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 нахождения Гамильтоновых циклов»

Выполнил:

студент группы 20ВВ2

Кодиров Искандар

Приняли:

**Пенза 2021ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**Факультет Вычислительной техники

Кафедра “Вычислительная техника”

“УТВЕРЖДАЮ”

Зав. кафедрой ВТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_20\_\_

**ЗАДАНИЕ**

**на курсовое проектирование по курсу**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Студенту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Группа\_\_\_\_\_\_\_\_\_\_\_\_\_  
Тема проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходные данные (технические требования) на проектирование

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Объем работы по курсу**

1. Расчетная часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Графическая часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Экспериментальная часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Срок выполнения проекта по разделам

1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания “\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты проекта “\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Задание получил** “\_\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Содержание**

[Введение 6](#_Toc91217240)

[Постановка задачи 7](#_Toc91217241)

[Теоретическая часть задания 8](#_Toc91217242)

[Описание алгоритма программы 10](#_Toc91217243)

[Описание программы 12](#_Toc91217244)

[Тестирование 14](#_Toc91217245)

[Ручной расчет задачи 19](#_Toc91217246)

[Заключение 20](#_Toc91217247)

[Список литературы 21](#_Toc91217248)

[Листинг программы. 22](#_Toc91217249)

**Реферат**

Отчет 27 страниц, 8 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, НЕОРГРАФ, ГАМИЛЬТОНОВ ПУТЬ, ТЕОРЕМА ДИРАКА.

Цель исследования – разработка программы, способной найти гамильтонов цикла для любого графа.

В работе рассмотрены правила поиска гамильтонова цикла, и создания графа с существующим гамильтоновым циклом используя алгоритм Дирака.

# Введение

Гамильтонов граф — граф, содержащий гамильтонов цикл. 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. Также с гамильтоновым графом тесно связано понятие гамильтонова пути, который является простым путём (путём без петель), проходящим через каждую вершину графа ровно один раз. гамильтонов путь отличается от цикла тем, что у пути начальные и конечные точки могут не совпадать, в отличие от цикла. гамильтонов цикл является гамильтоновым путём.

Гамильтоновы путь, цикл и граф названы в честь ирландского математика У. Гамильтона, который впервые определил эти классы, исследовав задачу «кругосветного путешествия» по додекаэдру. В этой задаче вершины додекаэдра символизировали известные города, такие как Брюссель, Амстердам, Эдинбург, Пекин, Прага, Дели, Франкфурт и др., а рёбра — соединяющие их дороги. Путешествующий должен пройти «вокруг света», найдя путь, который проходит через все вершины ровно один раз. Чтобы сделать задачу более интересной, порядок прохождения городов устанавливался заранее. А чтобы было легче запомнить, какие города уже соединены, в каждую вершину додекаэдра был вбит гвоздь, и проложенный путь отмечался небольшой верёвкой, которая могла обматываться вокруг гвоздя. Однако такая конструкция оказалась слишком громоздкой, и Гамильтон предложил новый вариант игры, заменив додекаэдр плоским графом, изоморфным графу, построенному на рёбрах додекаэдра.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, 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 Гамильтонового цикла.

# Постановка задачи

# 

Требуется разработать программу, которая находит все гамильтоновы циклы в графе.

Исходный граф в программе должен задаваться матрицей смежности. При старте программы пользователь должен выбрать размер и каким образом будет генерироваться граф: случайно или загружаться из текстового файла. После чего программа должна выполнить необходимую функцию, при помощи которого найти все гамильтоновы циклы для каждой вершины. И вывести исходную матрицу и полученный результат на экран.

# 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. Так же граф может быть неориентированным, т.е. ребра двунаправленны, а матрица смежности для такого графа симметрична относительно главной диагонали.

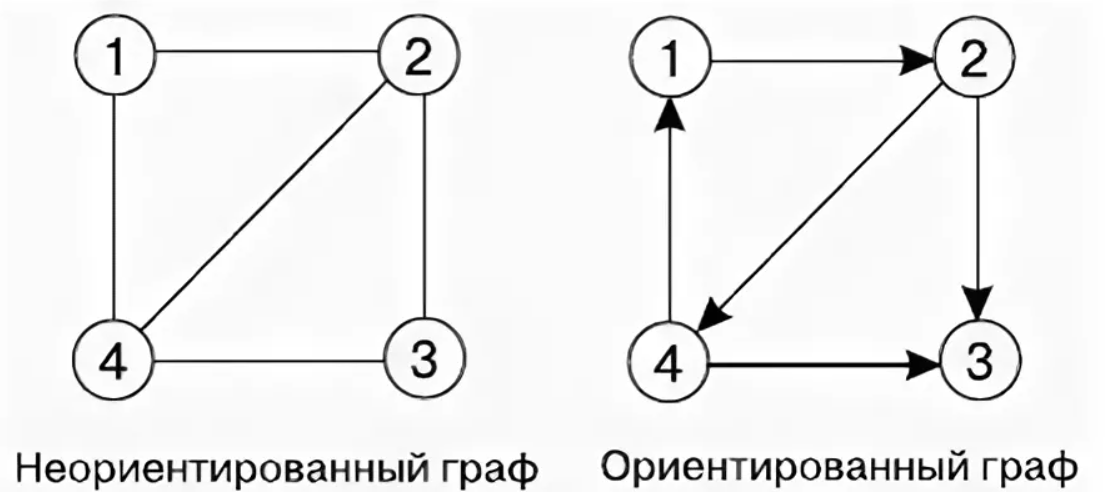


Рисунок 1 – Примеры графов.

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.

Гамильтонов цикл содержит в себе каждую вершину ровно один раз. Если в графе есть такой цикл, то он называется гамильтоновым. Не все графы могут быть гамильтоновыми, но существует пару методов определения гамильтонова графа без перебора всех вершин. Для этого я использую алгоритм Дирака. Если в каждой строке матрицы смежности у каждой вершины будет количество инцидентных ребер не меньше чем n/2, то такой граф всегда будет гамильтоновым.

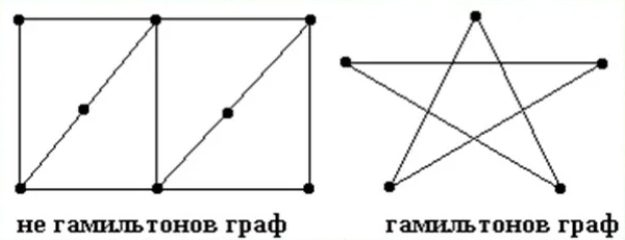


Рисунок 2. Гамильтонов граф

# Описание алгоритма программы

Для программной реализации алгоритма нам понадобится два массива: path(int) – массив в котором будет храниться гамильтонов путь графа, vis(int) – массив в котором хранится информация о посещенных вершинах графа. Итак имеется граф fmatrix(i, v). Каждая из вершин входящая во множество I отмечается как не посещенная, т.е. элементам массива vis пока присвоено значение -1.

В качестве исходного пункта выбирается вершина pos и ей приписывается, что она посещена, вводится переменная success которая отвечает за успешность выполнения алгоритма. В path заносится номер вершины. Далее программа проверяет существование ребра между вершинами. Далее программа проверяет насколько далеко вы уже продвинулись и вернулись ли вы в начальную точку. После того как значение success будет равно 1, программа выполнится и выведется путь path для каждой вершины. Если пока не дошли программа будет проверять каждую вершину по массиву vis проходя каждую вершину он будет заносить номер в прохода в vis, а номер вершины в массив path. Так как из вершин всегда существует несколько путей, то возможно такое, когда достигнута последняя вершина, но не все вершины были пройдены и отмечены. Поэтому алгоритм рекурсивен и будет производить откат к предыдущей вершине, для того чтобы пройти все вершины в графе и занести их в path[i].

Ниже представлен псевдокод функции gamilton()

Функция gamilton:

1. success = 0 признак нахождения пути
2. для i=0 пока i<size делать i++
3. проверить существует ли ребро между вершинами

if (mat[i][path[count - 1]] || mat[path[count - 1]][i])

1. если количество проходов программы уже равно size, а i равна начальной вершине.
   1. Путь найден
2. Если нет
   1. Формируем путь если не были в этой вершине vis[i]==-1
   2. Заносим в vis номер прохода
   3. В path номер вершины
   4. Вызываем функцию gamilton(count+1)
   5. Если были то continue.
3. Конец цикла
4. Возвращаем значение success

Полный код программы можно увидеть в Приложении А.

# Описание программы

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

Данная программа является многомодульной, поскольку состоит из нескольких функций: main, create, createneo, visitor, gamilton, neo, orient, fromfile, printmass, printmatrix.

Работа программы начинается с запроса на ввод пользователем количества вершин графа (размера матрицы смежности). Далее программа предоставляет пользователю выбрать тип графа для работы с ним: 1-ориентированный, 2-неориентированный. Далее пользователь может выбрать откуда взять граф из файла или сгенерировать рандомный.

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

int main()

{

setlocale(0, "");

int x, size;

cout << "Введите размер для графа:";

cin >> size;

cout << "Выберите тип графа:\n1-Ориентированный\n2-Неориентированный\n";

cin >> x;

switch (x)

{

case 1:

cout << "Выберите откуда взять граф:\n1-Из файла\n2-Создать случайный граф"<<endl;

cin >> x;

if (x == 1)

orient\_2(size);

else

orient(size);

break;

case 2:

cout << "Выберите откуда взять граф:\n1-Из файла\n2-Создать случайный граф" << endl;

cin >> x;

if (x == 1)

neo\_2(size);

else

neo(size);

break;

}

return 0;

}

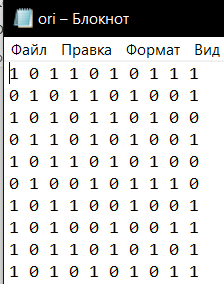


Рисунок 3 – Содержимое "ori.txt"

# Тестирование

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 выводимых данных, алгоритмом программы, взаимодействием функций.

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.).

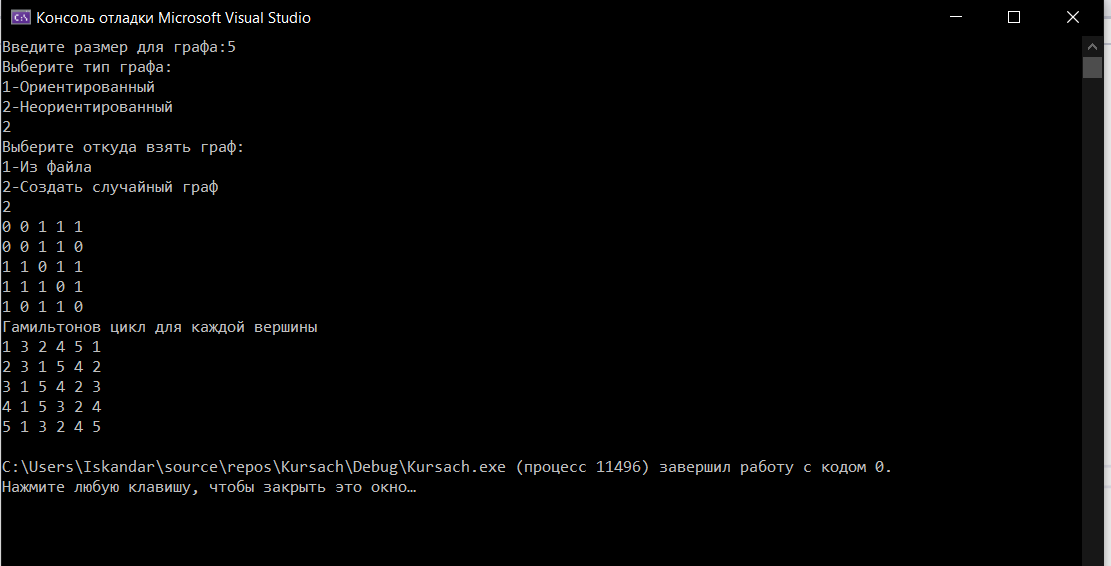


Рисунок 4 – Случайная генерация неор.матрицы размера: 5

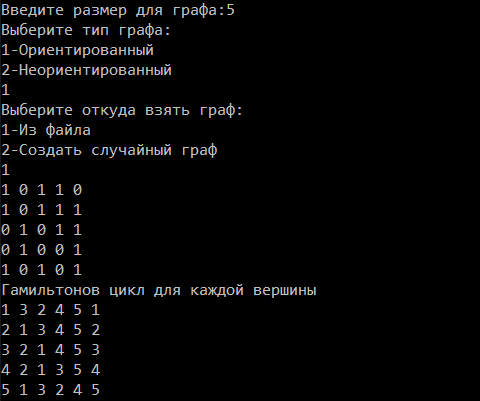


Рисунок 5 – Генерация ор. матрицы размера 5.

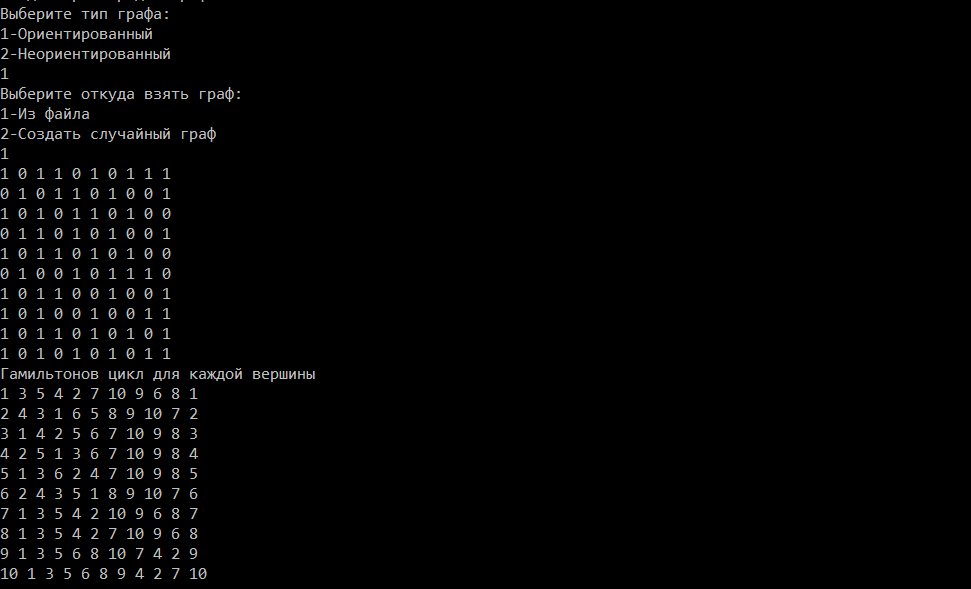


Рисунок 6 – Поиск г. пути для ор. матрицы из файла

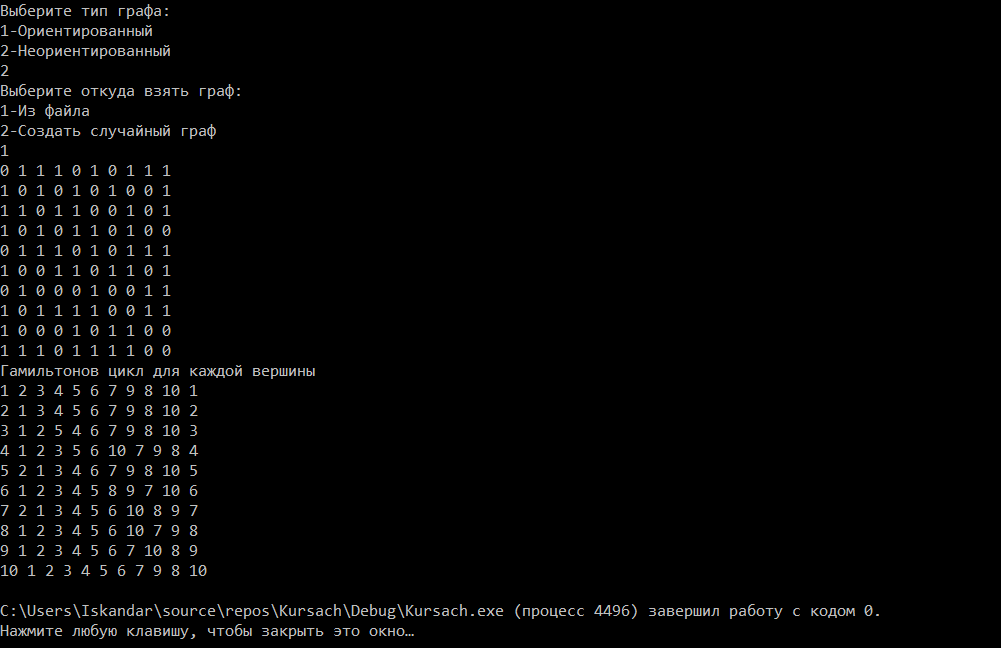


Рисунок 7 – Поиск г. Пути для неор. матрицы из файла

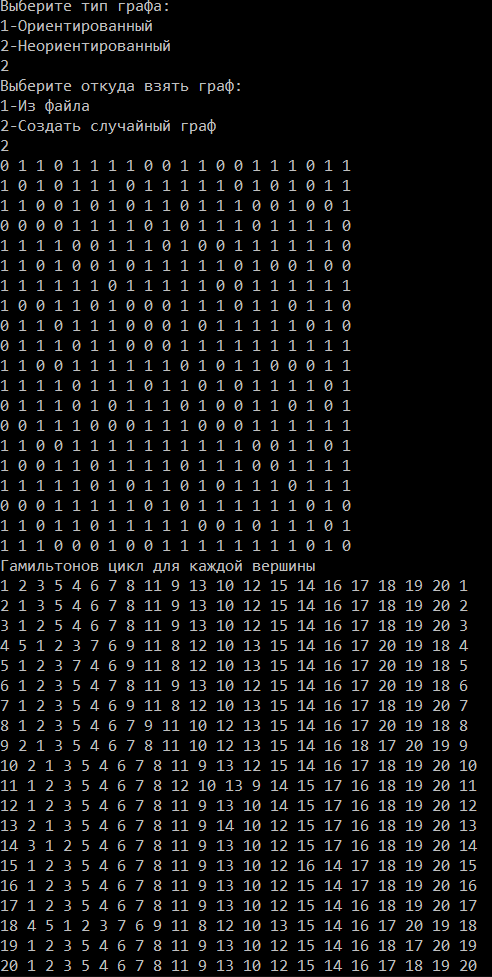


Рисунок 8 – Генерация неор. матрицы размера 20

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы  (рис. 4 – 8) | Вывод запроса у пользователя ввести количество вершин графа | Верно |
| Выбор типа матрицы  (рис. 4 – 8) | Выбор типа матрицы ориентированная/  неориентированная | Верно |
| Выбор загрузки матрицы из файла  (рис. 6, 7) | Загрузка матрицы из заранее подготовленного файла «nem.txt» или «ori.txt» | Верно |
| Выбор генерации матрицы случайным образом  (рис. 4, 5, 8) | Случайная генерация значений матрицы смежности | Верно |
| Нахождение и вывод гамильтоновых циклов для всех вершин  (рис. 4 – рис. 9) | Успешное выполнение программы и вывод всех циклов в консоль | Верно |

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

# Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере случайно сгенерированного графа с 5-ю вершинами(рис. 5).

Берем первую вершину как стартовую. Проверяем есть ли из нее путь в другую вершину или из других в нашу. Стартовая вершина не должна быть изолированной так как граф получится тогда не гамильтоновый.

Далее так как первая вершина уже является использованной, мы добавляем отмечаем ее как пройденную и добавляем ее в начало пути. Дальше двигаемся по графу если есть путь в какую-нибудь не посещенную вершину. Переходим в вершину 3 отмечаем ее посещенной и смотрим куда ведут ее ребра она связана с 2-ой , 4-ой и 5-ой вершинами. Переходим по первой не посещенной. Вторая вершинам связана с вершинами 1, 3, 4, 5 так как первую и третью вершины мы уже посетили, то их откидываем и переходим к 4-ой, и добавляем ее в массив посещенных вершин. Четвертая вершина связана со второй и пятой вершинами, так как вторая уже отмечена посещенной мы переходим на пятую и добавляем ее в массив посещенных вершин. Из пятой вершины есть путь к первой начальной вершине. Мы нашли гамильтонов путь для нашей ориентированной матрицы fmatrix. В итоге мы получили путь   
1-3-2-4-5-1.

Таким образом, можно сделать вывод что программа работает правильно

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска гамильтоновых циклов в орграфе и неориентипованном графе, 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 языка программирования C++.

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

Файл main.cpp

#include <iostream>

using namespace std;

void orient(int size);

void neo(int size);

void orient\_2(int size);

void neo\_2(int size);

int main()

{

setlocale(0, "");

int x, size;

cout << "Введите размер для графа:";

cin >> size;

cout << "Выберите тип графа:\n1-Ориентированный\n2-Неориентированный\n";

cin >> x;

switch (x)

{

case 1:

cout << "Выберите откуда взять граф:\n1-Из файла\n2-Создать случайный граф"<<endl;

cin >> x;

if (x == 1)

orient\_2(size);

else

orient(size);

break;

case 2:

cout << "Выберите откуда взять граф:\n1-Из файла\n2-Создать случайный граф" << endl;

cin >> x;

if (x == 1)

neo\_2(size);

else

neo(size);

break;

}

return 0;

}

Файл gam.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <fstream>

#include <string>

using namespace std;

int\* path;

void create(int size, int\*\* mat) {

int i, j, x;

srand(time(NULL));

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

mat[i] = (int\*)malloc(size \* sizeof(int));

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

mat[i][j] = rand() % 100;

}

}

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

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

if (mat[i][j] < 70) mat[i][j] = 1;

else mat[i][j] = 0;

}

}

}

void createneo(int size, int\*\* mat) {

int i, j, x=0, n=0;

srand(time(NULL));

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

mat[i] = (int\*)malloc(size \* sizeof(int));

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

mat[i][j] = rand()%100;

}

}

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

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

if (mat[i][j] < 70) mat[i][j] = 1;

else mat[i][j] = 0;

}

}

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

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

mat[i][j] = mat[j][i];

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

}

}

}

void printmatrix(int size, int\*\* mat) {

int i, j;

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

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

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

}

cout << endl;

}

}

void printmass(int size, int\* mas) {

int i;

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

cout << mas[i]+1 << " ";

cout << mas[0]+1;

cout << endl;

}

void visitor(int size, int\* mas) {

int i;

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

mas[i] = -1;

}

}

int gamilton(int size, int pos, int\*\* mat, int\* mas, int count) {

int i;

int success = 0;

for (i = 0; i < size && !success; i++)

{

if (mat[i][path[count - 1]] || mat[path[count - 1]][i])

{

if (count == size && i == pos)

{

success = 1;

continue;

}

else if (mas[i] == -1)

{

mas[i] = count;

path[count] = i;

success = gamilton(size, pos, mat, mas, count + 1);

if (!success) mas[i] = -1;

}

else continue;

}

}

return success;

}

int gamilton\_2(int size, int pos, int\*\* mat, int\* mas, int count) {

int i;

int success = 0;

for (i = 0; i < size && !success; i++)

{

if (mat[path[count - 1]][i])

{

if (count == size)

{

success = 1;

continue;

}

else if (mas[i] == -1)

{

mas[i] = count;

path[count] = i;

success = gamilton\_2(size, pos, mat, mas, count + 1);

if (!success) mas[i] = -1;

}

else continue;

}

}

return success;

}

void orient(int size) {

int pos, count = 1;

int\* vis;

int\*\* fmatrix;

fmatrix = (int\*\*)malloc(size \* sizeof(int));

path = (int\*)malloc(size \* sizeof(int));

vis = (int\*)malloc(size \* sizeof(int));

create(size, fmatrix);

printmatrix(size, fmatrix);

cout << "Гамильтонов цикл для каждой вершины" << endl;

for (pos = 0; pos < size; pos++)

{

visitor(size, vis);

path[0] = pos;

vis[pos] = pos;

if (gamilton\_2(size, pos, fmatrix, vis, count))

printmass(size, path);

else

cout << "Для точки: " << pos + 1 << " нет цикла" << endl;

}

}

void neo(int size) {

int pos, count = 1;

int\* vis;

int\*\* fmatrix;

fmatrix = (int\*\*)malloc(size \* sizeof(int));

path = (int\*)malloc(size \* sizeof(int));

vis = (int\*)malloc(size \* sizeof(int));

createneo(size, fmatrix);

printmatrix(size, fmatrix);

cout << "Гамильтонов цикл для каждой вершины" << endl;

for (pos = 0; pos < size; pos++)

{

visitor(size, vis);

path[0] = pos;

vis[pos] = pos;

if (gamilton(size, pos, fmatrix, vis, count))

printmass(size, path);

else

cout << "Для точки: " <<pos+1 <<" нет цикла" << endl;

}

}

void fromfile\_1(int size, int\*\* mat)

{

int i, j;

ifstream fin;

fin.open("ori.txt");

if (!fin.is\_open())

{

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

system("pause>>void");

}

else

{

string str;

int iin = 0;

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

{

mat[i] = (int\*)malloc(size \* sizeof(int));

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

{

fin >> str;

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

{

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

{

iin = stoi(str);

mat[i][j] = iin;

}

else

{

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

system("pause>>void");

}

}

}

}

fin.close();

}

}

void fromfile\_2(int size, int\*\* mat)

{

int i, j;

ifstream fin;

fin.open("nem.txt");

if (!fin.is\_open())

{

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

system("pause>>void");

}

else

{

string str;

int iin = 0;

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

{

mat[i] = (int\*)malloc(size \* sizeof(int));

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

{

fin >> str;

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

{

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

{

iin = stoi(str);

mat[i][j] = iin;

}

else

{

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

system("pause>>void");

}

}

}

}

fin.close();

}

}

void neo\_2(int size) {

int i, pos, count = 1;

int\* vis;

int\*\* fmatrix;

ofstream fout;

fout.open("output.txt");

fmatrix = (int\*\*)malloc(size \* sizeof(int));

path = (int\*)malloc(size \* sizeof(int));

vis = (int\*)malloc(size \* sizeof(int));

fromfile\_2(size, fmatrix);

printmatrix(size, fmatrix);

cout << "Гамильтонов цикл для каждой вершины" << endl;

for (pos = 0; pos < size; pos++)

{

visitor(size, vis);

path[0] = pos;

vis[pos] = pos;

if (gamilton(size, pos, fmatrix, vis, count))

{

printmass(size, path);

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

fout << path[i]+1;

}

fout << path[0]+1;

fout << endl;

}

else

cout << "Для точки: " << pos + 1 << " нет цикла" << endl;

}

fout.close();

}

void orient\_2(int size) {

int i, pos, count = 1;

int\* vis;

int\*\* fmatrix;

ofstream fout;

fout.open("output.txt");

fmatrix = (int\*\*)malloc(size \* sizeof(int));

path = (int\*)malloc(size \* sizeof(int));

vis = (int\*)malloc(size \* sizeof(int));

fromfile\_1(size, fmatrix);

printmatrix(size, fmatrix);

cout << "Гамильтонов цикл для каждой вершины" << endl;

for (pos = 0; pos < size; pos++)

{

visitor(size, vis);

path[0] = pos;

vis[pos] = pos;

if (gamilton\_2(size, pos, fmatrix, vis, count))

{

printmass(size, path);

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

fout << path[i] + 1;

}

fout << path[0]+1;

fout << endl;

}

else

cout << "Для точки: " << pos + 1 << " нет цикла" << endl;

}

fout.close();

}