

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 22ВВВ1:

Саветкин Д.Д.

Приняли:

к.т.н. доцент Юрова О.В.

д.т.н. профессор Митрохин М.А.

Пенза 2023

**Содержание**

Оглавление

[Реферат 3](#_Toc154608559)

[Введение 4](#_Toc154608560)

[1. Постановка задачи 5](#_Toc154608561)

[2. Теоретическая часть задания 6](#_Toc154608562)

[3. Описание алгоритма программы 8](#_Toc154608563)

[4. Ручной расчет задачи 10](#_Toc154608564)

[5. Описание программы 11](#_Toc154608565)

[6. Тестирование 17](#_Toc154608566)

[Заключение 20](#_Toc154608567)

[Список литературы 21](#_Toc154608568)

[Приложение А 22](#_Toc154608569)

# Реферат

Отчет 25 стр., 9 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 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 MicrosoftVisualStudio, 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.

# 

# 2. 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.

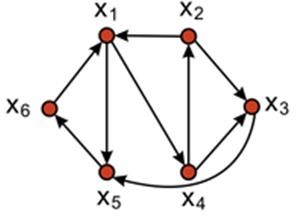


Рисунок 1 – Пример орграфа

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 njпросматривается 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(), DFS()

**DFS():**

Отметка о посещении вершины;

Добавление вершины в вектор;

Для (i = 0; i < кол-ва вершин; i++)

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

**findComponents():**

Для (i = 0; i < кол-ва вершин; i++)

Если вершина не посещена

DFS();

Вывод на экран компоненты;

Запись компоненты в файл;

Обнуление посещения вершин;

Очистка вектора;

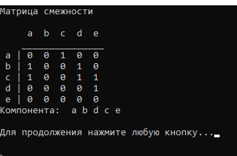
# 4. Ручной расчет задачи

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

Начинаем обход из A вершины в последующие. Проверяем, если есть путь из A в другие вершины, то дальше идем. В нашем случае идем в вершину B. Аналогично проверяем и в вершине B. Из вершины B идем в D. Из вершины D идем в C. Из вершины C идем в E.

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

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



Тестирование работы программы

# 5. Описание программы

Для написания данной программы использован язык программирования Си.Язык программирования Си - 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++).

Для начала работы с алгоритмом требуется указать размерность матрицы и выделить под нее память. Для этого используется функция numberVershin().

void numberVershin(int\* num) {

if (\*num != 0 && G != NULL) {

for (int i = 0; i < \*num; i++) {

free(G[i]);

}

free(G);

free(numV);

}

system("cls");

printf("Размер графа: ");

scanf\_s("%d", &\*num);

G = (int\*\*)malloc(\*num \* sizeof(int\*));

numV = (int\*)malloc(\*num \* sizeof(int));

for (int i = 0; i < \*num; i++) {

G[i] = (int\*)malloc(\*num \* sizeof(int));

numV[i] = NULL;

}

}

Затем, требуется заполнить матрицу либо случайными числами, используя функцию generateMatrix(), либо вручную, используя функцию scanfMatrix().

void generateMatrix(int num) {

if (num == 0) return;

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

numV[i] = 1;

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

if (i == j) {

G[i][j] = 0;

continue;

}

G[i][j] = rand() % 2;

}

}

}

void scanfMatrix(int num) {

if (num == 0 || G == NULL) return;

int x = 0, y = 0;

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

numV[i] = 1;

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

G[i][j] = 0;

}

}

while (true) {

system("cls");

printf("Вы вводите вершину и связную с ней вершину\n");

printf("Нумерация вершин начинается с 1\n");

printf("(-1) прекращает заполнение\n");

printf("Введите вершину 1: ");

scanf\_s("%d", &x);

printf("Введите вершину 2: ");

scanf\_s("%d", &y);

if (x == -1 || y == -1) break;

if (x > num || y > num || x < 1 || y < 1) {

printf("\nВы вышли за границы матрицы");

printf("\n\nДля продолжения нажмите любую кнопку...");

\_getch();

continue;

}

if (x == y) continue;

G[x-1][y-1] = 1;

}

}

После, ранее описанных манипуляций , пользователь может применить алгоритм выделения компонент, используя поиск в глубину. Для его реализации используются функции DFS() и findComponents().

void DFS(int v, int total) {

numV[v] = 0;

comp.push\_back(v);

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

if ((G[v][i] == 1 || G[i][v] == 1) && numV[i] == 1)

DFS(i, total);

}

}

void findComponents(int num, FILE\* f) {

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

if (numV[i] == 1) {

DFS(i, num);

printf("Компонента: ");

fprintf(f, "Компонента: ");

for (int j = 0; j < comp.size(); j++) {

printf(" %c", comp[j] + 97);

fprintf(f, " %c", comp[j] + 97);

}

fprintf(f, "\n");

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

numV[i] = 1;

}

comp.clear();

}

}

}

Для выведения результатов на экран используется функция printMatrix().

Она так же сохраняет записи в файл.

void printMatrix(int num, int v) {

if (num == 0 || numV[v] == NULL) return;

FILE\* f = fopen("answer.txt", "w");

system("cls");

printf("Матрица смежности \n");

fprintf(f, "Матрица смежности \n");

printf("\n ");

fprintf(f, "\n ");

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

printf(" %c", (i + 97));

fprintf(f, " %c", (i + 97));

}

printf("\n ");

fprintf(f, "\n ");

for (int i = 0; i < num \* 3; i++) {

printf("\_");

fprintf(f, "\_");

}

printf("\n");

fprintf(f, "\n");

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

printf(" %c | ", (i + 97));

fprintf(f, " %c | ", (i + 97));

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

printf("%d ", G[i][j]);

fprintf(f, "%d ", G[i][j]);

}

printf("\n");

fprintf(f, "\n");

}

fprintf(f, "\n");

findComponents(num, f);

fclose(f);

printf("\n\nДля продолжения нажмите любую кнопку...");

\_getch();

}

Перед тем, как завершить работу, программа очищает выделенную память, используя функцию freeMatrix().

void freeMatrix(int num) {

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

free(G[i]);

}

free(G);

free(numV);

}

Для использования вышеперечисленного функционала используется меню.

Описанное в функции main().

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int num = 0, v = 0, var = 0;

while (var != 5) {

system("cls");

printf("\tМеню:\n\n"

"1. Кол - во вершин\n"

"2. Вывод матрицы\n"

"3. Случайная генерация матрицы\n"

"4. Заполнение матрицы\n"

"5. Выход\n"

"> ");

scanf\_s("%d", &var);

switch (var) {

case 1:

numberVershin(&num);

break;

case 2:

printMatrix(num, v);

break;

case 3:

srand(time(NULL));

generateMatrix(num);

break;

case 4:

scanfMatrix(num);

break;

case 5:

freeMatrix(num);

break;

default:

printf("\nТакого пункта нет...");

\_getch();

break;

}

}

}

Ниже на рисунках можно увидеть работу выше описанной программы.

Рисунок 1 – Меню программы

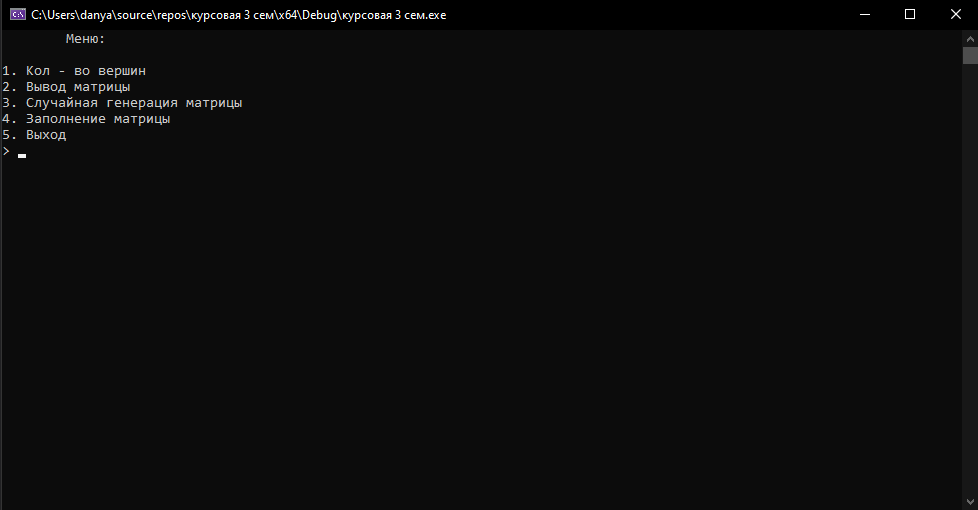


Рисунок 2 - Задание размера матрицы

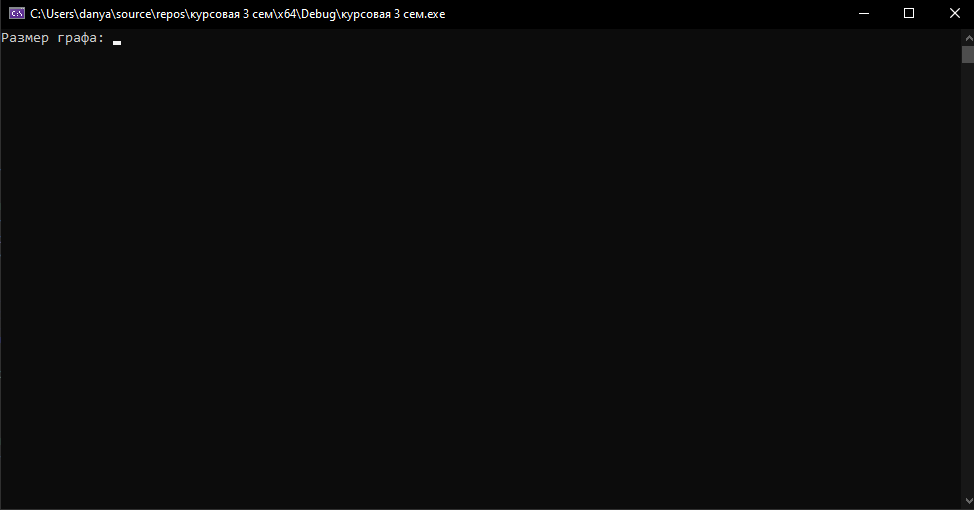


Рисунок 3 – Заполнение матрицы вручную

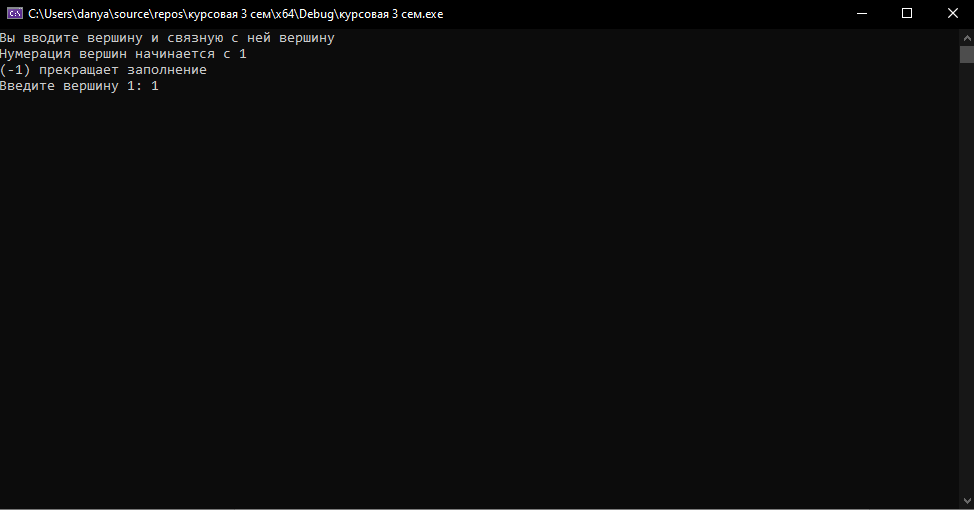
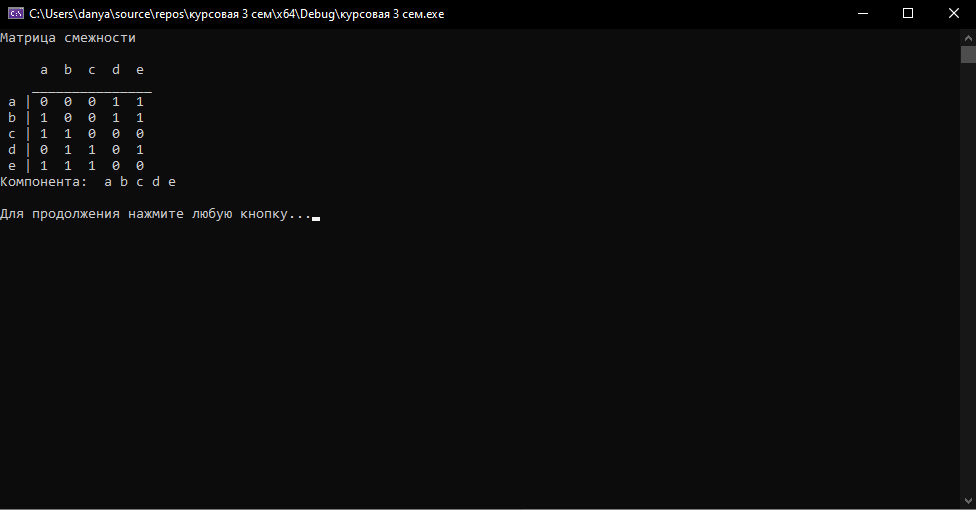


Рисунок 4 – Вывод работы программы на экран



# 6. Тестирование

Среда разработки MicrosoftVisualStudio 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 5 – Тестирование при вводе количество вершин = 5

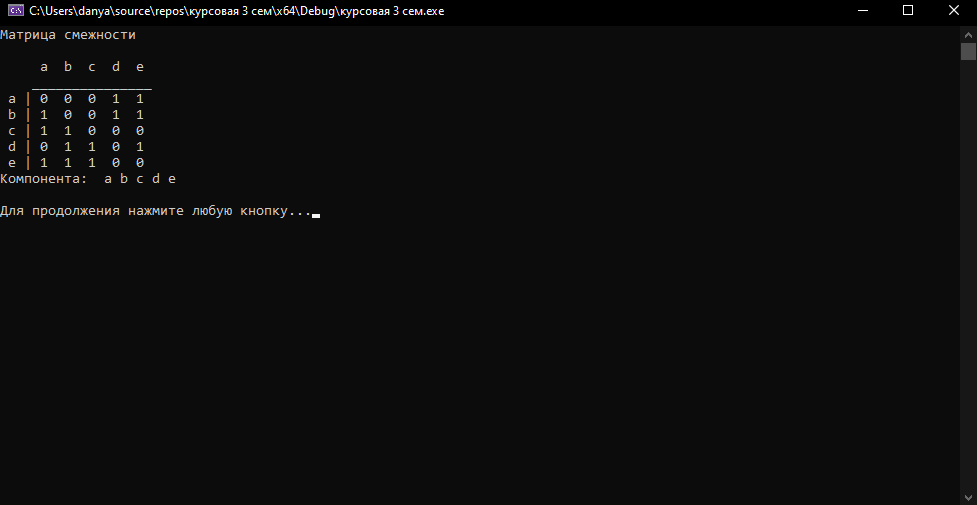


Рисунок 6 – Тестирование при вводе количество вершин = 8

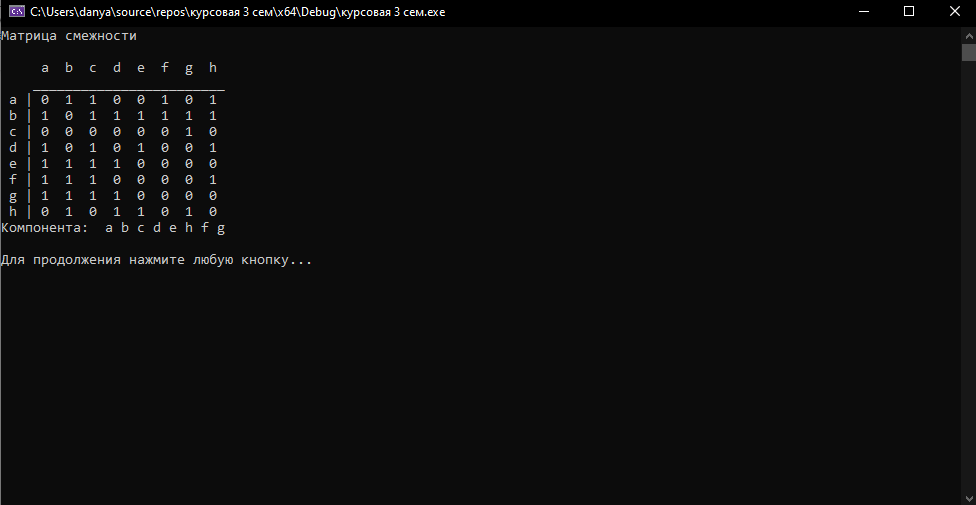
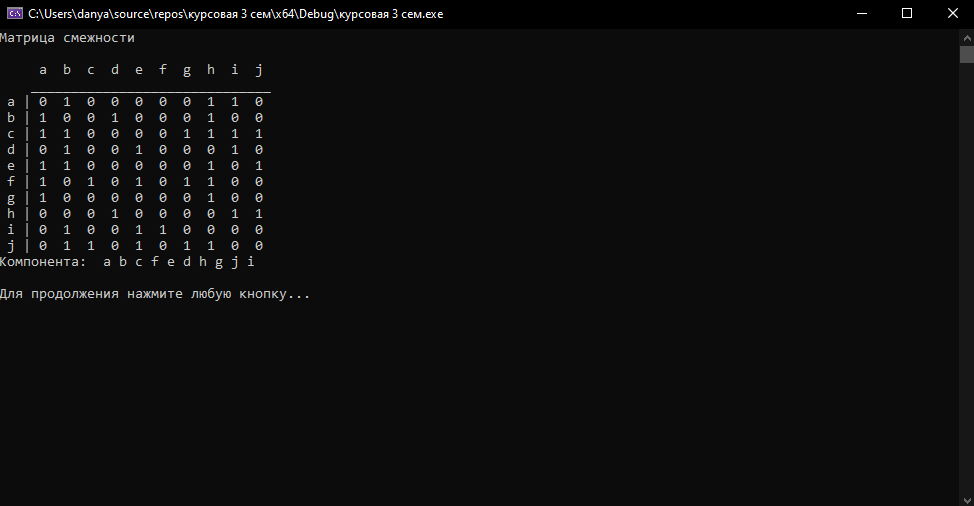


Рисунок 7 - Тестирование при вводе количество вершин = 10



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

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска в глубину для поиска компонент 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 ширину. 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**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<windows.h>

#include<string.h>

#include<conio.h>

#include <fstream>

#include<queue>

using namespace std;

queue <int> Q;

vector <int> comp;

int\*\* G = NULL;

int\* numV = NULL;

void findComponents(int num, FILE\* f); //объявление функции

//случайная генерация матрицы

void generateMatrix(int num) {

if (num == 0) return;

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

numV[i] = 1;

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

if (i == j) {

G[i][j] = 0;

continue;

}

G[i][j] = rand() % 2;

}

}

}

//определение размера матрицы и выделения под нее памяти

void numberVershin(int\* num) {

if (\*num != 0 && G != NULL) {

for (int i = 0; i < \*num; i++) {

free(G[i]);

}

free(G);

free(numV);

}

system("cls");

printf("Размер графа: ");

scanf\_s("%d", &\*num);

G = (int\*\*)malloc(\*num \* sizeof(int\*));

numV = (int\*)malloc(\*num \* sizeof(int));

for (int i = 0; i < \*num; i++) {

G[i] = (int\*)malloc(\*num \* sizeof(int));

numV[i] = NULL;

}

}

//вывод матрицы на экран

void printMatrix(int num, int v) {

if (num == 0 || numV[v] == NULL) return;

FILE\* f = fopen("answer.txt", "w");

system("cls");

printf("Матрица смежности \n");

fprintf(f, "Матрица смежности \n");

printf("\n ");

fprintf(f, "\n ");

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

printf(" %c", (i + 97));

fprintf(f, " %c", (i + 97));

}

printf("\n ");

fprintf(f, "\n ");

for (int i = 0; i < num \* 3; i++) {

printf("\_");

fprintf(f, "\_");

}

printf("\n");

fprintf(f, "\n");

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

printf(" %c | ", (i + 97));

fprintf(f, " %c | ", (i + 97));

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

printf("%d ", G[i][j]);

fprintf(f, "%d ", G[i][j]);

}

printf("\n");

fprintf(f, "\n");

}

fprintf(f, "\n");

findComponents(num, f);

fclose(f);

printf("\n\nДля продолжения нажмите любую кнопку...");

\_getch();

}

//вписывание своих значений в матрицу

void scanfMatrix(int num) {

if (num == 0 || G == NULL) return;

int x = 0, y = 0;

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

numV[i] = 1;

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

G[i][j] = 0;

}

}

while (true) {

system("cls");

printf("Вы вводите вершину и связную с ней вершину\n");

printf("Нумерация вершин начинается с 1\n");

printf("(-1) прекращает заполнение\n");

printf("Введите вершину 1: ");

scanf\_s("%d", &x);

printf("Введите вершину 2: ");

scanf\_s("%d", &y);

if (x == -1 || y == -1) break;

if (x > num || y > num || x < 1 || y < 1) {

printf("\nВы вышли за границы матрицы");

printf("\n\nДля продолжения нажмите любую кнопку...");

\_getch();

continue;

}

if (x == y) continue;

G[x - 1][y - 1] = 1;

}

}

//очистка памяти

void freeMatrix(int num) {

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

free(G[i]);

}

free(G);

free(numV);

}

//обход в глубину

void DFS(int v, int total) {

numV[v] = 0;

comp.push\_back(v);

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

if ((G[v][i] == 1 || G[i][v] == 1) && numV[i] == 1)

DFS(i, total);

}

}

//поиск компонент

void findComponents(int num, FILE\* f) {

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

if (numV[i] == 1) {

DFS(i, num);

printf("Компонента: ");

fprintf(f, "Компонента: ");

for (int j = 0; j < comp.size(); j++) {

printf(" %c", comp[j] + 97);

fprintf(f, " %c", comp[j] + 97);

}

fprintf(f, "\n");

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

numV[i] = 1;

}

comp.clear();

}

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int num = 0, v = 0, var = 0;

while (var != 5) {

system("cls");

printf("\tМеню:\n\n"

"1. Кол - во вершин\n"

"2. Вывод матрицы\n"

"3. Случайная генерация матрицы\n"

"4. Заполнение матрицы\n"

"5. Выход\n"

"> ");

scanf\_s("%d", &var);

switch (var) {

case 1:

numberVershin(&num);

break;

case 2:

printMatrix(num, v);

break;

case 3:

srand(time(NULL));

generateMatrix(num);

break;

case 4:

scanfMatrix(num);

break;

case 5:

freeMatrix(num);

break;

default:

printf("\nТакого пункта нет...");

\_getch();

break;

}

}

}