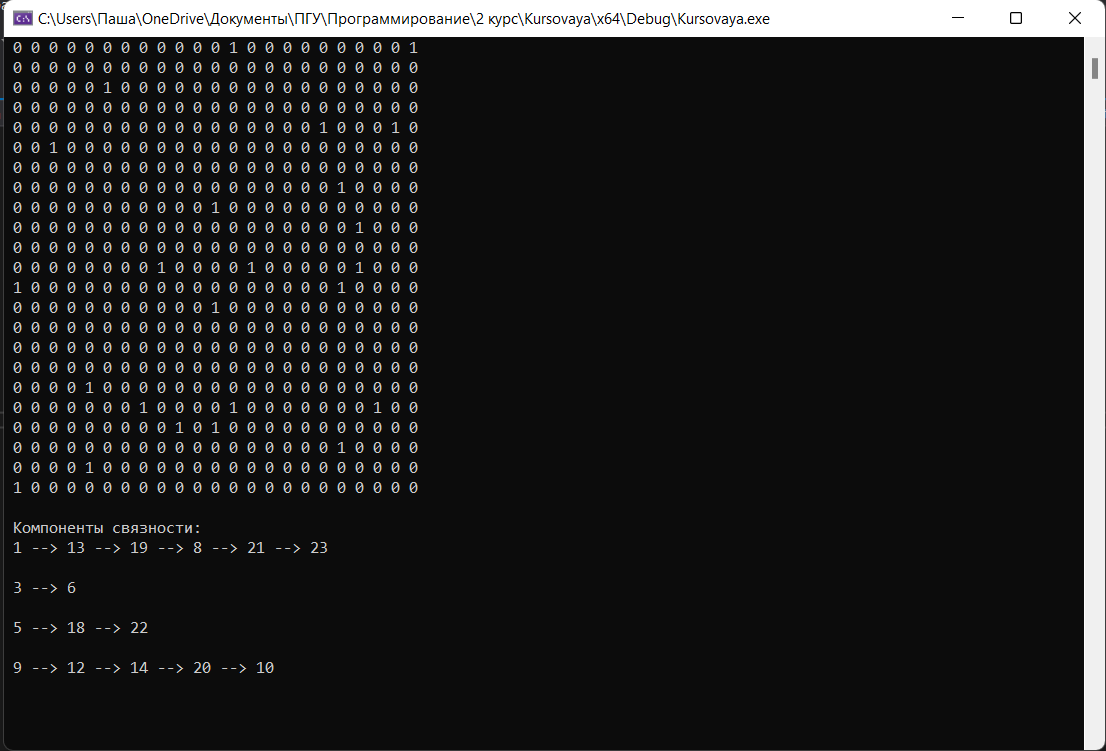


Рисунок 10 – Случайная генерация матрицы размера: 23

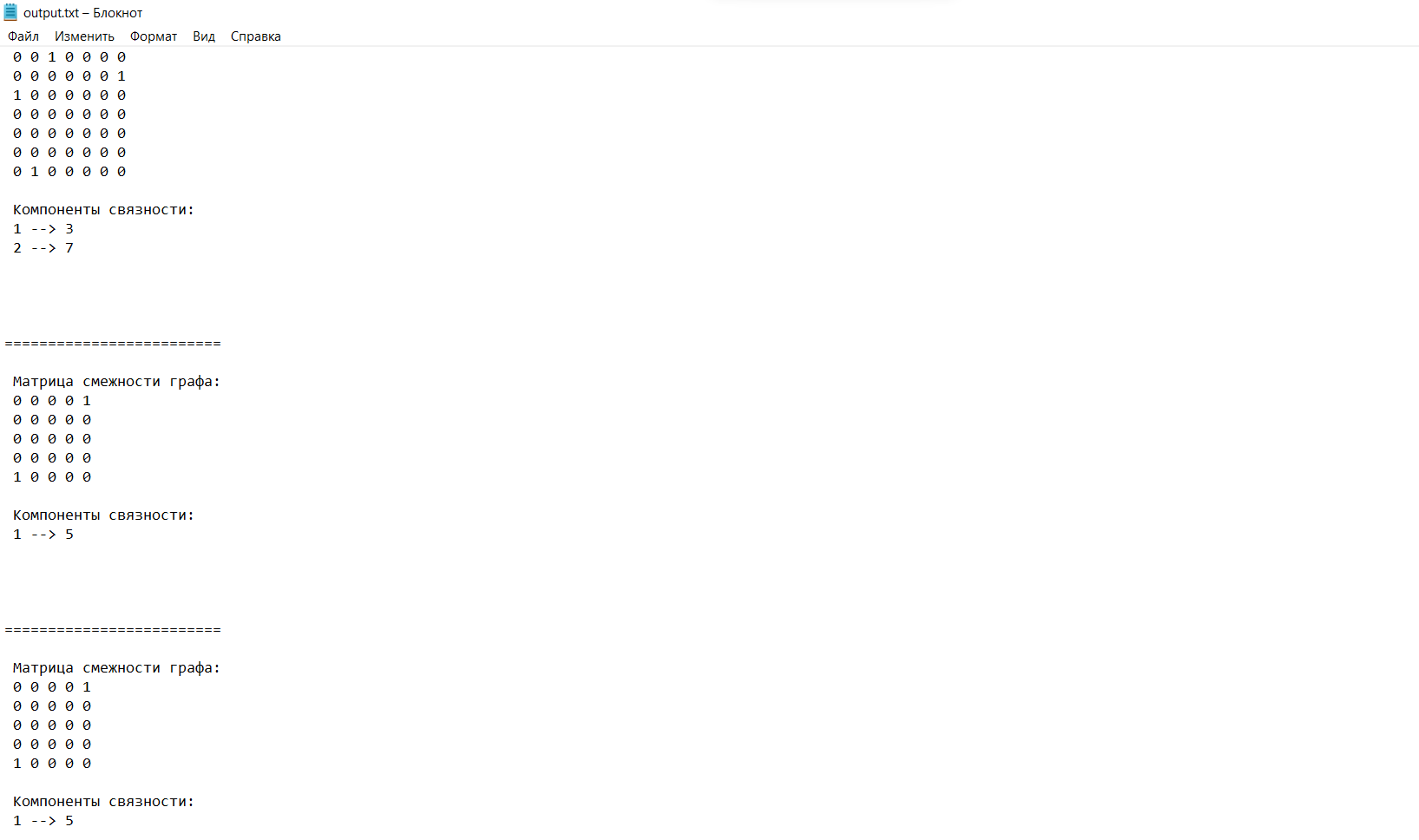


Рисунок 11 – Записанные в файл «output.txt» результаты предыдущих тестов



Рисунок 12 – Записанные в файл «output.txt» результаты предыдущих тестов

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы  (рис. 4 – рис. 10) | Вывод запроса у пользователя ввести количество вершин графа | Верно |
| Выбор генерации матрицы случайным образом  (рис. 5, 7, 8, 10) | Случайная генерация значений матрицы смежности | Верно |
| Выбор загрузки матрицы из файла  (рис. 6, 9) | Загрузка матрицы из заранее подготовленного файла «input.txt» | Верно |
| Нахождение и вывод всех компонент связности графа  (рис. 5 – рис. 10) | Успешное выполнение программы и вывод всех компонент связности в консоль | Верно |
| Выбор количества вершин графа  (рис. 5 – рис. 10) | Генерация матрицы в соответствии с заданным размером | Верно |
| Запись результатов работы программы в файл «output.txt»  (рис. 11, 12) | Запись матрицы смежности и компонент связности в файл с заданным текстовым дополнением | Верно |

По результатам тестирования можно заключить, что программа выполняет все свои функции верно.

# Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере случайно сгенерированного графа с 7 вершинами (рис. 8).

Берем первую вершину как стартовую. Проверяем если ли путь в другие или из других вершин в первую. В нашем случае она связана с 3 вершиной. Вдвоем они представляют компонент связности графа.

Далее, так как первая вершина уже является использованной, то мы берем вторую вершину как стартовую и также выполняем обход в глубину поочередно проверяя наличие ребра, идущего из или во вторую вершину. Такое ребро есть только на пересечении с седьмым столбцом (соответственно с седьмой вершиной). Так как других пересечений второй вершины не оказывается мы переходим к следующей.

Так как третья вершина уже используется в первой компоненте связности, мы ее не используем, а сразу переходим к четвертой вершине, Также как в прошлый раз проверяем наличие ребра между 4 и любой другой вершиной, не находим его – соответственно идем дальше. Остальные вершины, кроме пройденной – седьмой не имеют ребер друг с другом, то есть являются изолированными.

Итак, в итоге в ручном расчете задачи получилось 2 компоненты связности: [1-3] и [2-7].

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

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска компонент связности орграфа через обход графа в глубину, в Microsoft Visual Studio 2019.

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 ширину. Углублены знания языка программирования C++.

Недостатком разработанной программы является примитивный пользовательский интерфейс.

Однако она имеет большой спектр применений на практике.

# 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 <locale.h>

#include <stack>

#include <iostream>

#include <queue>

#include <vector>

#include <fstream>

#include <string>

#include <conio.h>

#include <cstdlib>

#include <time.h>

using namespace std;

int i, j;

ofstream fout;

void DFS\_No\_Rec(int\*\* G, const int n);

void Init\_Matrix(int\*\* G, const int n);

void Out\_Matrix(int\*\* G, const int n);

void Input\_Matrix(int\*\* G, const int n);

void Delete\_Matrix(int\*\* G, const int n);

void find\_comps(int\*\* G, const int n);

void main()

{

setlocale(LC\_ALL, "Rus");

bool check = false;

int n;

cout << " Введите количество вершин в графе: ";

cin >> n;

int\*\* G = new int\* [n];//граф

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

{

G[i] = new int[n];

}

while (check == false)

{

cout << "\n Выберите действие: \n 1) Сгенерировать граф размера : " << n << ", псевдослучайным образом;\n 2) Выгрузить граф из файла;\n ";

int z = \_getch() - 48;

switch (z)

{

case 1:

cout << "1\n\n";

Init\_Matrix(G, n);

check = true;

break;

case 2:

cout << "2\n\n";

Input\_Matrix(G, n);

check = true;

break;

default:

cout << "\n Ошибка выбора. Пожалуйста, введите корректный номер варианта! \n\n";

break;

}

}

system("cls");

fout.open("output.txt", ofstream::app);

cout << " Матрица смежности графа: \n";

fout << " Матрица смежности графа: \n";

Out\_Matrix(G, n);

cout << " Компоненты связности: \n";

fout << " Компоненты связности: \n";

find\_comps(G, n);

fout << "\n=========================\n\n";

fout.close();

Delete\_Matrix(G, n);

system("pause>>void");

}

vector <int> comp;

void dfs(int\*\* G, int n, int v, bool\* used)

{

used[v] = true;

comp.push\_back(v);

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

{

if (G[v][i] == 1 && used[i] == false) //проверяем смежные с ней вершины

{

dfs(G, n, i, used); //вызываем рекурсию

}

}

}

void find\_comps(int\*\* G, const int n)

{

bool\* used = new bool [n];

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

used[i] = false;

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

if (used[i] == false) {

comp.clear();

dfs(G, n, i, used);

for (size\_t k = 0; k < comp.size(); ++k)

{

if (k < 1)

{

cout << " " << comp[k] + 1;

fout << " " << comp[k] + 1;

}

else

{

cout << " --> " << comp[k] + 1;

fout << " --> " << comp[k] + 1;

}

}

cout << endl;

fout << endl;

}

}

void Init\_Matrix(int\*\* G, const int n)

{

srand(time(NULL));

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

{

G[i][i] = 0;

for (j = 0; j < i; j++)

{

if (rand() % 100 <= 5) //заполнение графа случайными числами

G[i][j] = 1;

else

G[i][j] = 0;

if (rand() % 100 <= 5)

G[j][i] = 1;

else

G[j][i] = 0;

}

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

{

G[j][i] = G[i][j];

}

}

}

void Out\_Matrix(int\*\* G, const int n)

{

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

{

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

{

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

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

}

cout << endl;

fout << endl;

}

cout << endl;

fout << endl;

}

void Input\_Matrix(int\*\* G, const int n)

{

ifstream fin;

fin.open("input.txt");

if (!fin.is\_open())

{

cout << "Ошибка при открытии файла!\n";

system("pause>>void");

}

else

{

string str;

int iin = 0;

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

{

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

{

fin >> str;

for (int m = 0; m < str.length(); m++)

{

if (str.at(m) == '1' || str.at(m) == '0')

{

iin = stoi(str);

G[i][j] = iin;

}

else

{

cout << "В файл введены некорректные данные.\n";

system("pause>>void");

}

}

}

}

fin.close();

}

}

void Delete\_Matrix(int\*\* G, const int n)

{

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

delete[] G[i];

delete[] G;

}