Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Пояснительная записка**

К курсовому проектированию

По курсу «Логика и основы алгоритмизации

в инженерных задачах»

на тему «Реализация алгоритма обхода в ширину»

Выполнил:

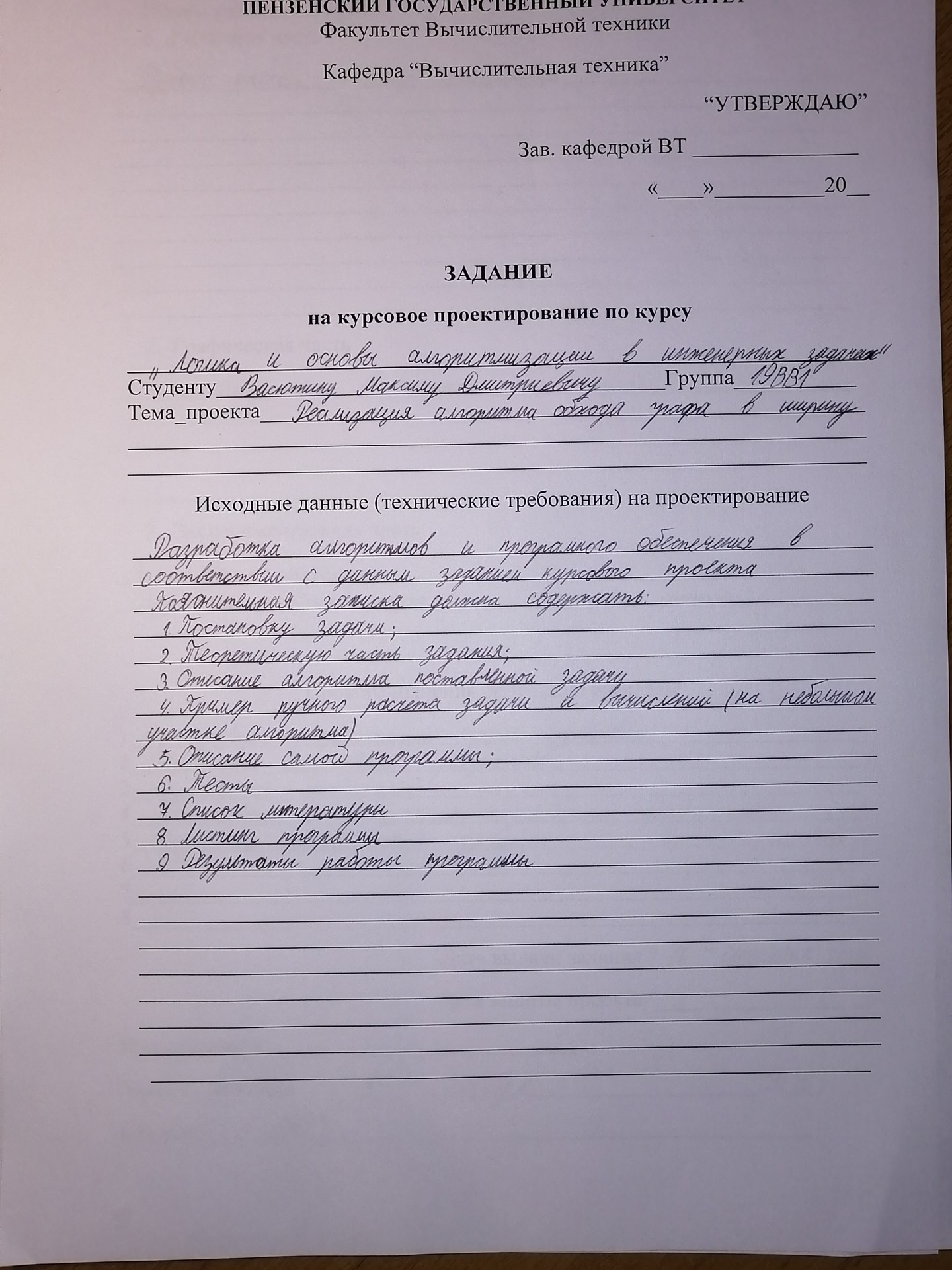
студент группы 19ВВ1

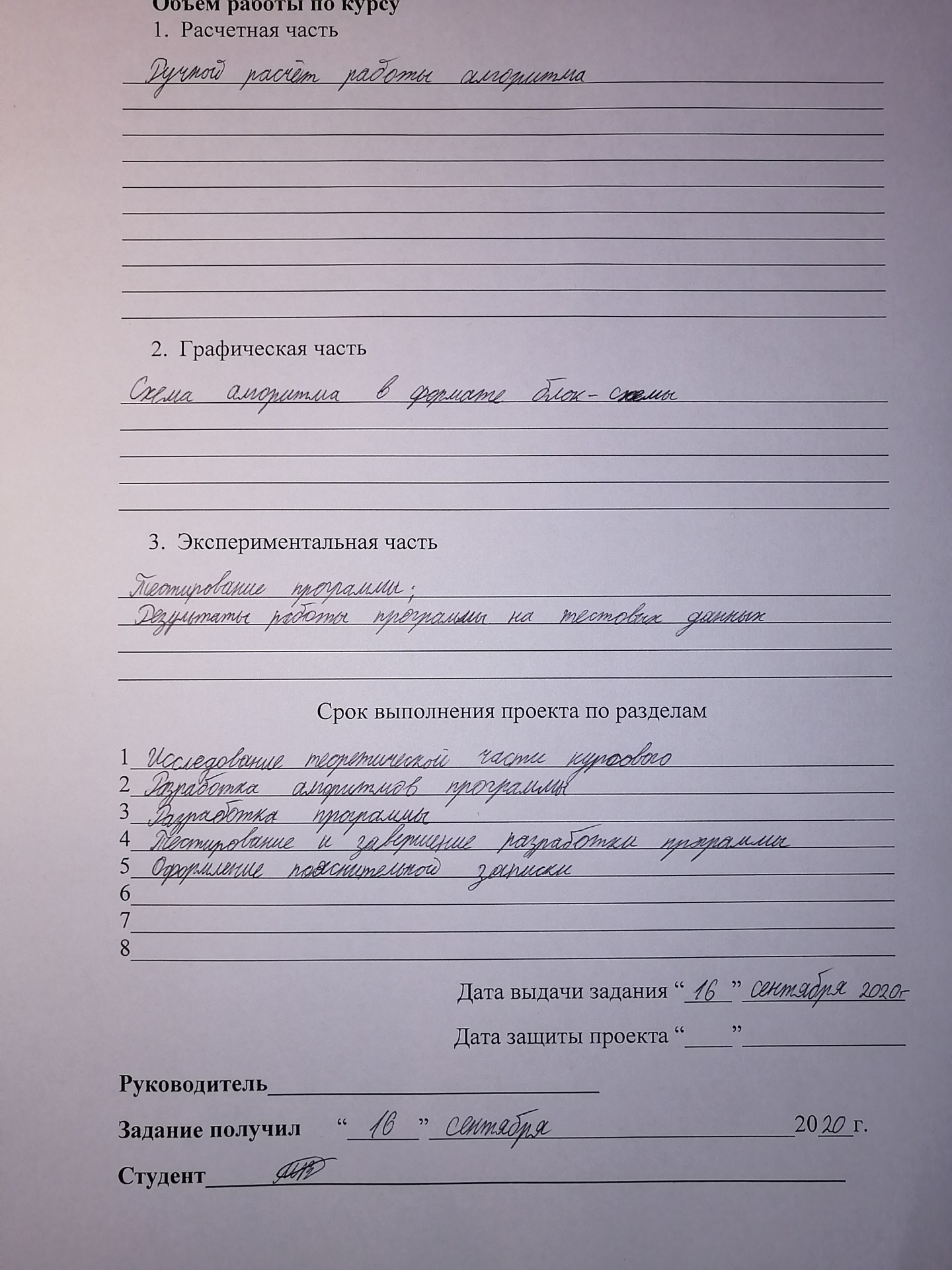
Васютин М.Д.

Принял:

Митрохин М.А.

Пенза 2020





**Содержание**

[1. Реферат 5](#_Toc58799884)

[2. Введение 6](#_Toc58799885)

[3. Постановка задачи 6](#_Toc58799886)

[4. Теоретическая часть задания 7](#_Toc58799887)

[5. Описание алгоритма программы 9](#_Toc58799888)

[6. Описание программы 12](#_Toc58799889)

[7. Тестирование 20](#_Toc58799890)

[8. Ручной расчёт задачи 24](#_Toc58799891)

[9. Заключение 26](#_Toc58799892)

[10. Список литературы 27](#_Toc58799893)

[11. Приложения 28](#_Toc58799894)

# Реферат

Отчет 27 стр, 12 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ОБХОДА В ШИРИНУ.

Цель исследования – разработка программы, реализующую алгоритм обхода в ширину.

В работе рассмотрен алгоритм обхода в ширину.

# Введение

Обход графа в ширину – один распространенный способ обхода графов. Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины. Например, если для графа на рисунке 1 начать обход из первой вершины, то вершины 3, 6 и 2 будут находиться на уровне удаленности в 1 ребро, а вершины 5 и 4 на уровне удаленности в 2 ребра.



Рисунок 1 – Граф

Тогда при обходе этого графа в ширину, мы сначала посетим вершины первого уровня удаленности (с номерами 2, 3 и 6), и только после того, как закончатся не посещенные вершины на этом уровне, мы перейдем к следующему. На втором уровне мы посетим все вершины, которые удалены от исходной на 2 ребра (вершины 4 и 5).

Так, алгоритм обхода в ширину продолжает осматривать уровень за уровнем, пока не пройдет все доступные вершины.

Чтобы не заходить повторно в уже пройденные вершины, они помечаются, как и в алгоритме обхода в глубину.

Для того, чтобы проход осуществлялся по уровням необходимо хранить информацию о требуемом порядке посещения вершин. Вершины, которые являются ближайшими соседями исходной вершины (из которой начат обход) должны быть посещены раньше, чем соседи соседей и т.д. Такой порядок позволяет задать структура данных «очередь». Просматривая строку матрицы смежности (или список смежности) для текущей вершины мы помещаем всех её ещё не посещенных соседей в очередь. На следующей итерации текущей вершиной становится та, которая стоит в очереди первой и уже её не посещенные соседи будут помещены в очередь. Но место в очереди они займут после тех вершин, которые были помещены туда на предыдущих итерациях.

# Постановка задачи

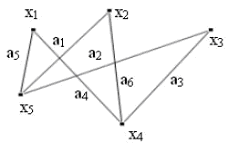
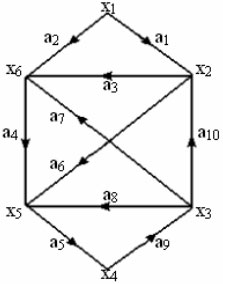
Требуется разработать программу, которая осуществит обход графа в ширину.

Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь мог выбирать способ создания матрицы смежности (ручной или автоматический метод заполнения), вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности графа. Для совершения обхода в ширину пользователь должен ввести номер начальной вершины. Если пользователь введет номер несуществующей команды или выберет для поиска расстояния несуществующую вершину, программа должна вывести соответствующее сообщение с просьбой заново ввести требуемые данные. Необходимо предусмотреть различные исходы, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

# Теоретическая часть задания

**Сведения о графах**

Граф G задается множеством точек (вершин) х1, х2,..., хn. (которое обозначается через Х) и множеством линий (ребер) а1, а2,...,аm. (которое обозначается символом А), соединяющих между собой все или часть этих точек. Таким образом, граф G полностью задается (и обозначается) парой (Х, А). Если ребра из множества А ориентированы, что обычно показывается стрелкой, то они называются дугами, и граф с такими ребрами называется ориентированным графом.



*а б*

Рисунок 2 – Примеры графов: орграф (*а*) и неорграф (*б*)

Например, если дорога имеет не двух-, а одностороннее движение то направление этого движения будет показано стрелкой.

Если ребра не имеют ориентации, то граф называется неориентированным, (двухстороннее движение).

В ориентированном графе дуга обозначается упорядоченной парой, состоящей из начальной и конечной вершин, ее направление предполагается заданным от первой вершины ко второй.

Путем (или ориентированным маршрутом) ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей.

Так, путями являются последовательности дуг:

а6, а5, а9, а8, а4. (1)

а1, а6, а5, а9. (2)

а1, а6, а5, а9, а10, а6, а4. (3)

Ориентированной цепью (орцепью) называется такой путь, в котором каждая дуга используется не больше одного раза.

Простой орцепью называется такой путь, в котором каждая вершина используется не более одного раза. Например, путь (2).

Маршрут есть неориентированный “двойник” пути, и это понятие рассматривается в тех случаях, когда можно пренебречь направленностью дуг в графе. Таким образом, маршрут есть последовательность ребер ä1, ä2,..., äq, в которой каждое ребро аi, за исключением первого и последнего ребер, связано с ребрами аi-1 и аi+1 своими концевыми вершинами. Последовательности дуг:

ä2, ä4, ä8, ä10, (4)

ä2, ä7, ä8, ä4, ä3, (5)

ä10 , ä7 , ä4 , ä8 , ä7 , ä2. (6)

# Описание алгоритма программы

Таблица 1 – Описание переменных

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Роль** |
| i, j | int | Счётчики |
| st | int | Начальная вершина |
| N | int | Количество вершин графа |
| menu | int | Номер команды главного меню |
| mean | int | Переменная для выбора способа создания графа (1 – ручной, 2 – автоматический) |
| visited | int | Массив, содержащий посещенные вершины |
| graph | int | Матрица смежности графа |
| Save\_matr[3], save\_result[3] | char | Переменные, в которые записывается реакция пользователя на запрос программы |

**Описание работы функции BFS()**

Реализация состоит из подготовительной части, в которой все вершины помечаются как не посещенные и осуществляется запуск процедуры обхода для вершин графа.

В самой процедуре обхода сначала создается пустая очередь, в которую помещается начальная вершина, из которой начат обход.

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной. Затем в цикле просматривается **st**-я строка матрицы смежности графа graph(st, j). Как только алгоритм встречает смежную с **st** не посещенную вершину, эта вершина помещается в очередь и помечается как посещенная. После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла или заканчивает работу, если очередь пуста.

Ниже представлен псевдокод функций BFS()

**BFS()**

Создать пустую очередь Q = {};

Поместить v в очередь Q.push(st);

пометить st как "посещенную" visited[st] = True;

**ПОКА**  Q != ∅ очередь не пуста **ВЫПОЛНЯТЬ**

st = Q.front() установить текущую вершину;

Удалить первый элемент из очереди Q.pop();

вывести на экран st;

**ДЛЯ** i = 1 **ДО** N **ВЫПОЛНЯТЬ**

**ЕСЛИ** graph(st,j) = = 1**И** Не visited[j]

**ТО**

Поместить j в очередь Q.push(j);

пометить j как "посещенную" visited[j] = True;

# 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 VISUAL C++ .

Данная программа является многомодульной, поскольку состоит из нескольких функций: main(), MMM(), BFS(), BFS\_SAVE (), MENU().

Работа программы начинается с вывода меню программы. Меню программы реализовано в функции MENU(), код которой представлен ниже.

void MENU()

{

system("cls");

printf("\n Меню программы ");

printf("\n 1. Начать работу");

printf("\n 2. Завершить работу");

printf("\n\n Введите номер команды - ");

scanf("%d", &menu);

while (menu > 2 || menu < 1)

{

system("cls");

printf("\n Меню программы ");

printf("\n 1. Начать работу");

printf("\n 2. Завершить работу");

printf("\n\n Введите номер команды - ");

scanf("%d", &menu);

}

system("cls");

return;

}

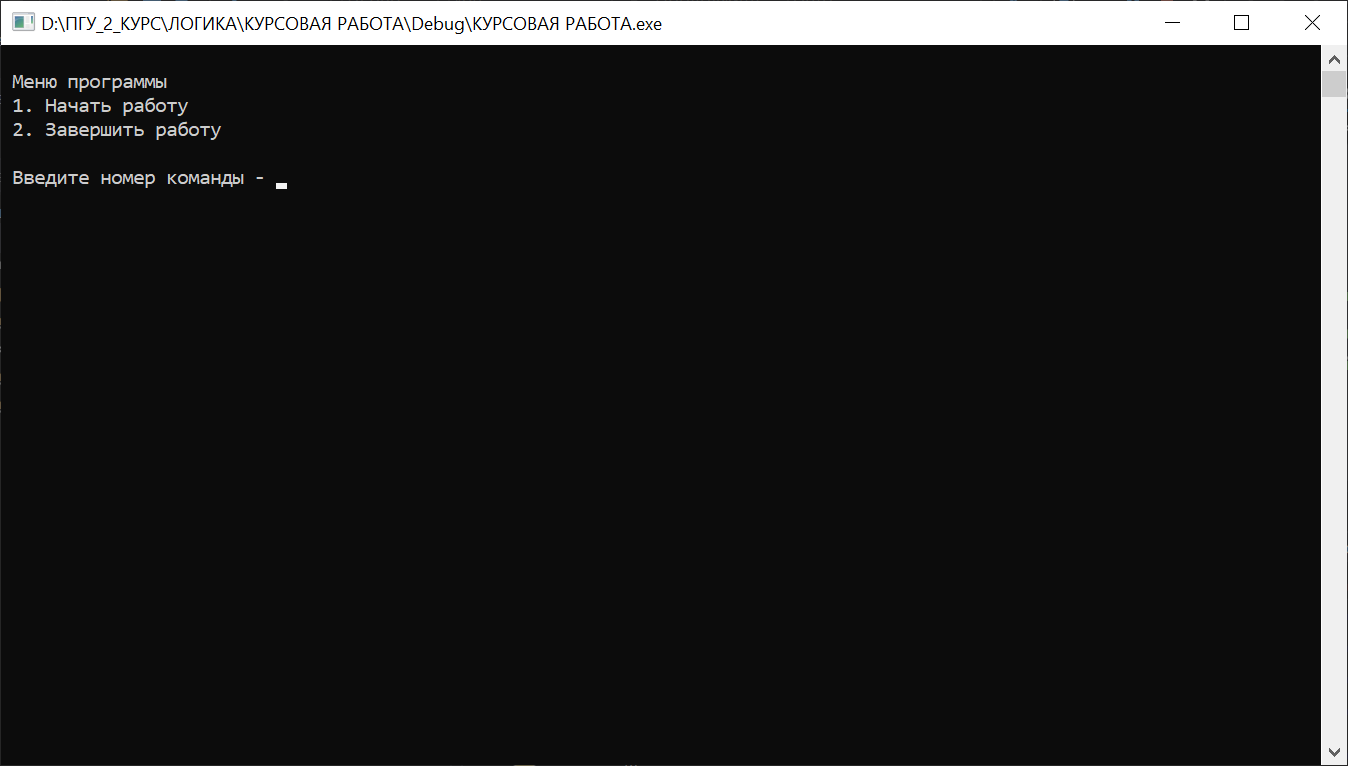


Рисунок 3 – Меню

При выборе Пункта 1 пользователь попадает в меню генерации матрицы смежности. Тут запускается функция MMM(), код которой представлен ниже.

void MMM()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("\n Введите размер матрицы ");

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

printf("\n Способ создания матрицы\n 1. Ручное заполнение\n 2. Автоматическое заполнение \n\n Выберите способ - ");

scanf("%d", &mean);

while (mean > 2)

{

printf("\n Номер такого способа не существует\n Выберите способ заново - ");

scanf("%d", &mean);

}

printf("\n");

srand(time(NULL));

graph = new int\* [N];

visited = new int[N];

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

graph[i] = new int[N];

}

if (mean == 1) {

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

{

for (j = i + 1; j < N; j++)

{

printf(" Ребро между %d и %d - ", i + 1, j + 1);

scanf\_s("%d", &graph[i][j]);

graph[j][i] = graph[i][j];

graph[i][i] = graph[j][j] = 0;

}

}

}

if (mean == 2) {

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

{

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

{

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

graph[i][i] = graph[j][j] = 0;

}

}

printf("\n ");

}

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

{

printf("%4d ", j + 1);

}

printf("\n\n");

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

{

printf(" %d ", i + 1);

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

{

printf("%4d ", graph[i][j]);

}

printf("\n\n");

}

printf("\n Введите вершину: ");

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

while (st > N) {

printf("\n Такая вершина не существует!\n");

printf("\n Введите вершину: ");

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

}

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

{

visited[i] = false;

}

printf("\n Результат обхода в ширину: ");

BFS(st - 1);

printf("\n\n");

printf(" Сохранить матрицу в файл? ");

scanf("%s", &save\_matr);

if ((strcmp(Yes, save\_matr) == 0) || (strcmp(yes, save\_matr) == 0))

{

result = fopen("РЕЗУЛЬТАТ.txt", "w");

fprintf(result, " Размер матрицы %dx%d \n\n ", N, N);

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

{

fprintf(result, "%4d ", j + 1);

}

fprintf(result, "\n\n");

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

{

fprintf(result, " %d ", i + 1);

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

{

fprintf(result, "%4d ", graph[i][j]);

}

fprintf(result, "\n\n");

}

fclose(result);

}

printf(" Сохранить результат обхода в файл? ");

scanf("%s", &save\_result);

if ((strcmp(Yes, save\_result) == 0) || (strcmp(yes, save\_result) == 0))

{

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

{

visited[i] = false;

}

BFS\_SAVE(st-1);

}

return;

}

Программа просит ввести размер матрицы, после чего просит выбрать способ заполнения матрицы – ручной или автоматический.

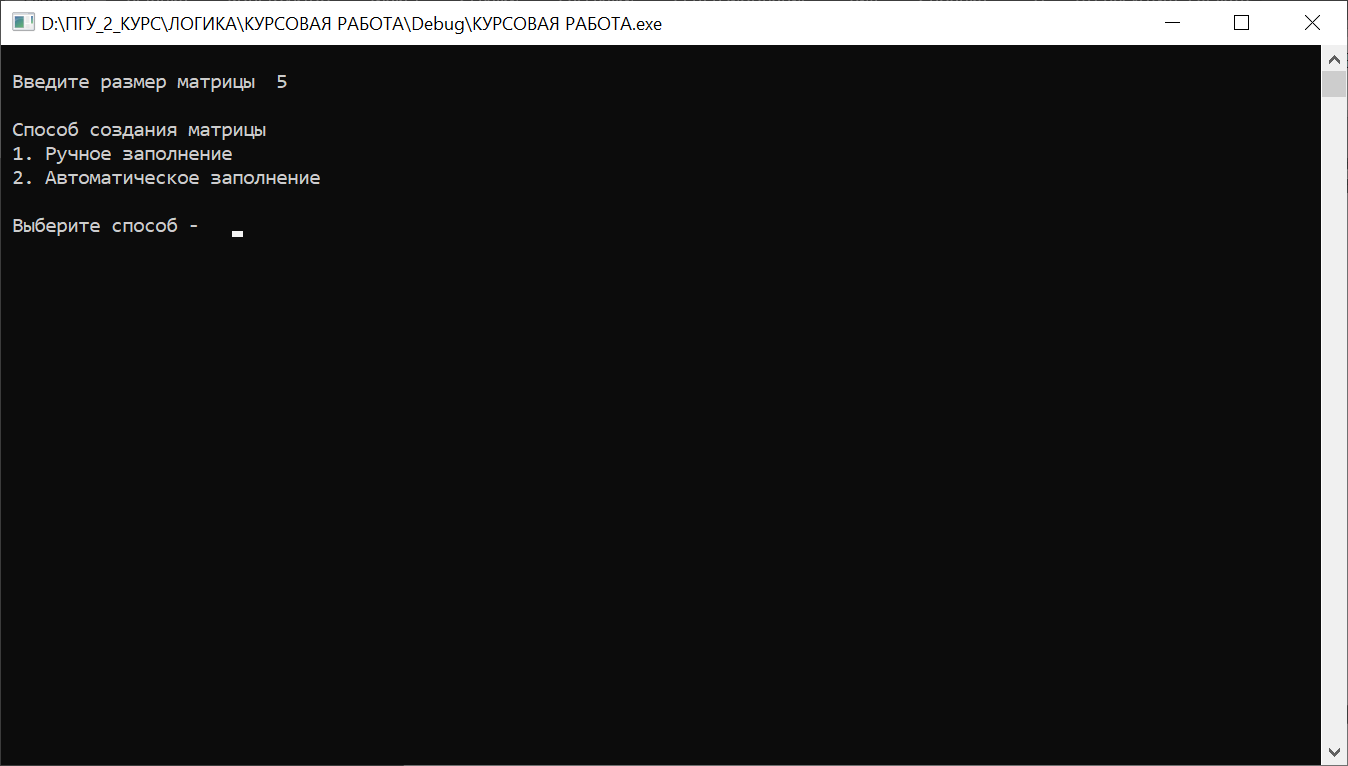


Рисунок 3 – Генерация матрицы

При выборе 1го пункта пользователю потребуется заполнить матрицу смежности. При выборе 2го пункта программа заполнит матрицу случайным образом.

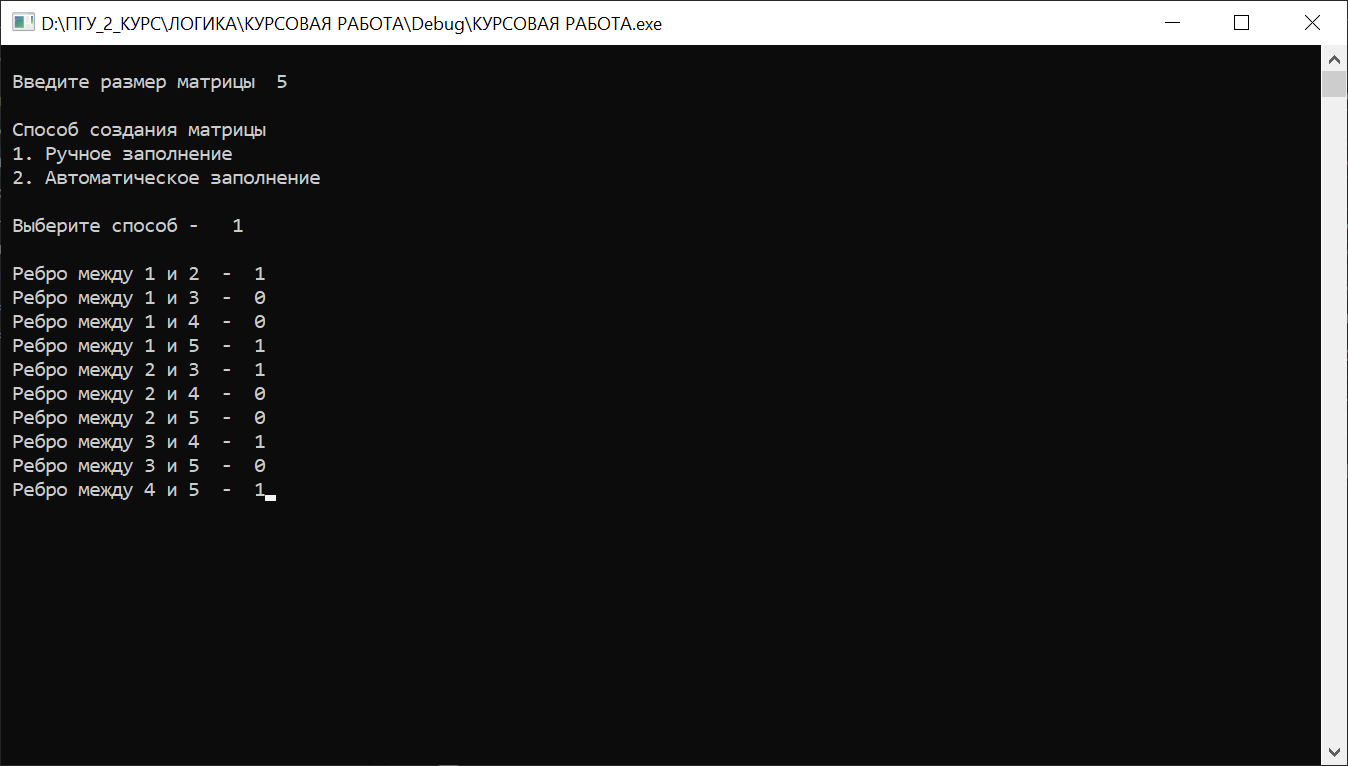


Рисунок 4 – Ручное заполнение матрицы

После заполнения полученная матрица будет выведена на экран.

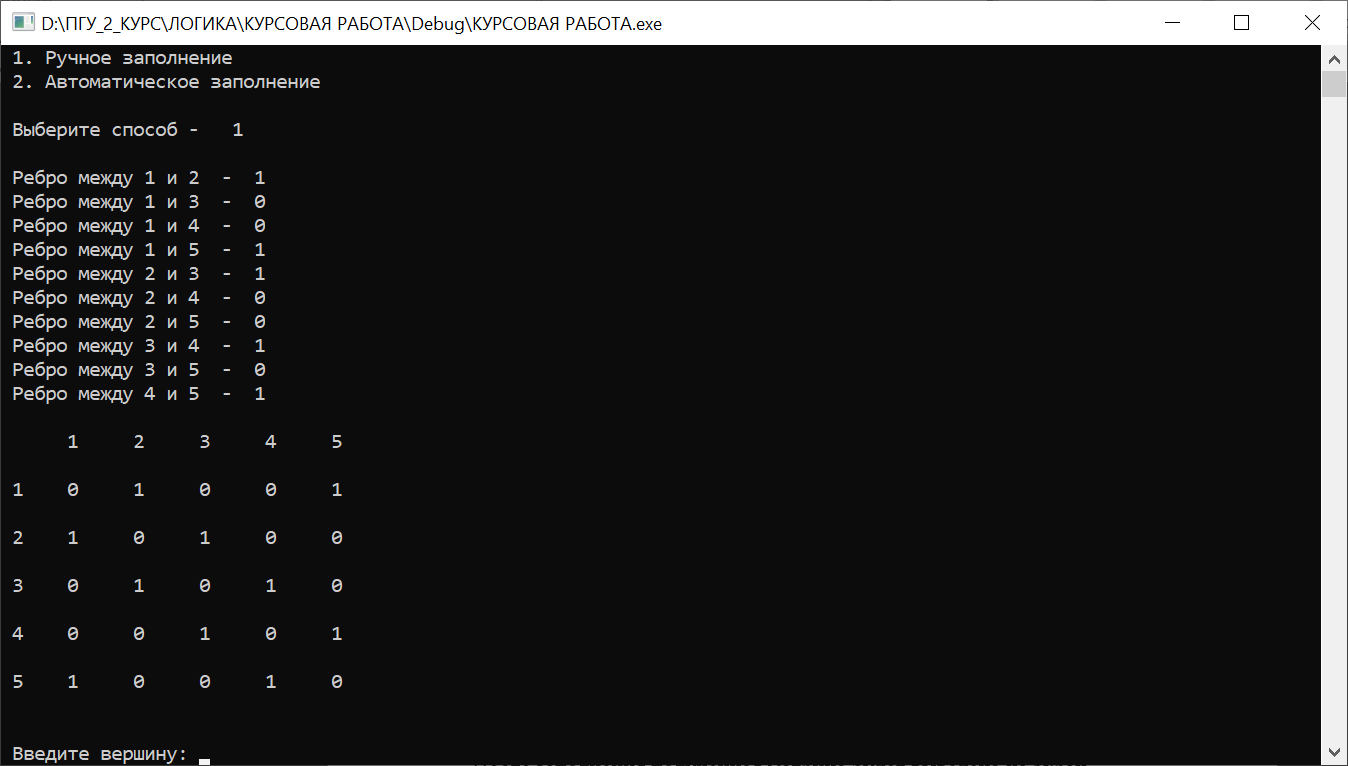


Рисунок 5 – Вывод матрицы

После вывода матрица программа попросит ввести начальную вершину для обхода в ширину. Когда вершина будет выбрана, вызывается функция BFS(), код которой представлен ниже.

void BFS(int st)

{

Q.push(st);

visited[st] = true;

while (!Q.empty())

{

st = Q.front();

Q.pop();

printf("%d ", st + 1);

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

{

if ((graph[st][j] == 1) && (!visited[j]))

{

Q.push(j);

visited[j] = true;

}

}

}

}

После выполнения алгоритма на экране будет результат обхода в ширину.



Рисунок 6 – Результат обхода в ширину

Далее программа предложит сохранить матрицу смежности и результат обхода в файл. Чтобы осуществить сохранение, пользователь должен будет ввести «Да» или «да». После чего произойдет сохранение и переход в стартовое меню.

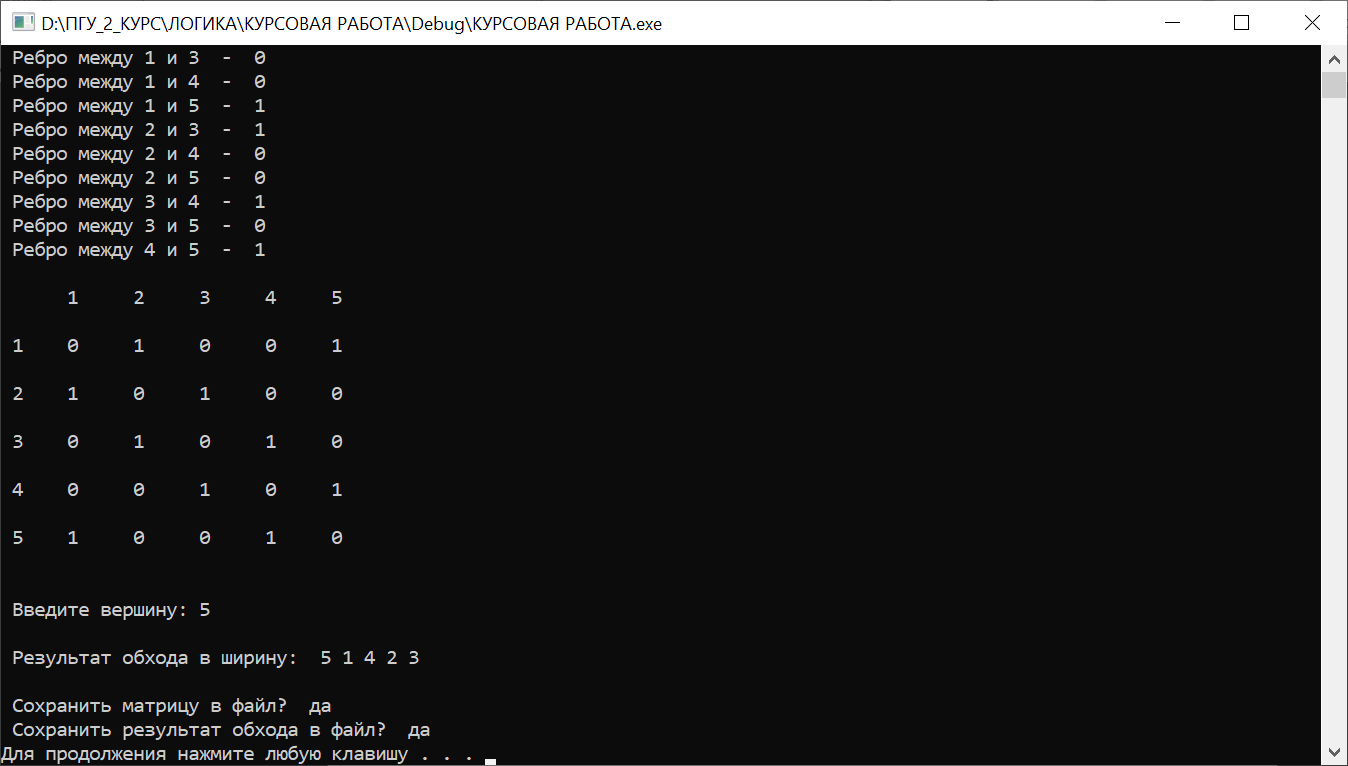
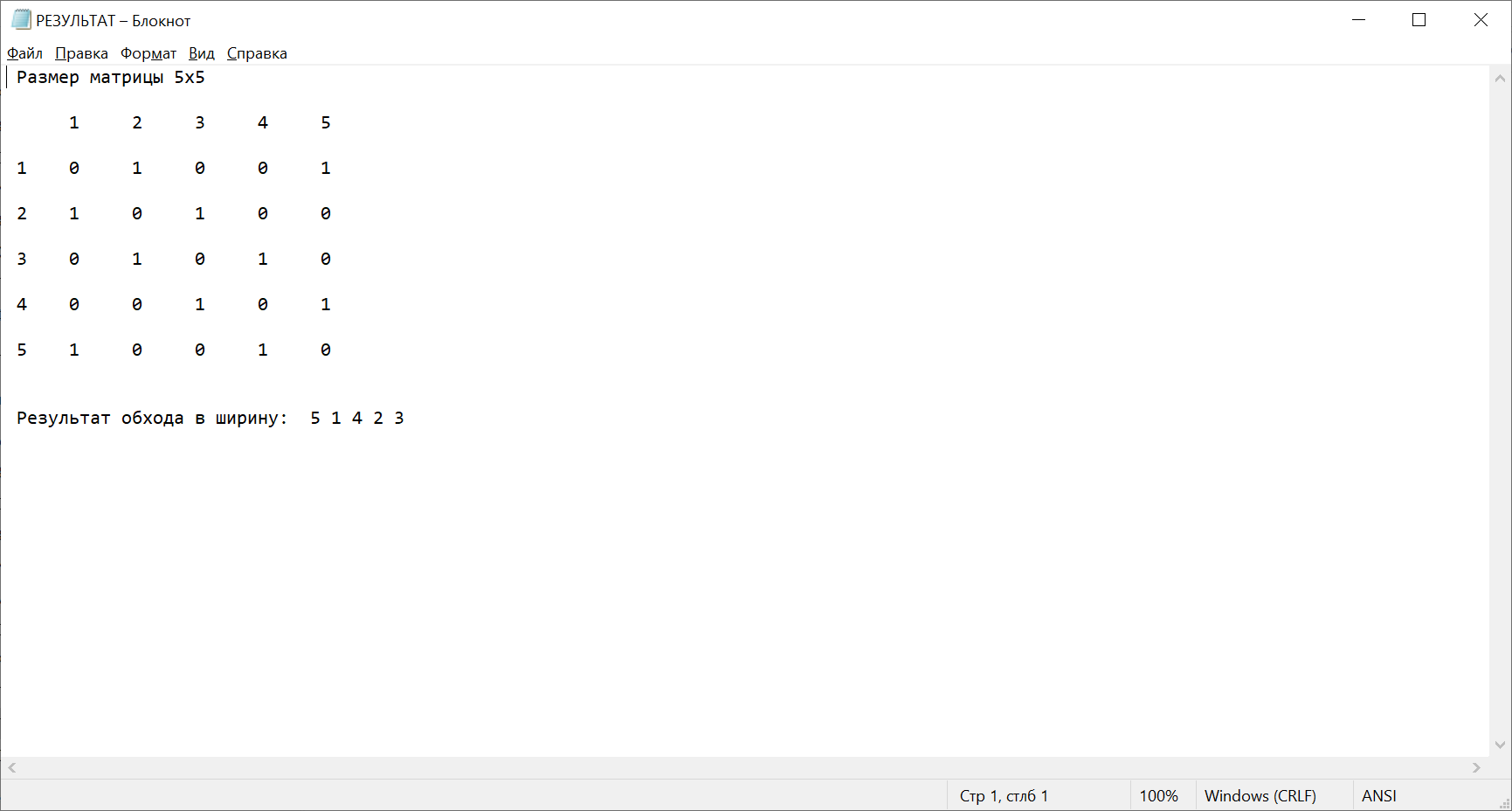
 

Рисунок 7 – Результат сохранения

При выборе пункта 2 в стартовом меню программа завершит работу.

# 

# Тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Ожидаемый результат | Выполненное действие | Полученный результат |
| Запуск программы | Вывод начального меню | Запустить файл программы тип .exe | Верно |
| Создание с помощью функции автоматического заполнения графа из 5 вершин | Вывод графа размера 5х5, который был создан с помощью функции автоматического заполнения | В главном меню выбрать «Начать работу».  Ввести количество вершин -5.  Выбрать пункт 2 – «Автоматическое заполнение» | Верно |
| Создание с помощью функции ручного заполнения графа из 4 вершин | Вывод графа размера 4х4, который был создан с помощью функции ручного заполнения | В главном меню выбрать «Начать работу».  Ввести количество вершин -4.  Выбрать пункт 2 – «Ручное заполнение»  Вводить вершины. | Верно |
| Сохранение созданного графа в файл | Обнаружение созданного графа в файле | После вывода результата обхода программой будет предложено сохранить матрицу в файл. Чтобы сохранить ее, введите слово «да» или «Да». | Верно |
| Обход в ширину | Вывод результата обхода в ширину | После вывода матрицы на экран выбрать начальную вершину для обхода. | Верно |
| Сохранение результата поиска в файл | Обнаружение полученного результата в файле | После предложения сохранить матрицу программой будет предложено сохранить резульат рабюоты в файл. Чтобы сохранить, введите слово «да» или «Да». | Верно |
| Выход из программы | Завершение работы программы | Выбрать «Завершить работу» в главном меню. | Верно |

Таблица 2 – План тестирования

Среда разработки Microsoft Visual Studio 2019 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, алгоритмом программы, взаимодействием функций, а также с правильностью и неправильностью выводимых данных.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных данных.

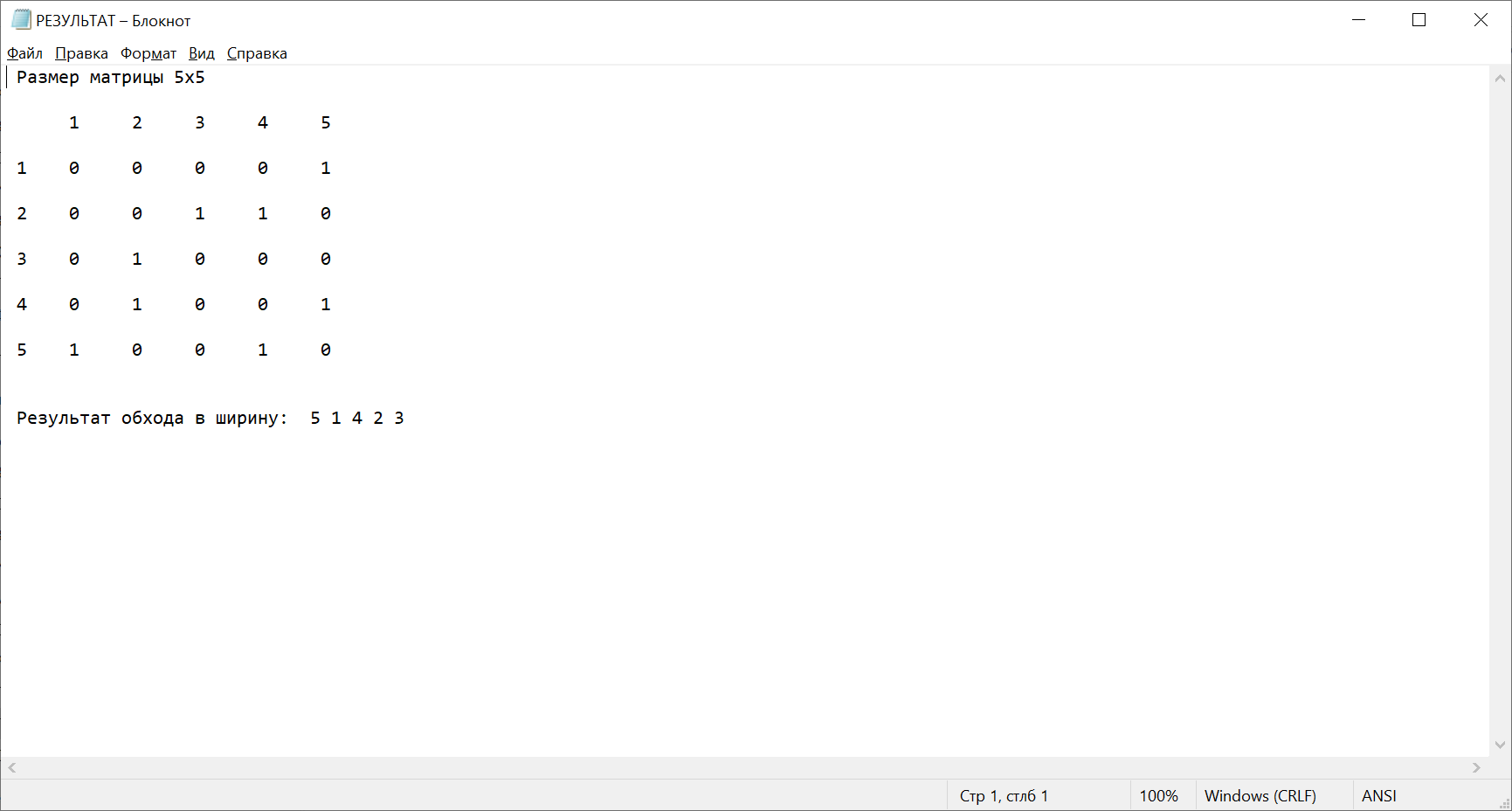
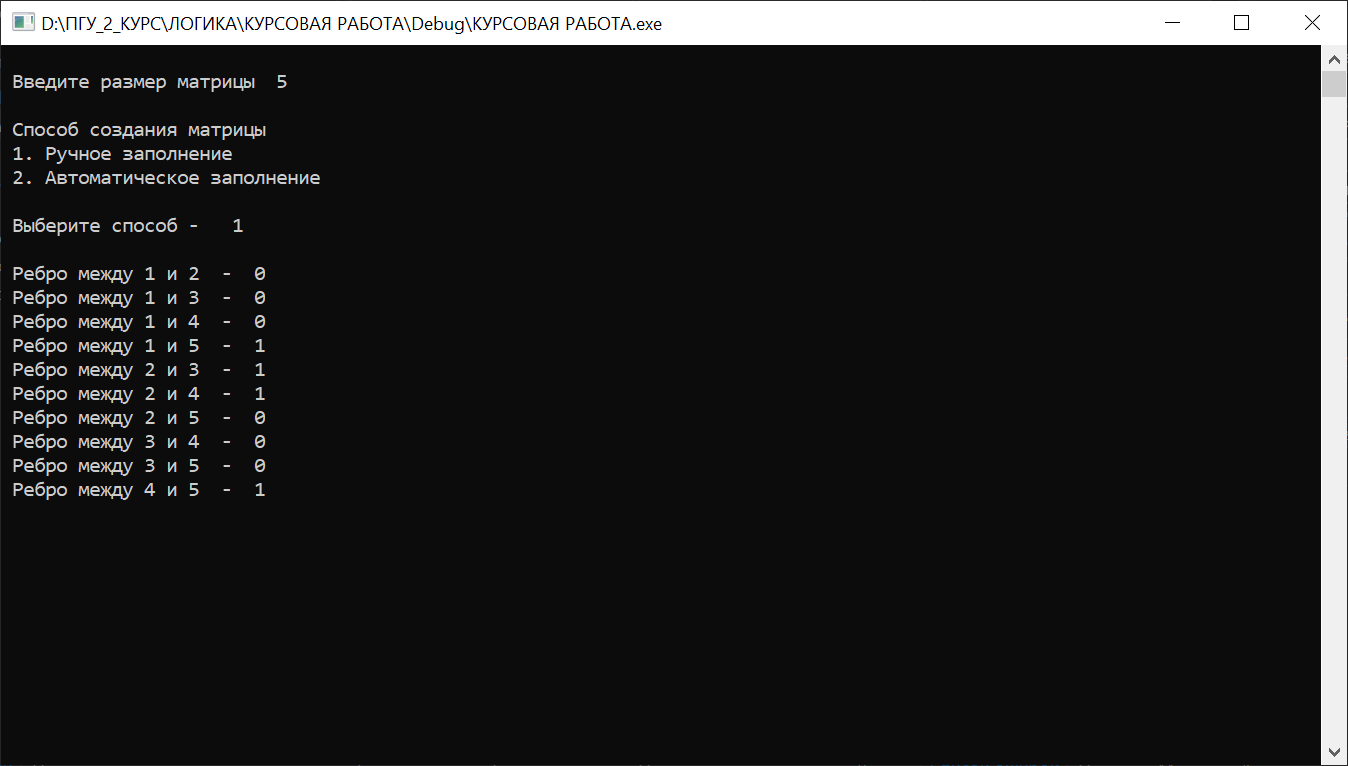


Рисунок 8 – Граф из 5 вершин, созданный вручную, матрица и результат сохранены в файл

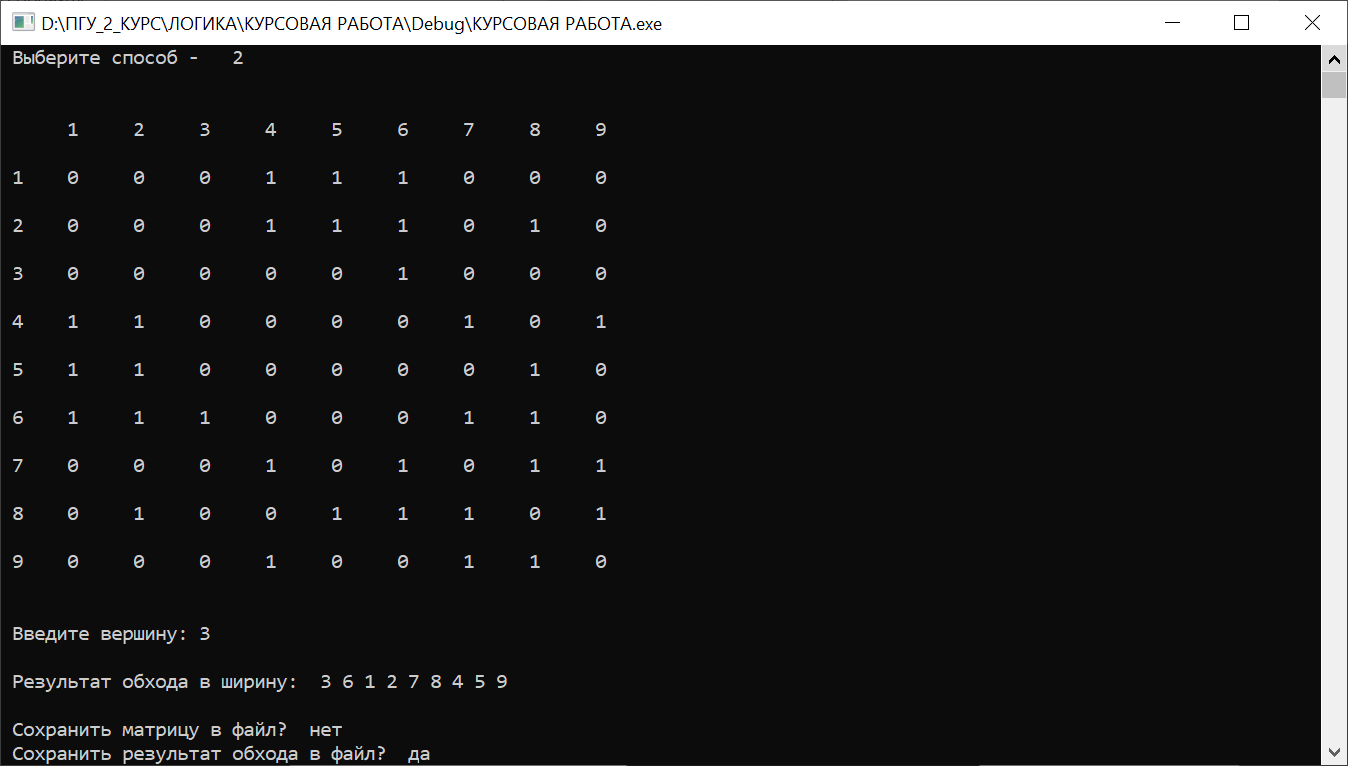
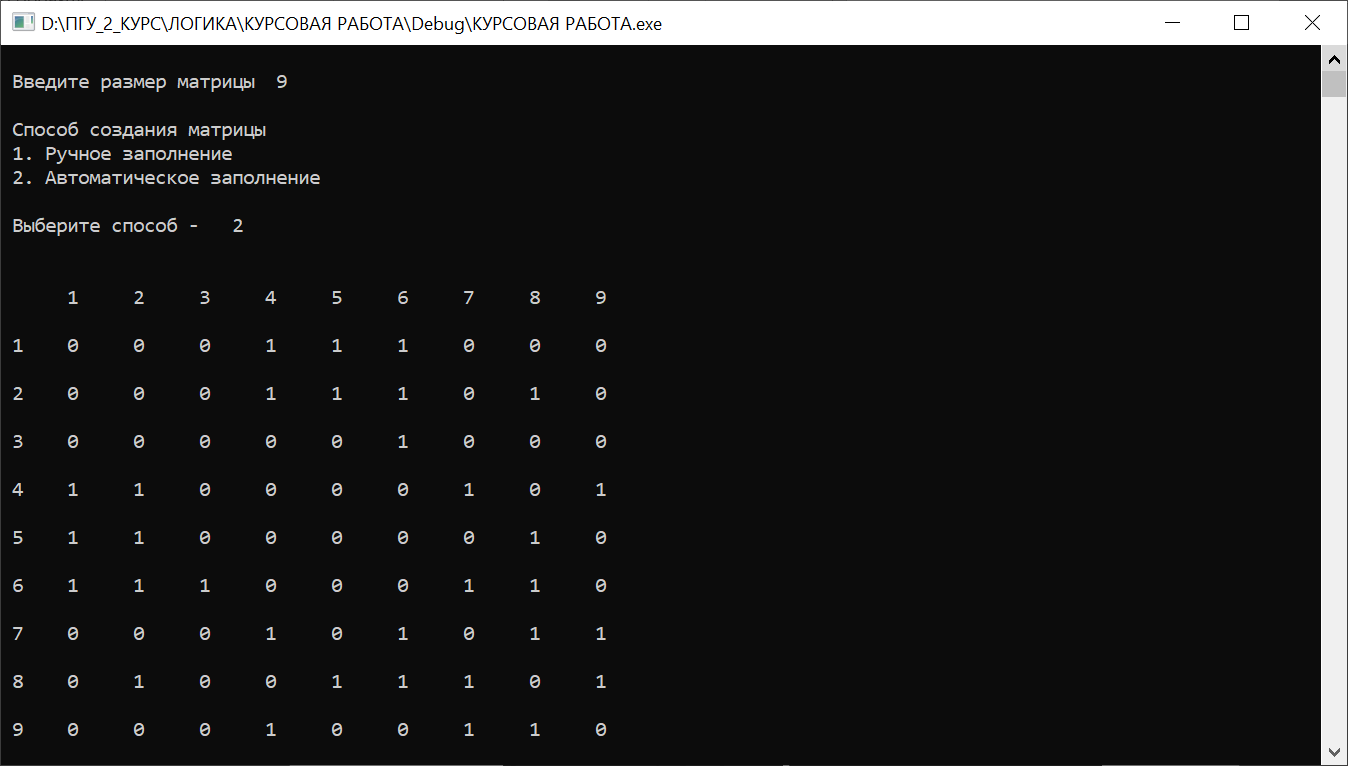


Рисунок 9 – Граф из 9 вершин, созданный автоматически, только результат сохранен в файле

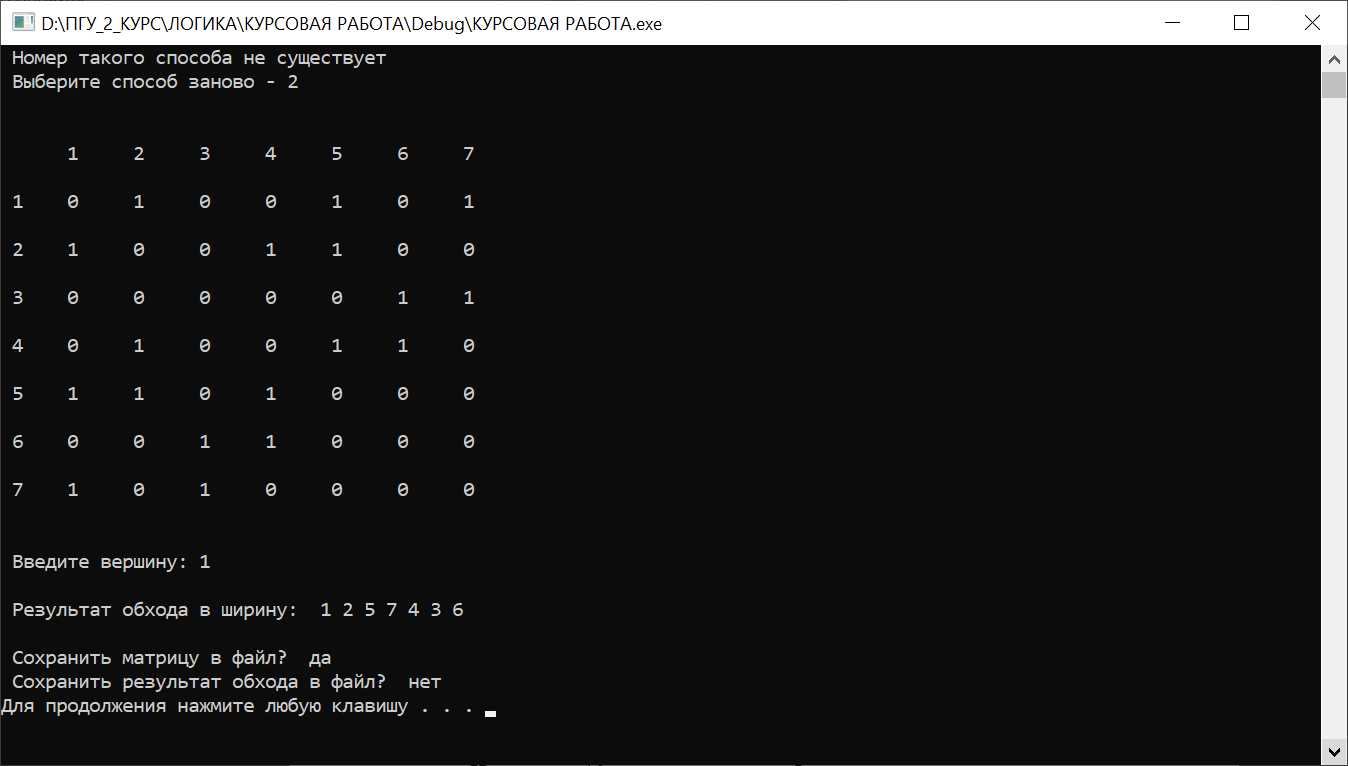
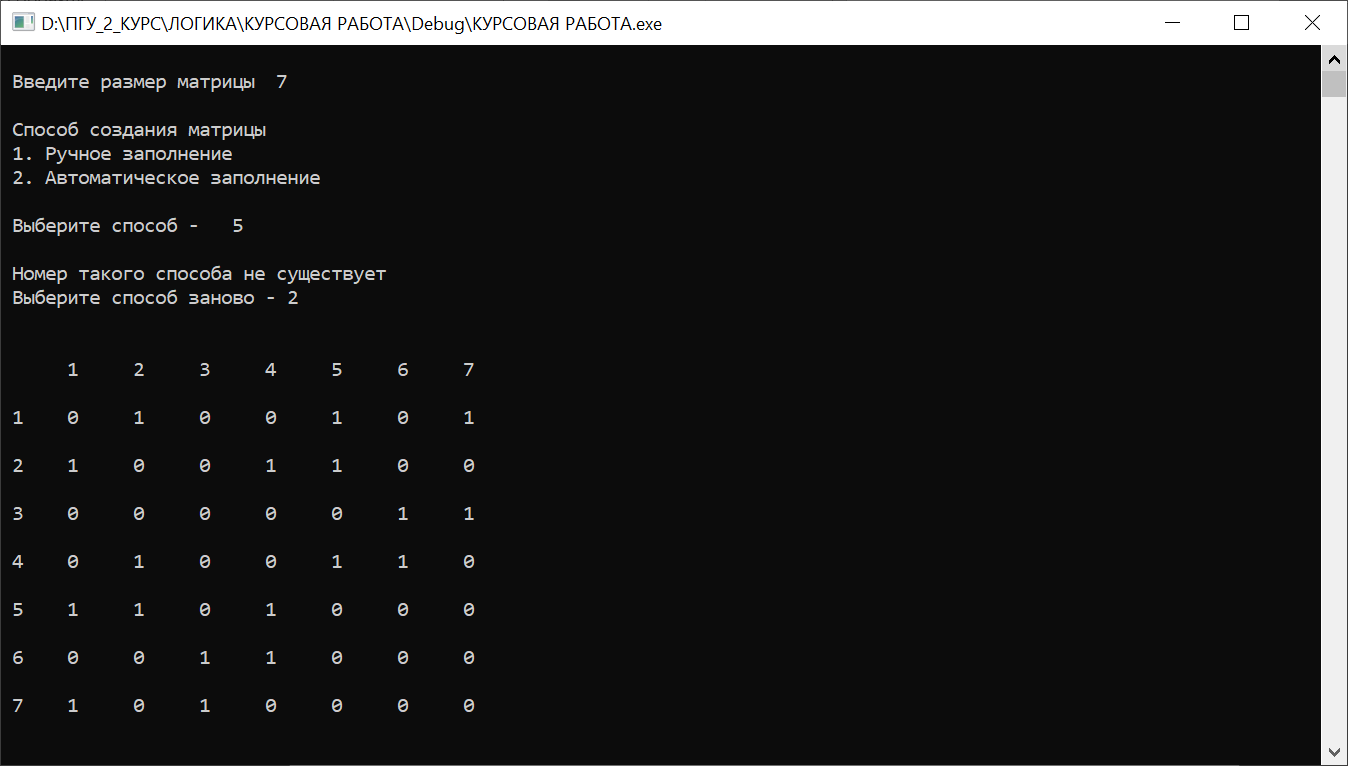


Рисунок 10 - Граф из 7 вершин, созданный автоматически, только матрица была сохранена в файл

На рисунке 8 приведен пример выбора несуществующей вершины для реализации обхода в ширину и реакция программы на это.

На рисунке 10 показан пример выбора несуществующих способа генерации графа и то, как программа реагирует на это.

# Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 9 вершинами. Граф создадим автоматически.

Начальная вершина – 3.

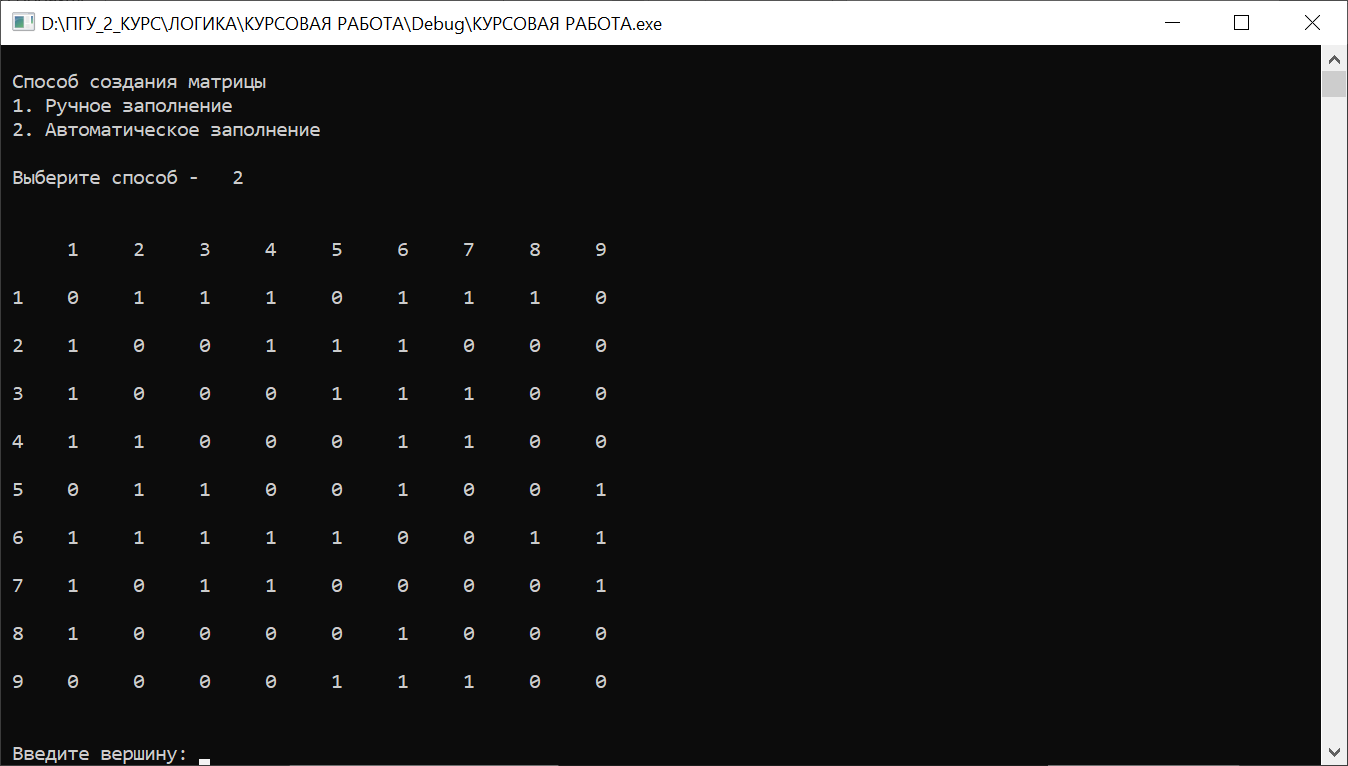


Рисунок 11 – Пример матрицы для ручного расчета

Стартовая вершина – 3. Данная вершина смежна с 1, 5, 6, 7. Запишем порядок 3-1-5-6-7. Отмечаем 1, 5, 6 и 7 как посещенные.

Переходим к вершине 1. Она смежна с 2, 3(уже посещена), 4, 6(уже посещена), 7(уже посещена), 8. Приписываем к прошлому порядку 3-1-5-6-7-2-4-8. 2, 4 и 8 отмечаем как посещенные.

Переходим к вершине 5. Она смежна с 2(уже посещена), 3(уже посещена), 6(уже посещена), 9. Приписываем к прошлому порядку 3-1-5-6-7-2-4-8-9. Отмечаем 9 как посещенную.

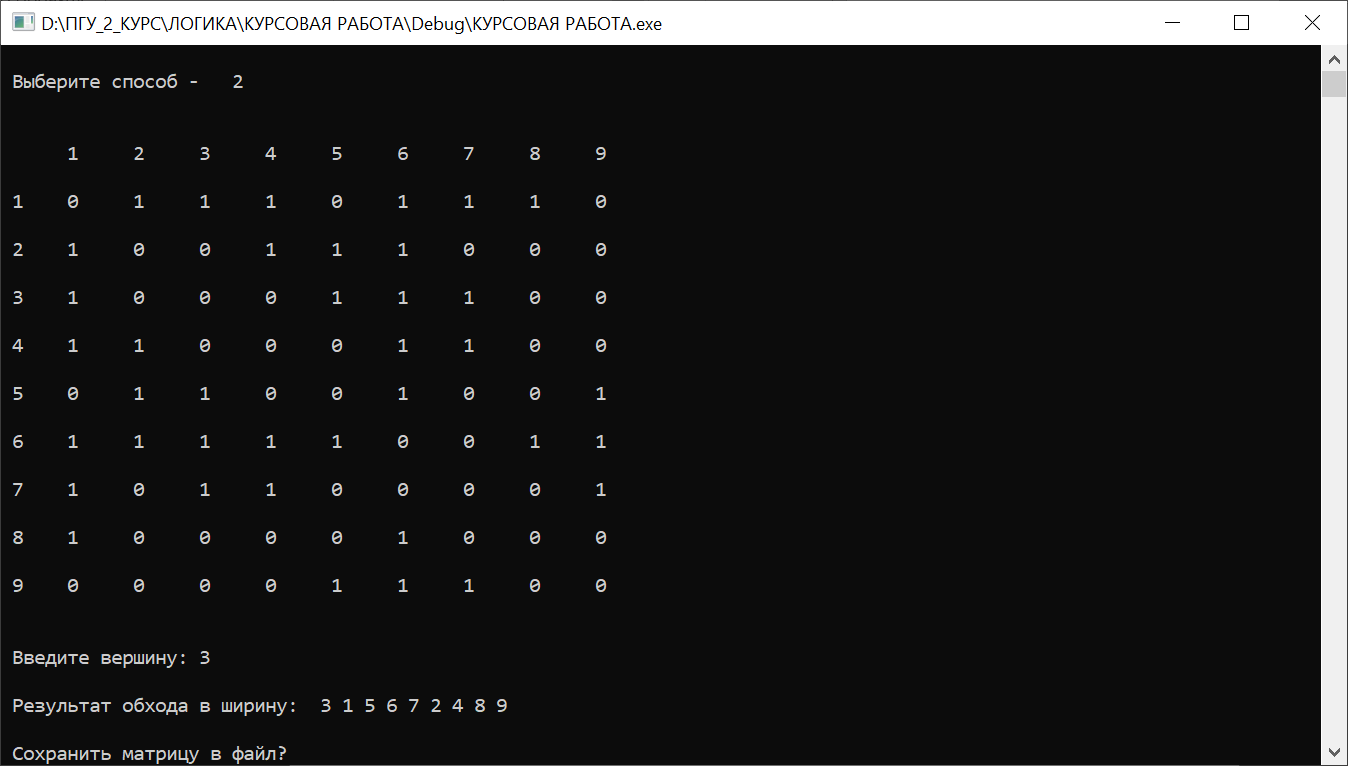
Все вершины посещены. Результат обхода в ширину 3-1-5-6-7-2-4-8-9

Рисунок 12 – Результат работы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм обхода в ширину в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории графов. Углублены знания языков программирования Cи и Си++.

Недостатком разработанной программы является примитивный пользовательский интерфейс.

# 

# Список использованных источников

1. Лекции
2. https://habr.com
3. http://cppstudio.com
4. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978

## Приложения

## Приложение 1. Листинг программы

## 

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <conio.h>

#include <time.h>

#include <stack>

#include <queue>

#include <stdio.h>

#include <Windows.h>

#include <stdlib.h>

using namespace std;

queue <int> Q;

int i, j, N, st, menu, mean;

char save\_matr[3], save\_result[3];

char Yes[] = "Да", yes[] = "да";

int\*\* graph; // матрица смежности

int\* visited; // посщенные вершины

FILE\* result;

void BFS(int st)

{

Q.push(st);

visited[st] = true;

while (!Q.empty())

{

st = Q.front();

Q.pop();

printf("%d ", st + 1);

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

{

if ((graph[st][j] == 1) && (!visited[j]))

{

visited[j] = true;

Q.push(j);

}

}

}

}

void BFS\_SAVE(int st)

{

result = fopen("РЕЗУЛЬТАТ.txt", "a");

fprintf(result, "\n Результат обхода в ширину: ");

Q.push(st);

visited[st] = true;

while (!Q.empty())

{

st = Q.front();

Q.pop();

fprintf(result, "%d ", st + 1);

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

{

if ((graph[st][j] == 1) && (!visited[j]))

{

visited[j] = true;

Q.push(j);

}

}

}

fclose(result);

}

void MENU()

{

system("cls");

printf("\n Меню программы ");

printf("\n 1. Начать работу");

printf("\n 2. Завершить работу");

printf("\n\n Введите номер команды - ");

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

while (menu > 2 || menu < 1)

{

system("cls");

printf("\n Меню программы ");

printf("\n 1. Начать работу");

printf("\n 2. Завершить работу");

printf("\n\n Введите номер команды - ");

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

}

system("cls");

return;

}

void MMM()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("\n Введите размер матрицы ");

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

printf("\n Способ создания матрицы\n 1. Ручное заполнение\n 2. Автоматическое заполнение \n\n Выберите способ - ");

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

while (mean > 2)

{

printf("\n Номер такого способа не существует\n Выберите способ заново - ");

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

}

printf("\n");

srand(time(NULL));

graph = new int\* [N];

visited = new int[N];

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

{

graph[i] = new int[N];

}

if (mean == 1)

{

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

{

for (j = i + 1; j < N; j++)

{

printf(" Ребро между %d и %d - ", i + 1, j + 1);

scanf\_s("%d", &graph[i][j]);

graph[j][i] = graph[i][j];

graph[i][i] = graph[j][j] = 0;

}

}

}

if (mean == 2) {

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

{

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

{

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

graph[i][i] = graph[j][j] = 0;

}

}

printf("\n ");

}

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

{

printf("%4d ", j + 1);

}

printf("\n\n");

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

{

printf(" %d ", i + 1);

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

{

printf("%4d ", graph[i][j]);

}

printf("\n\n");

}

printf("\n Введите вершину: ");

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

while (st > N)

{

printf("\n Такая вершина не существует!\n");

printf("\n Введите вершину: ");

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

}

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

{

visited[i] = false;

}

printf("\n Результат обхода в ширину: ");

BFS(st - 1);

printf("\n\n");

printf(" Сохранить матрицу в файл? ");

scanf\_s("%s", &save\_matr);

if ((strcmp(Yes, save\_matr) == 0) || (strcmp(yes, save\_matr) == 0))

{

result = fopen("РЕЗУЛЬТАТ.txt", "w");

fprintf(result, " Размер матрицы %dx%d \n\n ", N, N);

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

{

fprintf(result, "%4d ", j + 1);

}

fprintf(result, "\n\n");

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

{

fprintf(result, " %d ", i + 1);

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

{

fprintf(result, "%4d ", graph[i][j]);

}

fprintf(result, "\n\n");

}

fclose(result);

}

printf(" Сохранить результат обхода в файл? ");

scanf\_s("%s", &save\_result);

if ((strcmp(Yes, save\_result) == 0) || (strcmp(yes, save\_result) == 0))

{

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

{

visited[i] = false;

}

BFS\_SAVE(st - 1);

}

return;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

do

{

MENU();

switch (menu)

{

case 1:

MMM();

break;

case 2:

return 0;

}

if (menu != 2)

system("pause");

} while (menu != 2);

}