

**Содержание**

[Реферат 3](#_Toc122385185)

[Введение 4](#_Toc122385186)

[1. Постановка задачи 5](#_Toc122385187)

[2. Теоретическая часть задания 6](#_Toc122385188)

[3. Описание алгоритма программы 8](#_Toc122385189)

[4. Описание программы 10](#_Toc122385190)

[5. Тестирование 13](#_Toc122385191)

[6. Ручной расчет задачи 18](#_Toc122385192)

[Заключение 19](#_Toc122385193)

[Список литературы 20](#_Toc122385194)

[Приложение А. Листинг программы 21](#_Toc122385195)

# Реферат

Отчет 27 страниц, 14 рисунков, 1 таблица.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, ГАМИЛЬТОНОВ ЦИКЛ

Цель курсовой работы – разработать программу, которая способна найти гамильтонов цикл для любого графа.

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

# Введение

Гамильтонов граф — это граф, содержащий гамильтонов цикл. При этом гамильтоновым циклом является такой цикл, 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, 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. Отсюда следует, что 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 курсовой работе реализуется алгоритм поиска Гамильтонова цикла.

# Постановка задачи

Требуется разработать программу, которая найдет 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. После чего выведет исходную матрицу и полученный результат на экран.

# Теоретическая часть задания

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

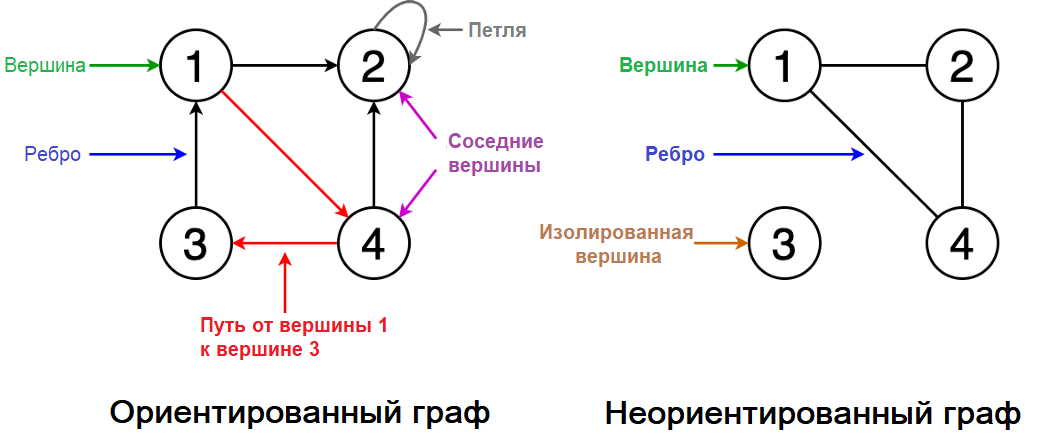


Рисунок 1 – Примеры графов

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 использовал 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.

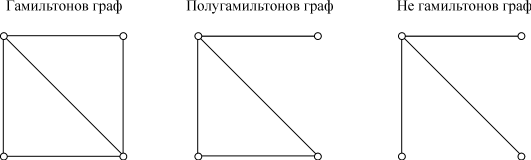


Рисунок 2 – Гамильтонов граф

# Описание алгоритма программы

Для реализации алгоритма понадобится два массива: doroga(int) – массив, в котором будет храниться гамильтонов путь графа, vis(int) – массив, в котором хранится информация о посещенных вершинах графа. И так имеется граф graph (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 versh и ей приписывается, что она посещена, вводится переменная success которая отвечает за успешность выполнения алгоритма. В doroga 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 doroga для каждой вершины. Если пока не дошли, программа будет проверять каждую вершину по массиву vis, проходя каждую вершину, он будет заносить номер в vis, а номер вершины в массив doroga. 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 doroga [i].

Ниже представлен псевдокод функции **hamilton**:

Функция hamilton:

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][ doroga [count - 1]] || mat[doroga [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 doroga номер вершины
   4. Вызываем функцию hamilton (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 получил популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Данная программа является многомодульной, поскольку состоит из нескольких функций:

**main** – главная функция программы,

**Create** – функция, в которой происходит создание матрицы для ориентированного графа;

**Createneo** – функция, в которой создается матрица для неориентированного графа;

**Visitor** – функция, для того, чтобы пометить посещенные вершины в графе;

**Hamilton** – данная функция отвечает за Гамильтонов цикл;

**Neorient, orient** – функции для работы с неориентированным и ориентированным графом.

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

int main()

{

setlocale(0, "");

int x, size;

cout << "Курсовую выполнил студент группы 21ВВ3 - Чабуев Ростислав\n";

cout << "Тема проекта: Реализация алгоритма нахождения Гамильтоновых циклов\n";

cout << "Введите размер для графа:";

cin >> size;

cout << "Выберите тип графа:\n1-Ориентированный граф\n2-Неориентированный граф\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)

neorient\_2(size);

else

neorient(size);

break;

}

return 0; }

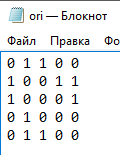


Рисунок 3 – Содержимое "ori.txt"

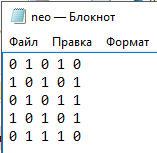


Рисунок 4 – Содержимое "neo.txt"

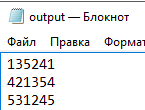


Рисунок 5 – Содержимое "output.txt" при ориент. графе взятого из файла

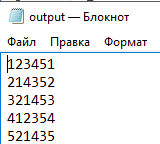


Рисунок 6 – Содержимое "output.txt" при неориент. графе взятого из файла

# 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 программы 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).

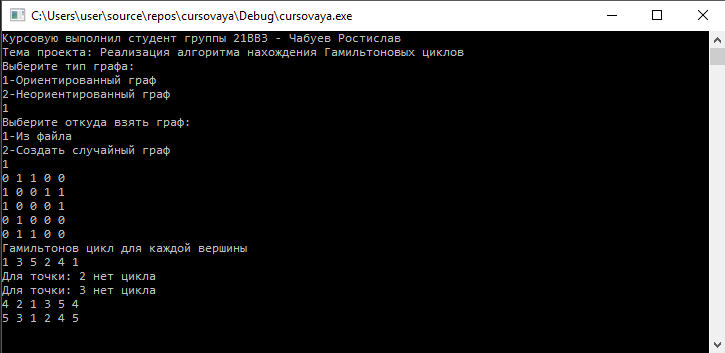


Рисунок 7 – Вывод ориентированного графа из файла

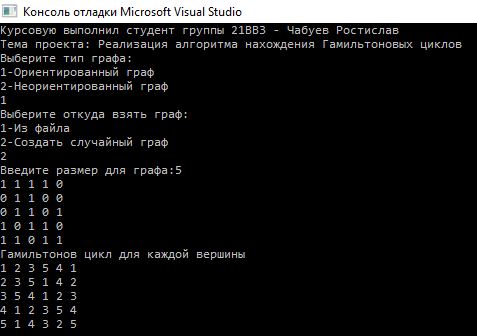


Рисунок 8 – Случайная генерация ориентированного графа размера 5

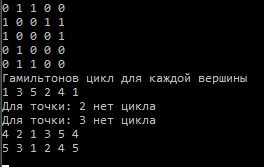


Рисунок 9 – Поиск Гамильтонова пути для ориент. графа из файла

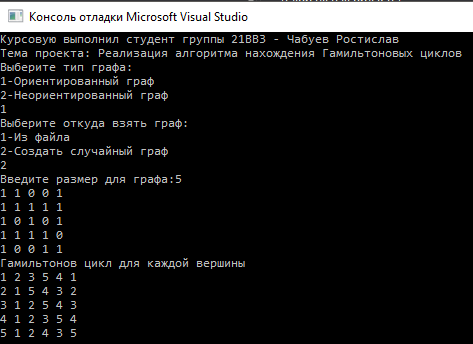


Рисунок 10 – Поиск Гамильтонова Пути для ориент. случайного графа

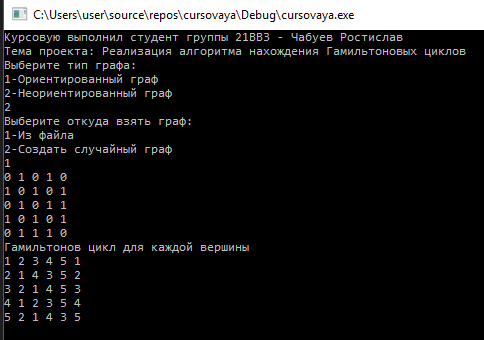


Рисунок 11 – Вывод неориентированного графа из файла

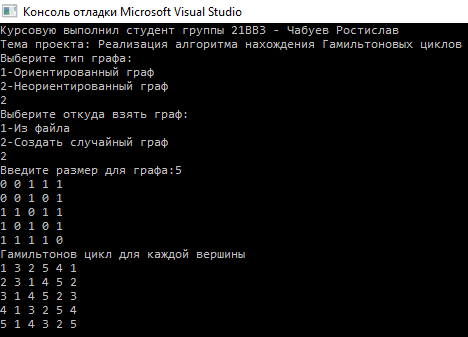


Рисунок 12 – Случайная генерация неориентированного графа размера 5

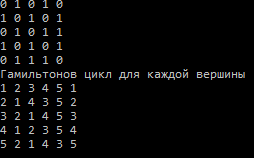


Рисунок 13 – Поиск Гамильтонова пути для неориент. графа из файла

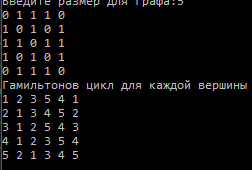


Рисунок 14 – Поиск Гамильтонова Пути для ориент. случайного графа

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы  (рис. 7 – 14) | Вывод запроса пользователя, ввести количество вершин графа | Верно |
| Выбор типа матрицы  (рис. 7 – 14) | Выбор типа матрицы ориентированная/  неориентированная | Верно |
| Выбор загрузки матрицы из файла  (рис. 7, 11) | Загрузка матрицы из заранее подготовленного файла «neo.txt» или «ori.txt» | Верно |
| Выбор генерации матрицы случайным образом  (рис. 8, 12) | Случайная генерация значений матрицы смежности | Верно |
| Нахождение и вывод гамильтоновых циклов для всех вершин  (рис. 7 – рис. 14) | Успешное выполнение программы и вывод всех циклов в консоль | Верно |

В результате тестирования программы было выявлено, что программа успешно проверяет данные на соответствие необходимых требований и выводит ту информацию, которую запросил пользователь.

# Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере неориентированного графа с 5-ю вершинами из файла (рис. 11).

Проходим все вершины с помощью обхода в глубину. После того как мы прошли все вершины, проверяем 5 вершину на смежность с начальной. Если условие выполняется, то выводим на экран Гамильтонов цикл, если нет, то мы помечаем последние вершины, как не посещённые и делаем это до тех пор, пока какая-то из вершин не окажется смежной с начальной.

Далее так как пятая вершина смежная с первой, то помечаем ее как посещенную. Дальше двигаемся по графу на 1 вершину, проверяем смежная ли она со 2 вершиной, если они смежные, то помечаем 2 вершину посещенной. Переходим в вершину 3 и проверяем смежность с 4 вершиной, так как они не смежные друг с другом, то отмечаем ее не посещенной и смотрим куда ведут ее ребра она связана со 2-ой и 5-ой вершинами. Проверяем смежность 3-ей вершины с 5-ой, а 5-ая смежные с 4-ой, то помечаем их как посещенные. Так как 4-ая вершина смежная с 1-ой, тогда выводим Гамильтонов путь на экран.

В итоге мы получили путь: 1-2-3-4-5-1.

Откуда следует, что результат, который получился при расчете вручную, сошелся с результатом, которая вывела программа, а это значит, что работы программы верна и уверенно можно доверять.

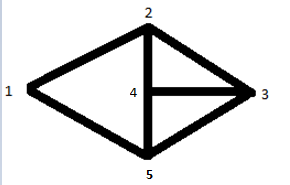


Рисунок 11 – Неориентированный граф из файла

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска гамильтоновых циклов в ориентированном и неориентированном графах, с помощью 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. Так как программа работает в консольном режиме и не позволяет улучшить интерфейс программы.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
2. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
3. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009г.
4. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.
5. Умножение матриц, Гамильтонов путь - https://habr.com
6. Hamiltonian Cycle - https://cppsecrets.com

# Приложение А. Листинг программы

Файл cursovaya.cpp

#include <iostream>

using namespace std;

void orient(int size1);

void neorient(int size1);

int orient\_2(int size1);

int neorient\_2(int size1);

int main()

{

setlocale(0, "");

int x, size1 = 0;

cout << "Курсовую выполнил студент группы 21ВВ3 - Чабуев Ростислав\n";

cout << "Тема проекта: Реализация алгоритма нахождения Гамильтоновых циклов\n";

cout << "Выберите тип графа:\n1-Ориентированный граф\n2-Неориентированный граф\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)

size1 = orient\_2(size1);

else

cout << "Введите размер для графа:";

cin >> size1;

orient(size1);

break;

case 2:

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

cin >> x;

if (x == 1)

size1 = neorient\_2(size1);

else

cout << "Введите размер для графа:";

cin >> size1;

neorient(size1);

break;

}

return 0;

}

Файл cursovaya.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <fstream>

#include <string>

using namespace std;

int\* doroga;

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 hamilton(int size, int versh, int\*\* mat, int\* mas, int count) {

int i;

int success = 0;

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

{

if (mat[i][doroga[count - 1]] || mat[doroga[count - 1]][i])//проверка на смежность вершин в графе

{

if (count == size && i == versh)//если все вершины просмотрены и вершина равна начальной

{

success = 1;

continue;

}

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

{

mas[i] = count;

doroga[count] = i;

success = hamilton(size, versh, mat, mas, count + 1);//рекурсия для просмотра других вершин

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

}

else continue;

}

}

return success;

}

int hamilton\_2(int size, int versh, int\*\* mat, int\* mas, int count) {

int i;

int success = 0;

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

{

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

{

if (count == size)

{

success = 1;

continue;

}

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

{

mas[i] = count;

doroga[count] = i;

success = hamilton\_2(size, versh, mat, mas, count + 1);

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

}

else continue;

}

}

return success;

}

void orient(int size) {

int versh, count = 1;

int\* vis;

int\*\* graph;

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

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

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

create(size, graph);

printmatrix(size, graph);

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

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

{

visitor(size, vis);

doroga[0] = versh;

vis[versh] = versh;

if (hamilton\_2(size, versh, graph, vis, count))

printmass(size, doroga);

else

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

}

}

void neorient(int size) {

int versh, count = 1;

int\* vis;

int\*\* graph;

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

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

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

createneo(size, graph);

printmatrix(size, graph);

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

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

{

visitor(size, vis);

doroga[0] = versh;

vis[versh] = versh;

if (hamilton(size, versh, graph, vis, count))

printmass(size, doroga);

else

cout << "Для точки: " << versh+ 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("neo.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();

}

}

int neorient\_2(int size) {

int i, versh, count = 1;

int\* vis;

int\*\* graph;

fstream fs;

fstream f2;

fs.open("neo.txt");

int a;

while (!fs.eof()) {

fs >> a;

size++;

}

fs.clear();

fs.seekg(0);

size = sqrt(size);

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

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

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

fromfile\_2(size, graph);

printmatrix(size, graph);

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

f2.open("output.txt");

f2.seekg(0);

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

{

visitor(size, vis);

doroga[0] = versh;

vis[versh] = versh;

if (hamilton\_2(size, versh, graph, vis, count))

{

printmass(size, doroga);

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

f2 << doroga[i] + 1;

}

f2 << doroga[0] + 1;

f2 << endl;

}

else

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

}

fs.close();

f2.close();

return(size);

}

int orient\_2(int size) {

int i, versh, count = 1;

int\* vis;

int\*\* graph;

fstream fs;

fstream f2;

fs.open("ori.txt");

int a;

while (!fs.eof()) {

fs >> a;

size++;

}

fs.clear();

fs.seekg(0);

size = sqrt (size);

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

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

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

fromfile\_1(size, graph);

printmatrix(size, graph);

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

f2.open("output.txt");

f2.seekg(0);

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

{

visitor(size, vis);

doroga[0] = versh;

vis[versh] = versh;

if (hamilton\_2(size, versh, graph, vis, count))

{

printmass(size, doroga);

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

f2 << doroga[i] + 1;

}

f2 << doroga[0] + 1;

f2 << endl;

}

else

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

}

fs.close();

f2.close();

return(size);

}