Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «**Поиск Гамильтоновых циклов**»

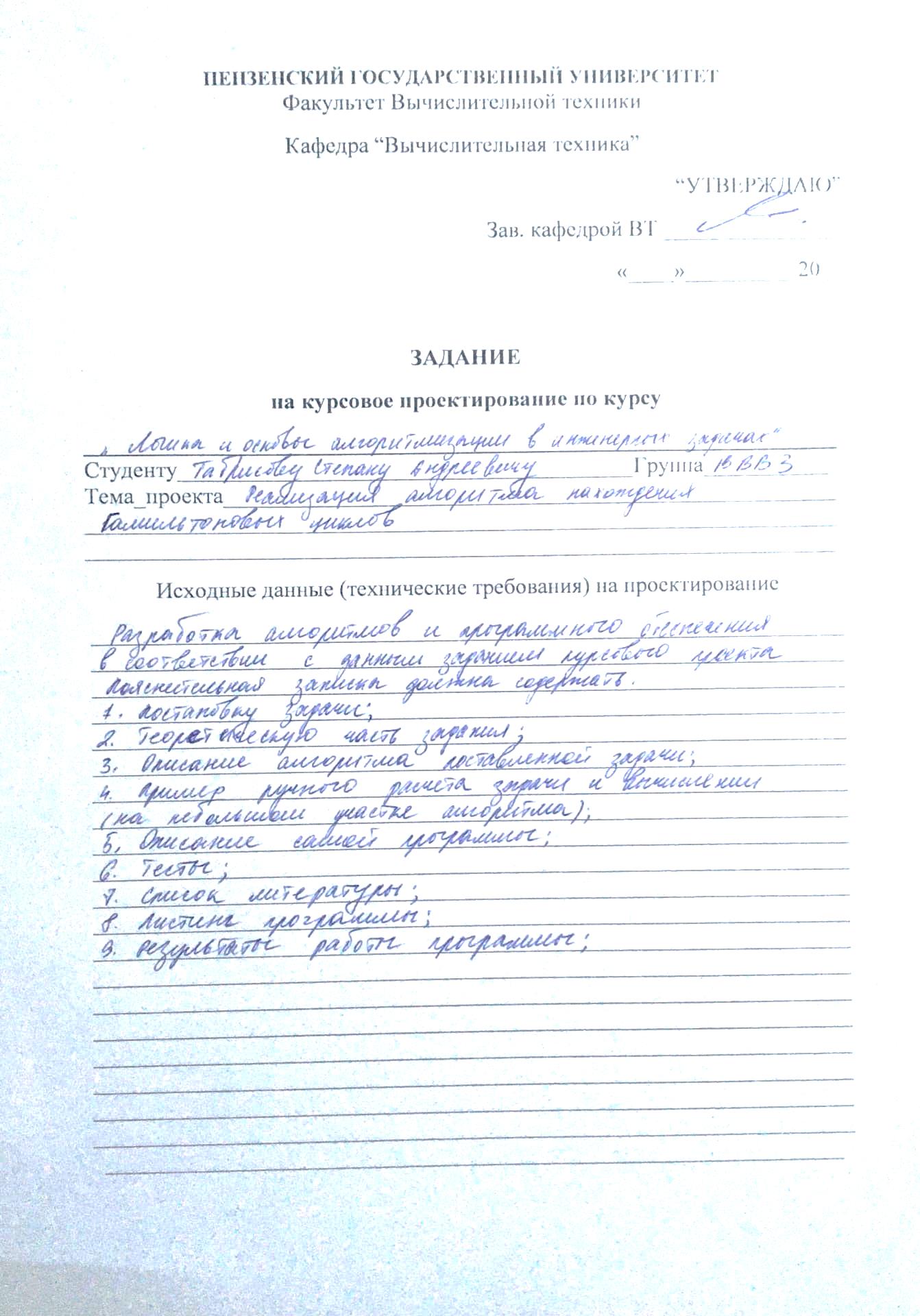
***Выполнил работу студент группы 19ВВ3:***

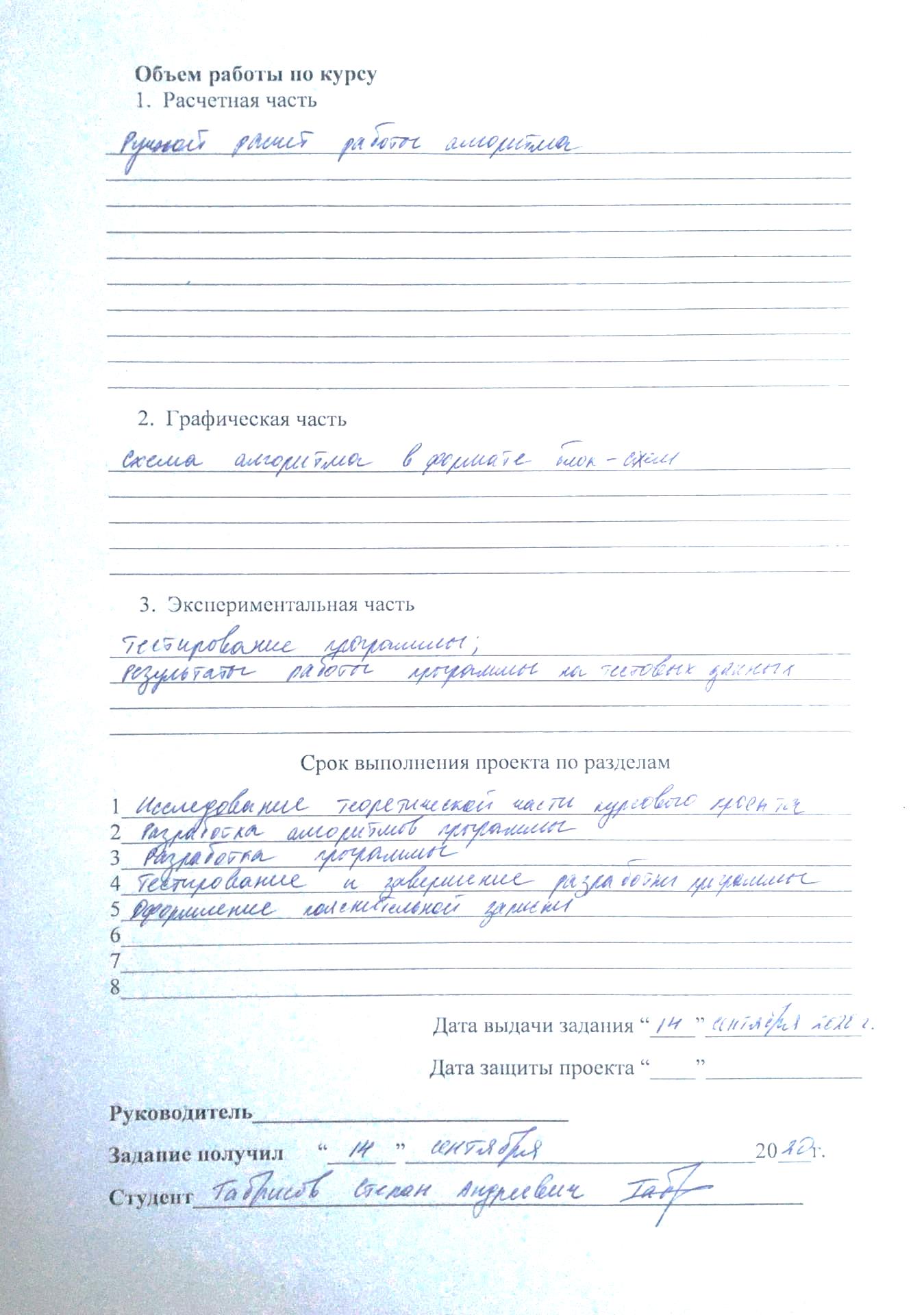
Табрисов С.А.

**Приняли:**

д.т.н. профессор Митрохин М. А.

Пенза 2020





**Содержание**

[Введение 5](#_Toc59775569)

[Постановка задачи 5](#_Toc59775570)

[Теоретическая часть задания 7](#_Toc59775571)

[Описание алгоритма программы 7](#_Toc59775572)

[Описание программы 10](#_Toc59775573)

[Тестирование 18](#_Toc59775574)

[Заключение 27](#_Toc59775575)

[Список литературы 28](#_Toc59775576)

[Приложение А. 29](#_Toc59775577)

**Реферат**

Отчет по курсовой работе: 34 страницы, 21 рисунок, 1 таблица, 1 приложение и 6 источников.

Цель работы – разработка программы для реализации алгоритма поиска Гамильтоновых циклов, используя поиск с возвратом.

Граф, содержащий гамильтонов цикл называется Гамильтоновым графом. При этом гамильтоновым циклом является такой цикл (замкнутый путь), который проходит через каждую вершину данного графа ровно по одному разу; то есть простой цикл, в который входят все вершины графа.

Гамильтонов цикл не обязательно содержит все ребра графа. Ясно, что гамильтоновым может быть только связный граф, который удовлетворяет условию p(i)>2. Где p – степень вершины графа.

# Введение

Задачи на графах находят применение в различных областях, связанных с самым широким спектром математических методов для анализа происходящих там процессов. В настоящее время описаны эффективные методы и алгоритмы решения таких задач, они приводятся в научной и учебной литературе. Разработаны пакеты прикладных программ, позволяющих решать ряд стандартных вопросов, например, задачи о покрытии графов, о раскрасках, об изоморфизме графов. Применение этих пакетов разумно, оправдано и вполне возможно без знания алгоритмов решения задач и технологий, обеспечивающих реализацию алгоритмов. В то же время теоретические исследования позволяют модернизировать и использовать более эффективно разработанные пакеты. В связи с этим целью данной работы является выработка комбинированного подхода к решению задач на графах, использующего как численные схемы и методы, так и теоретические положения. Исследуется вопрос о гамильтоновости неориентированных кубических графов, являющихся объединениями двух циклов длины n и соединяющих их боковых ребер. Формулируются гипотезы о существовании гамильтонова цикла в заданном кубическом графе. Приводится описание и результаты работы вычислительного алгоритма для проверки сформулированных гипотез.

# Постановка задачи

Требуется разработать программу для нахождения Гамильтоновых циклов в графе. Особенностью моего программного продукта является заполнение двумерной матрицы случайными числами, считывания из файла и сохранение результатов в файл.

Исходный граф невзвешенный неориентированный. Задается матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил вероятность появления ребра между вершинами и количество вершин для генерации матрицы смежности. Далее матрица проверяется условие существования Гамильтонова цикла. И если условие истинно, то есть каждая вершина имеет степень больше 2, то на экран должна выводиться первоначальная матрица для последующего поиска циклов. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Целью данной курсовой работы является разработка программы в MicrosoftVisualStudio2019.

Устройство ввода – клавиатура и мышь.

# 

# Теоретическая часть задания

Пусть граф G задается множеством вершин X1, X2, ..., Xn. И множеством ребер, соединяющих между собой определенные вершины. Ребра не имеют направлений, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется неориентированным графом. При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где наличие пути между вершинами обозначается единицей, иначе нулем. В основе моей программе 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.

Пусть задан граф G 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, за исключением вершины X1, т.к. она уже была посещена. Так, продолжая обходить массив и, проверяя, была ли посещена вершина, алгоритм обойдет все доступные из X1 вершины графа G. Алгоритм прекращает свою работу после обхода всех вершин графа при условии, что последняя вершина смежная с первой, либо в случае невыполнения этого условия.

# Описание алгоритма программы

Для программной реализации алгоритма понадобится один массив path для хранения информации о вершинах, включенных в цикл. Итак, имеется граф G=(v, E). Каждая из вершин входящих во множество v, изначально отмечена как не пройденная, т. е. элементам массива path присвоено значение -1.

В качестве исходного пункта помещаем 0 как первую вершину path = 0. Проверяем, является ли вершина последней добавленной. Далее, пробуются различные вершины в Гамильтонов цикл. И прежде чем добавить следующую вершину в путь, проверяется, является ли она смежной с предыдущей вершиной и была ли она уже включена в цикл. Далее, если условия истины, то в массив path заносим найденную вершину. После этого просматриваем вершины дальше таким же способом. И если к построенному пути нельзя добавить вершину, то программа возвращает false и на экран выводится соответствующее сообщение.

Ниже представлен псевдокод функции ham\_Cycle (), check\_vertex () и частично

main().

Ham\_cycle()

1. Если pos равно count
2. Если graph[path[pos – 1]][path[0]] равно 1
3. То возвращаем true
4. Иначе
5. Возвращаем false
6. Конец условия
7. Для v=1 пока v < count делать v++
8. Если check\_vertex(v, graph, path, pos)
9. path[pos] = v
10. Если ham\_Cycle(graph, path, pos + 1, count) равно true
11. То возвращаем true
12. Иначе path[pos] = -1
13. Конец условия
14. Конец цикла
15. Возвращаем false

check\_vertex()

1. Если graph[path[pos - 1]][v] равно 0
2. Возвращаем false
3. Для i=1 пока i < pos делать i++
4. Если path[i] равно v
5. То возвращаем false
6. Конец условия
7. Возвращаем true
8. Конец цикла

main()

1. path[0]=0
2. Если ham\_Cycle(arr, path, 1, num) равно false
3. printf("\nГамильтонов цикл не найден!")
4. test1 = true
5. Возвращаем false
6. Конец условия
7. print\_result(path, num)

# 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++), и является многомодульным, поскольку состоит из нескольких функций.

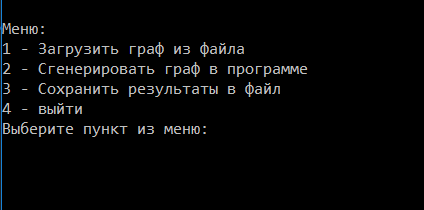
Работа программы начинается с того, что пользователь выбирает, как ему задать граф (сгенерировать в программе случайно или загрузить граф из файла).

printf("\nМеню:\n");

printf("1 - Загрузить граф из файла\n2 - Сгенерировать граф в программе\n3 - Сохранить результаты в файл\n4 - выйти\n");

printf("Выберите пункт из меню: ");

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

  
Рисунок 1.- Главное меню

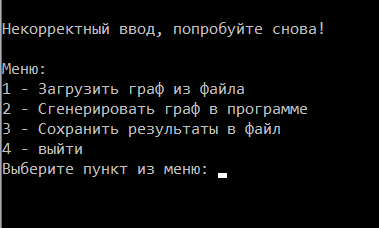
После того как пользователь ввел пункт из меню, программа проверяет правильность их ввода.

if ((MENU != 1) && (MENU != 2) && (MENU != 3) && (MENU != 4)) {

system("cls");

printf("\nНекорректный ввод, попробуйте снова!\n");

}

  
 Рисунок 2. – Некорректный ввод пункта меню.

Если пользователь выбрал сгенерировать матрицу смежности, то на экран выводится запрос на количество вершин в графе. Затем выводятся запрос на ввод вероятности присутствия ребра между вершинами. Если пользователь правильно указал нужные параметры, то на экран выводится сгенерированная матрица.

printf("Введите порядок матрицы: ");

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

printf("Введите вероятность появления ребра между вершинами: ");

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

arr = dynamic(size);

generate\_matrix(arr, size, chance);

print(arr, size);

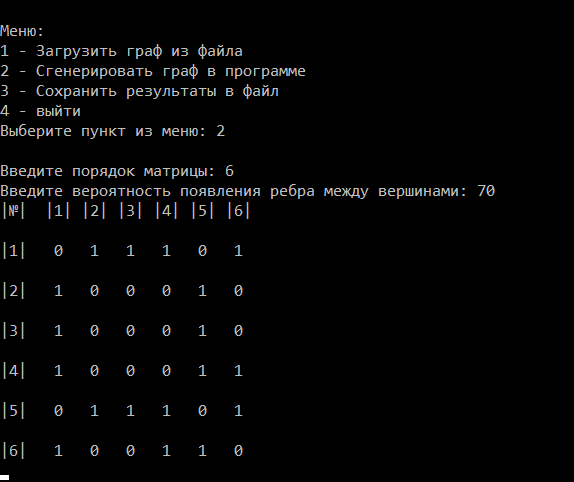


Рисунок 3. – Вывод сгенерированной матрицы.

Если пользователь выбирает второй пункт основного меню управления, то на экран выводится заранее заданная матрица, которая хранится в файле mass.txt, находящегося в папке с программой. Именно поэтому сначала программа открывает текстовый файл в режиме чтения. Далее программа проверяет содержание файла. Если там присутствуют символы, то на экран выводится сообщение "Неверный формат данных". После этого программа считает количество элементов в файле и записывает их в переменную n, != EOF означает что программа дошла до конца файла. После этого функция rewind переносит указатель на начало файла. Далее программа считает количество вершин полученный матрицы, путем вычисления квадратного корня из переменной n. После этого, программа безошибочно считывает матрицу из файла. Выводится текст «Массив считан из файла». И вызывается функция print отвечающая за вывод матрицы в консоль.

printf("Введите название файла из которого будет произведено чтение значений, не более 50 символов (Пример: mass.txt):");

//gets\_s(name);

scanf\_s("%s", &name, 50);

if ((fp = fopen(name, "r")) == NULL) {

while ((OPEN != '1') && (OPEN != '2')) {

printf("Данный файл не найдет. Убедитесь в правильности ввода названия файла!\n");

printf("Для выхода из программы нажмите - 1\n");

scanf\_s("%c", &OPEN, 1);

if (OPEN == '1') {

system("cls");

exit(1);

}

}

}

if (OPEN == 'A') {

while ((fscanf(fp, "%d", &s) != EOF)) {

v = (fscanf(fp, "%d", &s));

if (v == 0) {

printf("Неверный формат данных! Проверьте файл.\n");

exit(0);

break;

}

}

}

fclose(fp);

fp = fopen(name, "r");

// считывание данных с файла

if (OPEN == 'A') { // если переменная равна 'А' => удалось открыть файл

while ((fscanf(fp, "%d", &s) != EOF)) {

if (!fp) break;

n += 1;

}

rewind(fp); // перематываем файл для повторного чтения

num = sqrt(n);

printf("%d\n", num);

arr = dynamic(num);

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

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

fscanf\_s(fp, "%d", &arr[i][j]);

}

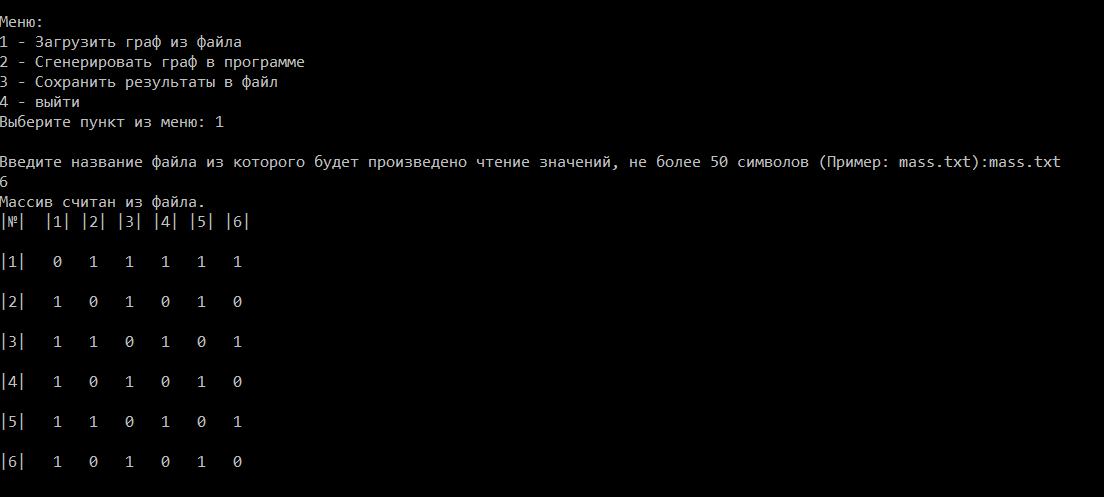
}

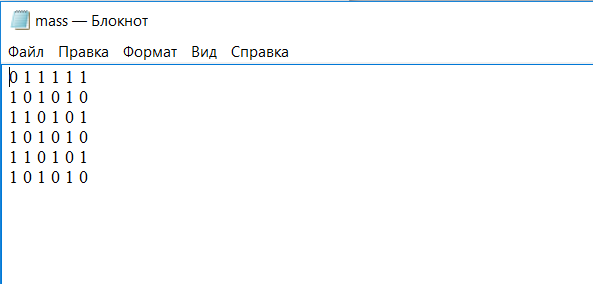
printf("Массив считан из файла.\n");

print(arr, num);

fclose(fp);

}

  
Рисунок 4. – Чтение массива из файла.

  
Рисунок 5. – Исходный массив из файла.

После того, как пользователь выбрал пункт меню, программа выводит матрицу смежности на экран и начинает поиск Гамильтоновых циклов. Но прежде чем вывести матрицу и начать поиск, программа проверяет граф на существование в нем цикла. Если все вершины имеют степень больше 2, то матрица выводится на экран и программа начинает поиск цикла.

pow1 = dynamic\_pow(num);

generate\_pow1(pow1, num);

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

pow\_vertex = 0;

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

if (arr[i][j] > 0) pow\_vertex++;

}

pow1[i] = pow\_vertex;

}

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

if (pow1[i] > 2) chet++;

}

if (chet == num) {

printf("\nВ данном графе возможно существует Гамильтонов цикл\n");

printf("\nОсуществим поиск цикла");

path = dynamic\_path(num);

generate\_path(path, num);

path[0] = 0;

if (ham\_Cycle(arr, path, 1, num) == false)

{

printf("\nГамильтонов цикл не найден!");

test1 = true;

return false;

}

print\_result(path, num);

}

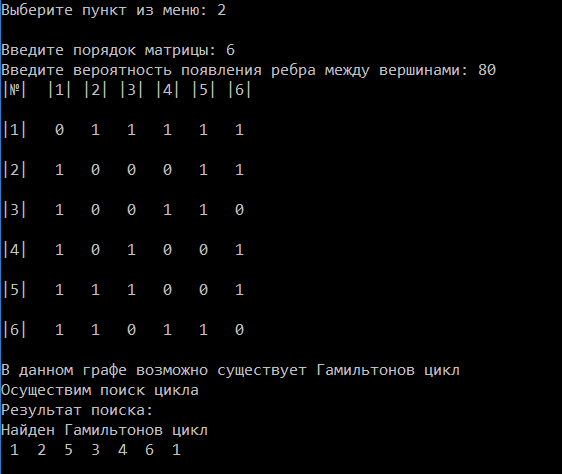
else {

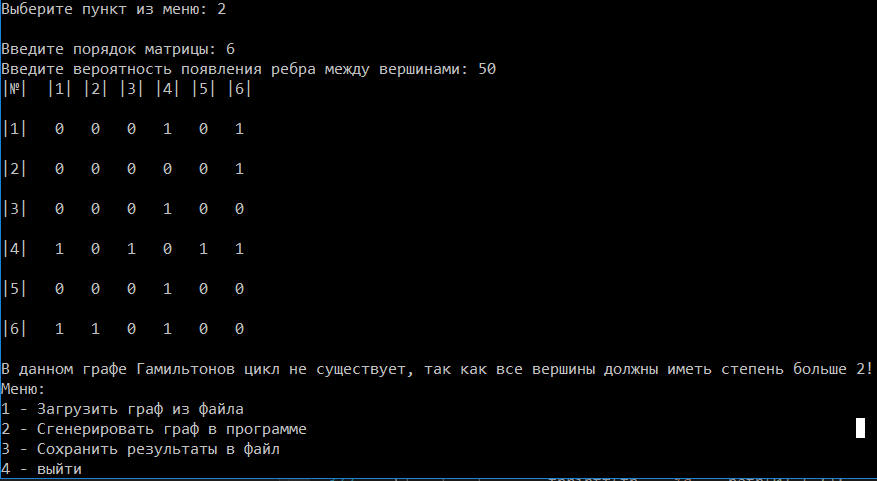
system("cls");

printf("\nВ данном графе Гамильтонов цикл не существует, так как все вершины должны иметь степень больше 2!");

}

chet = 0;

  
Рисунок 6. – Поиск Гамильтнова цикла.

  
Рисунок 7. – Условие существования цикла не выполнено.

Если пользователь перейдет по 3 пункту основного меню, то программа предложит ему сохранить результаты работы в файл. Имя файла пользователь вводит самостоятельно с клавиатуры.

if (MENU == 3) {

if (test == true) {

size = num;

}

printf("Введите название файла, в который хотели бы сохранить таблицу, с указанием типа файла через точку (не более 50 символов): ");

scanf\_s("%s", name\_save, 50);

if ((fp = fopen(name\_save, "w")) == NULL) {

system("cls");

printf("\nНе удалось создать файл!\n");

return 0;

}

fprintf(fp, "Результат работы программы:\n");

for (int k = 1; k <= size; k++) fprintf(fp,"%d ", k);

fprintf(fp,"\n -----------\n");

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

fprintf(fp,"%d |", i + 1);

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

fprintf(fp,"%d ", arr[i][j]);

}

fprintf(fp,"\n");

}

if (test1 == false) {

fprintf(fp, "Результат поиска:"

" Найден Гамильтонов цикл \n");

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

fprintf(fp, " %d ", path[i] + 1);

fprintf(fp, " %d ", path[0] + 1);

fprintf(fp, "\n");

}

else {

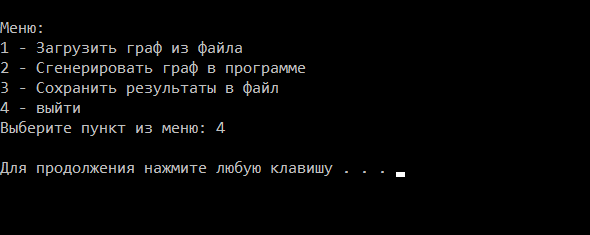
fprintf(fp, "\nГамильтонов цикл не найден!");

}

fclose(fp);

printf("\nРезультаты успешно сохранены в файле - %s\n", name\_save);

Пользователь всегда может завершить работу программы нажатием клавиши 4 или еще раз проследовать по уже реализованным пунктам, изменив какие-либо условия генерации матрицы. Это можно рассмотреть на рисунке .

  
Рисунок 8. – Выход из программы.

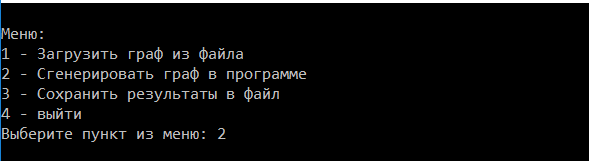
# Тестирование

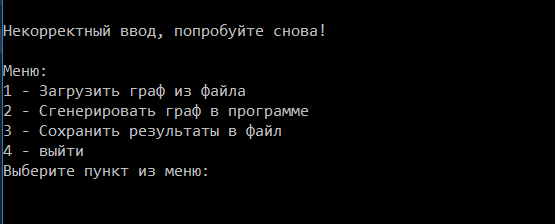
Среда разработки MicrosoftVisualStudio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

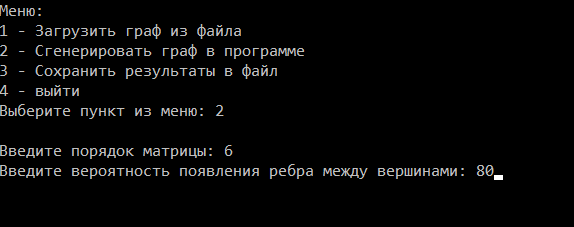
Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом и выводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

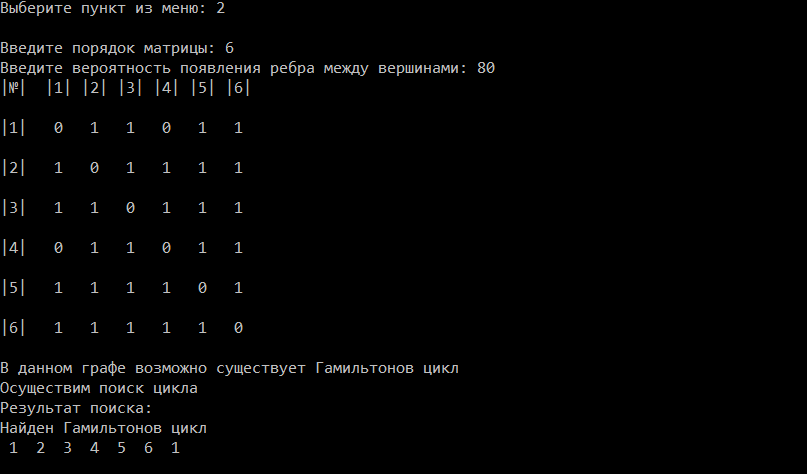
Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и выбор разных пунктов программы.

|  |  |  |
| --- | --- | --- |
| **Описание теста** | **Ожидаемый результат** | **Полученный результат** |
| Запуск программы | Вывод сообщения о выборе:  сгенерировать матрицу или  вывести матрицу из файла (рис. 9) | Верно |
| Ввод неверного пункта меню | Вывод сообщения на экран о некорректном вводе (рис. 10) | Верно |
| Выбор генерации матрицы | Вывод сообщения о  количестве вершин в графе  и вероятности присутствия  ребра между ними (рис. 11) | Верно |
| Генерация матрицы, которая содержит Гамильтонов цикл | Вывод матрицы на экран, проверка условия существования и поиск цикла  (рис. 12) | Верно |
| Генерация матрицы, которая не содержит Гамильтонов цикл | Вывод матрицы на экран, проверка условия существования и вывод соответствующего сообщения на экран (рис. 13) | Верно |
| Вывод матрицы из файла | Вывод самостоятельно заданной матрицы из файла (рис. 18, рис. 19) | Верно |
| Сохранение результата программы в файл | Успешное сохранение исходной матрицы с результатом поиска (рис. 14, рис. 15, рис. 16, рис. 17) | Верно |

  
Рисунок 9. – Запуск из программы.

  
Рисунок 10. – Некорректный ввод.

  
Рисунок 11. – Генерация матрицы размером 6\*6 и вероятностью 0,8.

  
Рисунок 12. – Гамильтонов цикл найден.

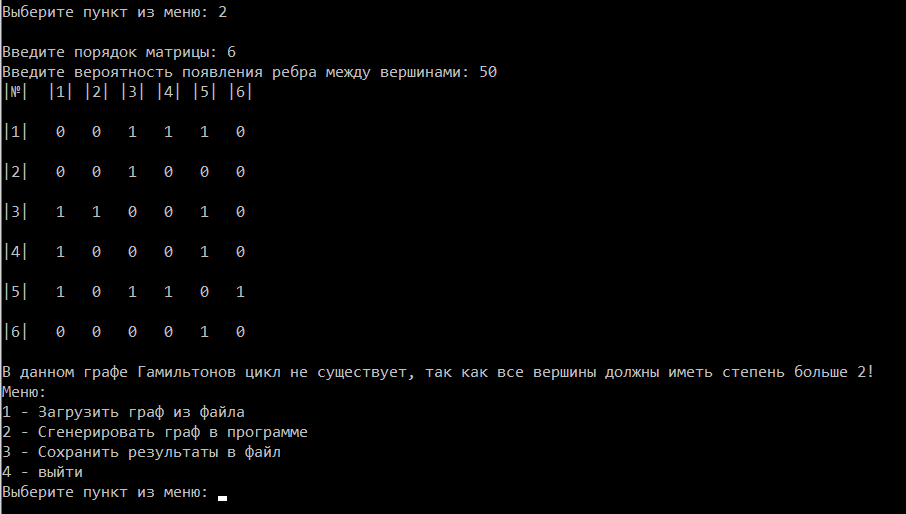
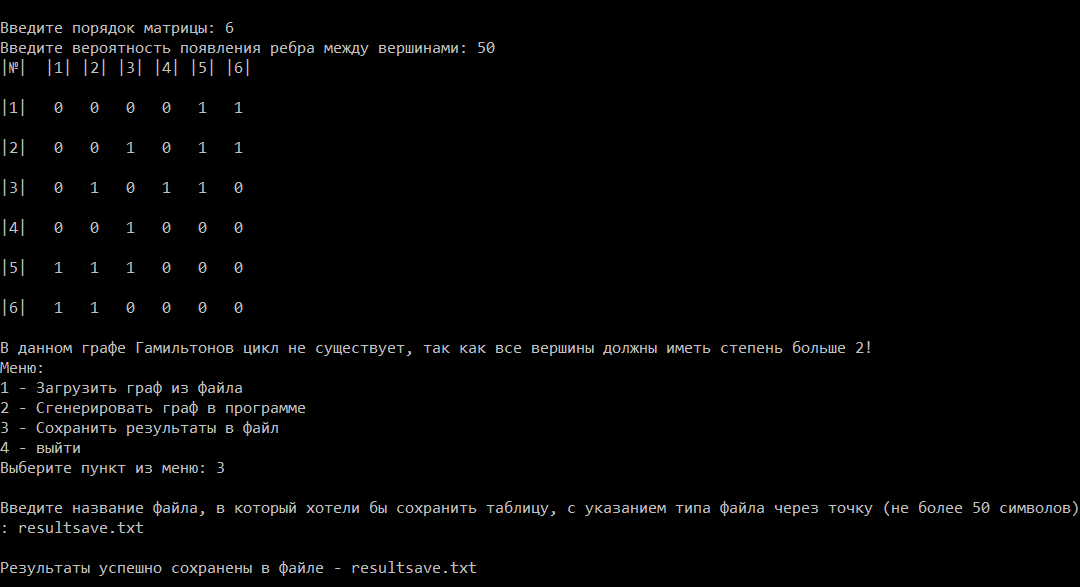
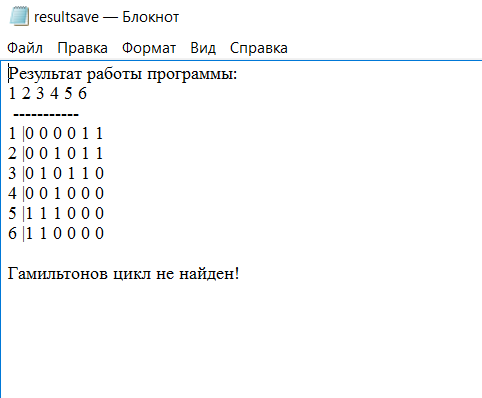


Рисунок 13. – Гамильтонов цикл не найден.

  
Рисунок 14. – Сохранение результатов (если цикл не найден).

  
Рисунок 15. – Файл с результатом работы программы.

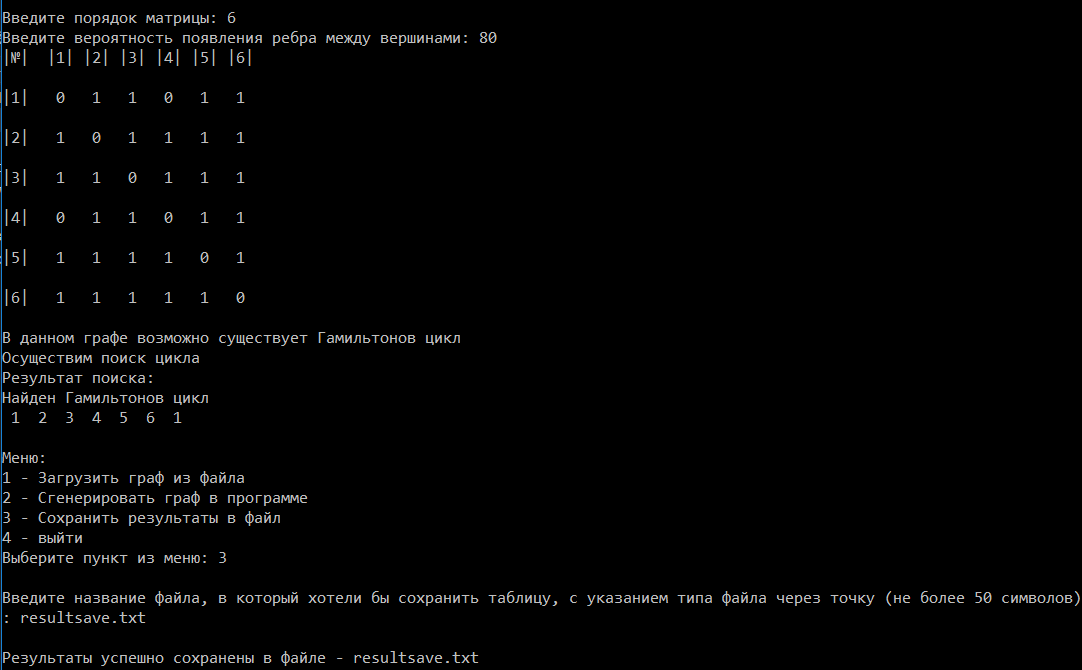
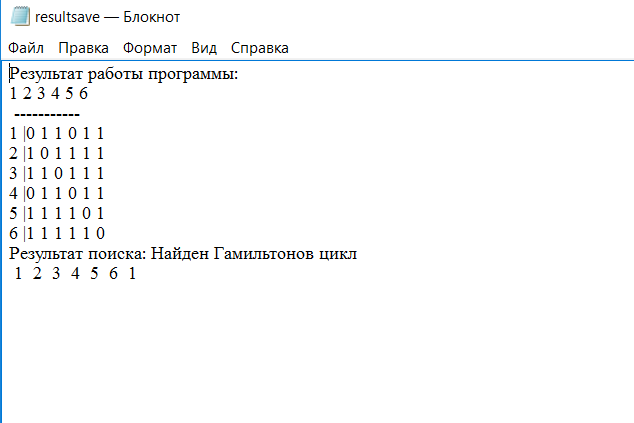
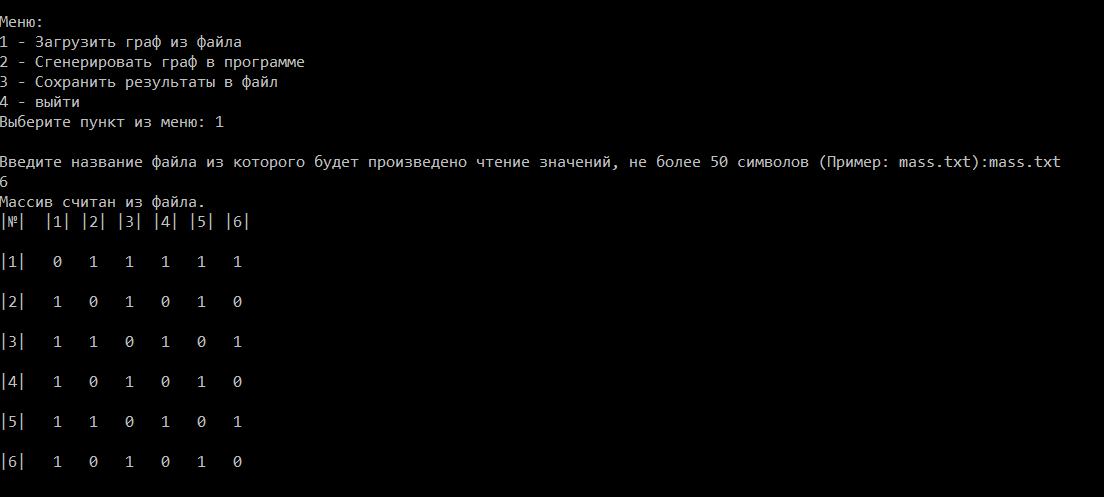
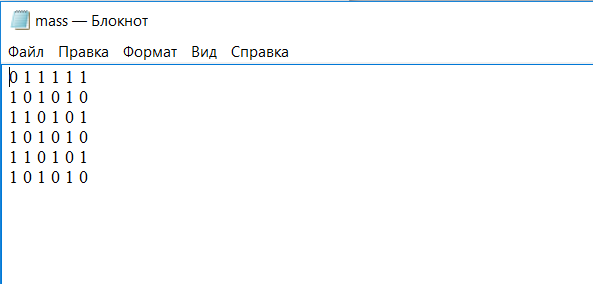


Рисунок 16. – Сохранение результатов (если цикл найден).

  
Рисунок 17. – Файл с результатом работы программы.

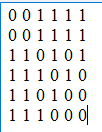
  
Рисунок 18. – Чтение массива из файла.

  
Рисунок 19. – Исходный файл с массивом.

**Ручной расчёт задачи.**

Проведем проверку программы посредством ручных вычислений на

примере графа с 6 вершинами (рис. 20).

  
Рисунок 20. – Ручной расчет.

Начинаем обход из 1 вершины в последующие. Проверяем, если есть путь из 1 в другие вершины, то дальше идем. В нашем случае идем в вершину 3, так как из вершины 3 есть путь в 1 вершину, и она не была еще включена в цикл. Аналогично проверяем и в вершине 3. Из вершины 3 идем в 2. Из вершины 2 в 4. Из вершины 4 в вершину 5. Из 5 вершины мы не можем дальше двигаться, таким образом, цикл не найден. Возвращаемся в вершину 3 и пробуем вместо 2 вершины 4. Далее из 4 в 2. Из 2 в 5 и опять попадаем в тупик. Возвращаемся в 4 вершину и из нее идем в 5. Далее посещаем 2 вершину и из нее 6. Смотрим, есть ли ребро между 6 и 1 вершиной. Видим то что ребро есть и получаем цикл, проходящий через все вершины: 1 3 4 5 2 6 1.

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

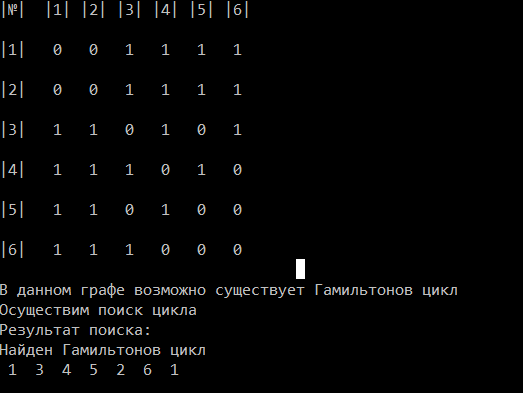


Рисунок 21 – Тестирование работы программы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, которая осуществляет поиск Гамильтоновых циклов в неориентированном графе MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц. Приобретены навыки по осуществлению поиска с возвратом, работы с файлами, а также работы с командной строкой. Углублены знания языка программирования Cи.

Недостатком разработанной программы является примитивный интерфейс, потому что программа работает в консольном режиме.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

**Список литературы**

1. https://www.geeksforgeeks.org/hamiltonian-cycle
2. [https://coderoad.ru](https://vk.com/away.php?to=https%3A%2F%2Fcoderoad.ru%2F13107545%2F%CA%E0%EA-%ED%E0%E9%F2%E8-%C3%E0%EC%E8%EB%FC%F2%EE%ED%EE%E2-%F6%E8%EA%EB-%E2-%E3%F0%E0%F4%E5&cc_key=)
3. https://ru.wikipedia.org
4. Ф. Харари. Теория графов. — М.: УРСС, 2003.
5. [http://www.apmath.spbu.ru/grafomann/book-part1](https://vk.com/away.php?to=http%3A%2F%2Fwww.apmath.spbu.ru%2Fgrafomann%2Fbook-part1-par21.html&cc_key=)
6. Б. Керниган, Д. Ритчи — Язык программирования Си. 2015 г. 352 стр.

## Приложение А.

## Листинг программы.

#include <stdio.h>

#include <tchar.h>

#include <conio.h>

#include <stdlib.h>

#include <windows.h>

#include <string.h>

#include <time.h>

#include <math.h>

int\*\* arr;

int\* save;

FILE\* fp;

int\* path;

int\* pow1;

void print\_result(int\* path1, int count)

{

printf("Результат поиска:"

" \nНайден Гамильтонов цикл \n");

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

printf(" %d ", path1[i] + 1);

// Напечатаем еще раз первую вершину, чтобы показать полный цикл

printf(" %d ", path1[0] + 1);

printf("\n");

}

bool check\_vertex(int v, int\*\* graph, int path[], int pos)

{

if (graph[path[pos - 1]][v] == 0)

return false;

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

if (path[i] == v)

return false;

return true;

}

bool ham\_Cycle(int\*\* graph, int path[], int pos, int count)

{

if (pos == count)

{

if (graph[path[pos - 1]][path[0]] == 1)

return true;

else

return false;

}

for (int v = 1; v < count; v++)

{

if (check\_vertex(v, graph, path, pos))

{

path[pos] = v;

if (ham\_Cycle(graph, path, pos + 1, count) == true)

return true;

path[pos] = -1;

}

}

return false;

}

int\*\* dynamic(int count) {

int\*\* mass;

mass = (int\*\*)malloc(count \* sizeof(int\*));

for (int i = 0; i < count; i++) mass[i] = (int\*)malloc(count \* sizeof(int));

return mass;

}

int\* dynamic\_path(int count) {

int\* path1;

path1 = (int\*)malloc(count \* sizeof(int));

return path1;

}

int\* dynamic\_pow(int count) {

int\* pow\_vertex;

pow\_vertex = (int\*)malloc(count \* sizeof(int));

return pow\_vertex;

}

void generate\_path(int\* path1, int count) {

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

path1[i] = -1;

}

}

void generate\_pow1(int\* massiv, int count) {

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

massiv[i] = 0;

}

}

void generate\_matrix(int\*\* mass, int count, int ver) { //невзвешанный неориентированынй

int chance;

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

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

chance = rand() % 101;

if (chance < ver){

mass[i][j] = 1;

mass[j][i] = mass[i][j];

}

else {

mass[i][j] = 0;

mass[j][i] = mass[i][j];

}

if (i == j) mass[i][j] = 0;

}

}

}

void print(int\*\* mass, int count) {

printf("|№| ");

for (int k = 1; k <= count; k++) printf("|%d| ", k);

printf("\n");

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

printf("\n|%d| ", i + 1);

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

printf("%3d ", mass[i][j]);

}

printf("\n");

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int size = 0;

char name[50];

char name\_save[50];

char OPEN = 'A'; // условие, при котором удалось открыть исходный файл

int MENU;

int n=0;

int s, k = 0;

int v;

int num;

bool test = false;

bool test1 = false;

int pow\_vertex;

int chance = 0;

int chet = 0;

do {

printf("\nМеню:\n");

printf("1 - Загрузить граф из файла\n2 - Сгенерировать граф в программе\n3 - Сохранить результаты в файл\n4 - выйти\n");

printf("Выберите пункт из меню: ");

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

printf("\n");

if ((MENU != 1) && (MENU != 2) && (MENU != 3) && (MENU != 4)) { // Проверка корректности ввода данных

system("cls");

printf("\nНекорректный ввод, попробуйте снова!\n");

}

if (MENU == 1) {

test = true;

printf("Введите название файла из которого будет произведено чтение значений, не более 50 символов (Пример: mass.txt):");

//gets\_s(name);

scanf\_s("%s", &name, 50);

if ((fp = fopen(name, "r")) == NULL) {

while ((OPEN != '1') && (OPEN != '2')) {

printf("Данный файл не найдет. Убедитесь в правильности ввода названия файла!\n");

printf("Для выхода из программы нажмите - 1\n");

scanf\_s("%c", &OPEN, 1);

if (OPEN == '1') {

system("cls");

exit(1);

}

}

}

if (OPEN == 'A') {

while ((fscanf(fp, "%d", &s) != EOF)) {

v = (fscanf(fp, "%d", &s));

if (v == 0) {

printf("Неверный формат данных! Проверьте файл.\n");

exit(0);

break;

}

}

}

fclose(fp);

fp = fopen(name, "r");

// считывание данных с файла

if (OPEN == 'A') { // если переменная равна 'А' => удалось открыть файл

while ((fscanf(fp, "%d", &s) != EOF)) {

if (!fp) break;

n += 1;

}

rewind(fp); // перематываем файл для повторного чтения

num = sqrt(n);

printf("%d\n", num);

arr = dynamic(num);

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

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

fscanf\_s(fp, "%d", &arr[i][j]);

}

}

printf("Массив считан из файла.\n");

print(arr, num);

fclose(fp);

}

pow1 = dynamic\_pow(num);

generate\_pow1(pow1, num);

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

pow\_vertex = 0;

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

if (arr[i][j] > 0) pow\_vertex++;

}

pow1[i] = pow\_vertex;

}

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

if (pow1[i] > 2) chet++;

}

if (chet == num) {

printf("\nВ данном графе возможно существует Гамильтонов цикл\n");

printf("\nОсуществим поиск цикла");

path = dynamic\_path(num);

generate\_path(path, num);

path[0] = 0;

if (ham\_Cycle(arr, path, 1, num) == false)

{

printf("\nГамильтонов цикл не найден!");

test1 = true;

return false;

}

print\_result(path, num);

}

else {

system("cls");

printf("\nВ данном графе Гамильтонов цикл не существует, так как все вершины должны иметь степень больше 2!");

}

chet = 0;

}

if (MENU == 2) {

printf("Введите порядок матрицы: ");

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

printf("Введите вероятность появления ребра между вершинами: ");

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

arr = dynamic(size);

generate\_matrix(arr, size, chance);

print(arr, size);

pow1 = dynamic\_pow(size);

generate\_pow1(pow1, size);

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

pow\_vertex = 0;

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

if (arr[i][j] > 0) pow\_vertex++;

}

pow1[i] = pow\_vertex;

}

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

if (pow1[i] > 2) chet++;

}

if (chet == size) {

printf("\nВ данном графе возможно существует Гамильтонов цикл");

printf("\nОсуществим поиск цикла");

path = dynamic\_path(size);

generate\_path(path, size);

path[0] = 0;

if (ham\_Cycle(arr, path, 1, size) == false)

{

printf("\nГамильтонов цикл не найден!");

test1 = true;

return false;

}

print\_result(path, size);

}

else {

printf("\nВ данном графе Гамильтонов цикл не существует, так как все вершины должны иметь степень больше 2!");

}

chet = 0;

}

if (MENU == 3) {

if (test == true) {

size = num;

}

printf("Введите название файла, в который хотели бы сохранить таблицу, с указанием типа файла через точку (не более 50 символов): ");

scanf\_s("%s", name\_save, 50);

if ((fp = fopen(name\_save, "w")) == NULL) {

system("cls");

printf("\nНе удалось создать файл!\n");

return 0;

}

fprintf(fp, "Результат работы программы:\n");

for (int k = 1; k <= size; k++) fprintf(fp,"%d ", k);

fprintf(fp,"\n -----------\n");

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

fprintf(fp,"%d |", i + 1);

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

fprintf(fp,"%d ", arr[i][j]);

}

fprintf(fp,"\n");

}

if (test1 == false) {

fprintf(fp, "Результат поиска:"

" Найден Гамильтонов цикл \n");

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

fprintf(fp, " %d ", path[i] + 1);

// Напечатаем еще раз первую вершину, чтобы показать полный цикл

fprintf(fp, " %d ", path[0] + 1);

fprintf(fp, "\n");

}

else {

fprintf(fp, "\nГамильтонов цикл не найден!");

}

fclose(fp);

printf("\nРезультаты успешно сохранены в файле - %s\n", name\_save);

}

} while (MENU != 4);

system("pause");

### }