**Министерство науки и высшего образования Российской Федерации**

**Государственное образовательное учреждение**

**высшего профессионального образования**

**ГОУ ВПО “Пермский государственный научно-исследовательский университет”**

*Кафедра прикладной математики и информатики*

Отчёт

О выполнении

Индивидуального задания №1

**“ Нахождение эйлерова цикла/цепи в графе (алгоритм Флери) ”**

По дисциплине “Дискретная математика”

|  |
| --- |
| **Выполнила:**  Студентка 1-го курса механико-математического ф-та  Кравчук Юлия, группа ПМИ-5.  **Проверила:**  Доцент кафедры МОВС  канд. физ.-мат. наук,  Алябьева В.Г. |
|  |
|  |
|  |

Пермь 2022

**Оглавление**

1. [Постановка задачи 3](#_Toc98172314)
2. [Алгоритм решения поставленной задачи 3](#_Toc98172315)
3. [Результаты тестов 7](#_Toc98172316)
4. [Текст программы и комментарии к коду 9](#_Toc98172317)

Постановка задачи

**Задание:** Нахождение эйлерова цикла либо эйлеровой цепи в графе по алгоритму Флери.

**Входные данные:** В первой строке записано одно число n – количество вершин в графе. Далее располагается матрица смежности графа (n строк по n чисел в каждой).

**Выходные данные:** В первой строке – слово «Cycle» (если в графе есть эйлеров цикл), «Chain» (если нет цикла, но есть цепь) либо «No» (если нет ни цикла, ни цепи). Если в графе есть цикл или цепь, то во второй строке необходимо вывести найденный цикл или цепь соответственно (в виде последовательности номеров вершин через пробел).

**Язык программирования:** С++.

Алгоритм решения поставленной задачи

В своей программе я использовала функции для определения существования эйлерова цикла, эйлеровой цепи по матрице смежности и при успешной проверке использовала функцию для поиска эйлерова цикла либо эйлеровой цепи.

1. Функция **DegV** определяет степень вершины.

Данная функция получает данные матрицы смежности графа, количество его вершин и номер вершины, степень которой необходимо узнать.

Суммируя нули и единицы из строки матрицы смежности с номером равным номеру вершины, функция находит ее степень.

1. Функция **Connection** проверяет граф на связность.

Данная функция получает данные матрицы смежности, данные массива пройденных вершин, номер текущей вершины графа и количество всех вершин.

Если функция смогла попасть в вершину с текущим номером, то она заносит данные об этом в массив пройденных вершин, и данное действие функция выполняет до тех пор, пока не пройдет все возможные вершины по инцидентным им ребрам.

Граф будет связным, если все ячейки массива пройденных вершин будут содержать в себе true.

1. Функция **Most** проверяет, является ли ребро мостом.

Данная функция получает данные матрицы смежности, количество всех вершин графа и данные массива пройденных вершин.

Изначально все ячейки массива пройденных вершин содержат false. Далее применяется функция Connection, она заносит в массив новые данные о пройденных вершинах. Если в массиве остались ячейки со значением false, то ребро является мостом.

1. Функция **Proverka\_Cycle** проверяет необходимое и достаточное условие существования эйлерова цикла по матрице смежности.

Данная функция получает данные матрицы смежности графа и количество всех его вершин.

Необходимое и достаточное условие звучит следующим образом: для того, чтобы в связном графе существовал эйлеров цикл, необходимо и достаточно, чтобы в нем все вершины имели четную степень.

Таким образом, благодаря функции DegV подсчитываем степень каждой вершины графа. Если степень равна нечетному числу, то существование эйлерова цикла в подобном графе также невозможно. Если ни одна вершина графа не попала под данное условие, то в графе обязательно существует эйлеров цикл.

1. Функция **Proverka\_Chain** проверяет необходимое и достаточное условие существования эйлеровой цепи по матрице смежности.

Данная функция получает данные матрицы смежности графа и количество всех его вершин.

Необходимое и достаточное условие звучит следующим образом: для того, чтобы в связном графе существовала эйлерова цепь, необходимо и достаточно, чтобы в нем ровно две вершины имели нечетную степень.

Таким образом, благодаря функции DegV подсчитываем степень каждой вершины графа. Для вершин с нечетной степенью устанавливаем дополнительный счетчик для подсчета их количества. Если счетчик при завершении функции равен двум, то в данном графе обязательно существует эйлерова цепь.

1. Функция **Poisk\_Fleury** содержит в себе вызовы всех вышеупомянутых функций и необходимые действия для поиска эйлерова цикла либо эйлеровой.

Изначально проверяем, существует ли эйлеров цикл, если да, то начинать его построение можно с любой вершины, поэтому всегда будем начинать с первой (записываем ее в файл). Если нет, то проверяем существование эйлеровой цепи, если существует, то находим номера вершин с нечетной степенью, так как начинать и заканчивать построение цепи необходимо именно с их помощью. Начинать всегда будем с вершины с меньшим номером. Если в графе нет ни эйлерова цикла, ни эйлеровой цепи, то печатаем в файле “No” и выходим из функции.

Далее следует поиск ребер, проходя по которым, мы формируем эйлеров цикл либо эйлерову цепь в виде последовательности вершин.

Действуем, как и сказано в задании, по алгоритму Флери:

1. Из вершины, уже записанной в файл, идем по ребру, инцидентному ей, до следующей вершины. Переходим в следующую вершину. Отмечаем ребро как пройденное. Записываем в файл вершину, к которой пришли.
2. Если очередной вершине инцидентно только одно ребро, то переходим по нему к следующей вершине. Отмечаем ребро как пройденное. Записываем новую вершину в файл.
3. Если очередной вершине инцидентно несколько ребер, то среди них обязательно найдется ребро, не являющееся мостом в оставшемся графе. По этому ребру переходим в следующую вершину. Отмечаем ребро как пройденное. Записываем новую вершину в файл.

Алгоритм заканчивает работ через конечное число шагов, равное количеству ребер в исходном графе.

Реализация алгоритма в виде программы:

1. Если существует ребро и текущая вершина равна последней записанной вершине, то считаем ее степень при помощи функции DegV. Если степень равна единице, значит, вершине инцидентно всего одно ребро, поэтому идем по нему и записываем в файл вершину, к которой пришли.
2. Если степень больше нуля, то вершине инцидентно несколько ребер, из которых необходимо выбрать любое ребро, не являющееся мостом. Для этого удаляем ребро, которое нашли, проверяем при помощи функции Most связность графа с удаленным ребром. Если граф связный, значит, удаленное ребро точно не является мостом, поэтому переходим по этому ребру и записываем в файл вершину, к которой пришли.

Если граф оказался несвязным при удалении ребра, то возвращаем ребро обратно и снова проверяем граф на связность при помощи функции Most. Если граф и сейчас несвязный, значит, проблема несвязности не в данном ребре, а в том, что в графе есть уже пройденные вершины со степенью равной нулю, и из этого выходит, что ребро мостом не является, поэтому переходим по нему и записываем в файл вершину, к которой пришли.

1. Повторяем данный алгоритм до тех пор, пока не пройдем все ребра в исходном графе.
2. Основная часть программы **main** несет в себе последовательные шаги алгоритма всей программы.

План действий:

1. Устанавливаем переменную **t1** для подсчета времени в начале программы, чтобы в дальнейшем узнать, сколько секунд уходит на выполнение программы.
2. Открываем файл **f\_in** для чтения из него входных данных. В переменную **countV** заносим значение количества вершин графа. Далее создаем динамический двумерный массив **mas** и записываем в него данные матрицы смежности. Закрываем файл f\_in.
3. Создаем переменную **countE**, в которую запишем количество ребер графа, найдя его по формуле .
4. Так как существование эйлерова цикла и эйлеровой цепи возможно только в связном графе, то необходимо сделать проверку на связность графа. Для этого создаем дополнительный массив **visited** и заносим в него данные о том, что пока ни одна вершина не была пройдена. Вызываем функцию Connection. Далее проверяем измененные данные массива. Если осталась хоть одна не пройденная вершина, то ни цикла, ни цепи в данном графе точно быть не может. Открываем файл **f\_out** для записи. Печатаем в нем “No”. Если все вершины пройдены, то граф связный и тогда вызываем функцию Poisk\_Fleury.
5. После завершения выполнения работы последней функции закрываем файл f\_out. Удаляем массивы mas и visited для освобождения памяти. Устанавливаем переменную **t2** для подсчета времени в конце программы. Вычитаем из t2 t1. Преобразуем значение в десятичное число со значение в секундах и выводим полученное время на экран.

Результаты тестов

1. **Поиск циклов:**

|  |  |  |  |
| --- | --- | --- | --- |
| Входные данные | Выходные данные | Граф | Время выполнения программы (с) |
| 5  0 0 0 1 1  0 0 0 1 1  0 0 0 1 1  1 1 1 0 1  1 1 1 1 0 | Cycle  1 4 5 3 4 2 5 1 |  | 0.005 |
| 6  0 0 0 0 1 1  0 0 0 1 1 0  0 0 0 1 1 0  0 1 1 0 0 0  1 1 1 0 0 1  1 0 0 0 1 0 | Cycle  1 5 3 4 2 5 6 1 |  | 0.005 |
| 8  0 1 1 0 0 0 0 0  1 0 1 0 0 0 0 0  1 1 0 1 1 0 0 0  0 0 1 0 1 1 1 0  0 0 1 1 0 1 1 0  0 0 0 1 1 0 1 1  0 0 0 1 1 1 0 1  0 0 0 0 0 1 1 0 | Cycle  1 2 3 4 5 6 7 8 6 4 7 5 3 1 |  | 0.007 |
| 13  0 1 0 0 0 0 1 1 1 0 0 0 0  1 0 0 1 0 0 0 0 1 0 0 1 0  0 0 0 0 0 1 0 1 0 0 1 1 0  0 1 0 0 1 0 0 0 1 1 0 0 0  0 0 0 1 0 1 0 0 1 1 1 0 1  0 0 1 0 1 0 0 0 0 0 1 0 1  1 0 0 0 0 0 0 0 0 0 0 1 0  1 0 1 0 0 0 0 0 0 0 1 1 0  1 1 0 1 1 0 0 0 0 0 1 1 0  0 0 0 1 1 0 0 0 0 0 0 0 0  0 0 1 0 1 1 0 1 1 0 0 1 0  0 1 1 0 0 0 1 1 1 0 1 0 0  0 0 0 0 1 1 0 0 0 0 0 0 0 | Cycle  1 2 4 5 6 11 12 9 5 10 4 9 11 8 3 6 13 5 11 3 12 8 1 7 12 2 9 1 |  | 0.005 |

1. **Поиск цепей:**

|  |  |  |  |
| --- | --- | --- | --- |
| Входные данные | Выходные данные | Граф | Время выполнения программы (с) |
| 6  0 1 1 0 0 0  1 0 1 0 0 0  1 1 0 1 0 0  0 0 1 0 1 1  0 0 0 1 0 1  0 0 0 1 1 0 | Chain  3 1 2 3 4 5 6 4 |  | 0.005 |
| 8  0 1 1 1 0 1 0 0  1 0 0 1 0 0 0 0  1 0 0 1 1 1 0 0  1 1 1 0 1 1 1 0  0 0 1 1 0 1 0 0  1 0 1 1 1 0 0 1  0 0 0 1 0 0 0 1  0 0 0 0 0 1 1 0 | Chain  5 3 4 5 6 8 7 4 2 1 3 6 4 1 6 |  | 0.005 |
| 14  0 1 0 1 0 0 0 0 0 0 0 0 0 0  1 0 1 1 0 1 0 0 0 0 0 0 0 0  0 1 0 1 1 0 1 0 0 0 0 0 0 0  1 1 1 0 1 0 0 0 0 0 0 0 0 0  0 0 1 1 0 1 1 0 0 0 1 0 0 0  0 1 0 0 1 0 0 1 0 0 0 0 0 0  0 0 1 0 1 0 0 1 0 1 0 0 0 0  0 0 0 0 0 1 1 0 1 0 0 1 0 0  0 0 0 0 0 0 0 1 0 1 1 1 0 0  0 0 0 0 0 0 1 0 1 0 0 0 0 0  0 0 0 0 1 0 0 0 1 0 0 0 0 0  0 0 0 0 0 0 0 1 1 0 0 0 1 1  0 0 0 0 0 0 0 0 0 0 0 1 0 1  0 0 0 0 0 0 0 0 0 0 0 1 1 0 | Chain  5 3 4 5 7 8 9 10 7 3 2 1 4 2 6 8 12 13 14 12 9 11 5 6 |  | 0.002 |

1. **Несуществование ни циклов, ни цепей:**

|  |  |  |  |
| --- | --- | --- | --- |
| Входные данные | Выходные данные | Граф | Время выполнения программы (с) |
| 5  0 1 0 0 0  1 0 0 0 0  0 0 0 1 1  0 0 1 0 1  0 0 1 1 0 | No |  | 0.005 |
| 8  0 1 1 0 1 0 0 0  1 0 0 1 0 0 0 0  1 0 0 1 0 0 0 0  0 1 1 0 1 0 0 0  1 0 0 1 0 1 0 1  0 0 0 0 1 0 1 1  0 0 0 0 0 1 0 1  0 0 0 0 1 1 1 0 | No |  | 0.005 |
| 12  0 0 0 0 0 0 0 0 1 0 0 1  0 0 0 0 0 0 0 1 0 1 0 0  0 0 0 0 0 0 0 0 0 1 0 0  0 0 0 0 0 1 0 1 1 0 0 0  0 0 0 0 0 0 1 0 0 0 1 0  0 0 0 1 0 0 1 1 0 0 0 0  0 0 0 0 1 1 0 0 0 0 0 0  0 1 0 1 0 1 0 0 0 0 0 0  1 0 0 1 0 0 0 0 0 0 0 0  0 1 1 0 0 0 0 0 0 0 0 0  0 0 0 0 1 0 0 0 0 0 0 1  1 0 0 0 0 0 0 0 0 0 1 0 | No |  | 0.005 |

Текст программы и комментарии к коду

#include <iostream>

#include <fstream>

using namespace std;

int DegV(int\*\* mas, int countV, int i); //счетчик степени вершины i

void Connection(int\*\* mas, bool\*& visited, int i, int countV); //проверка на связность графа

bool Most(int\*\* mas, int countV, bool\*& visited); //проверка на ребро-мост

bool Proverka\_Cycle(int\*\* mas, int countV); //проверка на существование эйлерова цикла

bool Proverka\_Chain(int\*\* mas, int countV); //проверка на существование эйлеровой цепи

void Poisk\_Fleury(int\*\* mas, int countV, int countE, bool\*& visited, ofstream& f\_out); //построение цикла/цепи

int main() {

clock\_t t1 = clock(); //переменная для определения времени в начале программы

ifstream f\_in("input.txt"); //открытие файла для чтения

int countV; f\_in >> countV; //количество вершин в графе

int\*\* mas = new int\* [countV]; //создание массива для матрицы смежности

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

mas[i] = new int[countV];

for (int i = 0; i < countV; i++) //заполнение массива данными из матрицы смежности

for (int j = 0; j < countV; j++)

f\_in >> mas[i][j];

f\_in.close(); //закрытие файла для чтения

int countE = 0; //количество ребер в графе

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

for (int j = 0; j < countV; j++)

if (mas[i][j] == 1) countE++;

countE /= 2;

ofstream f\_out("output.txt"); //открытие файла для записи

bool\* visited = new bool[countV]; //массив пройденных вершин для проверки на связность графа

for (int m = 0; m < countV; m++) visited[m] = false; //значение false означает, что вершина не пройдена

Connection(mas, visited, 0, countV); //проходим все возможные вершины, начиная с первой

bool flag = true; //логическая переменная для определения связности/несвязности графа

for (int m = 0; m < countV; m++)

if (!visited[m]) { f\_out << "No"; flag = false; break; } //если встретилась непройденная вершина, значит, граф несвязный, печатаем в файле "No"

if (flag) Poisk\_Fleury(mas, countV, countE, visited, f\_out); //если граф связный, начинаем поиск эйлерова цикла либо эйлеровой цепи

f\_out.close(); //закрытие файла для записи

for (int i = 0; i < countV; i++) delete[]mas[i]; //освобождение памяти

delete[]mas; delete[]visited;

clock\_t t2 = clock(); //переменная для определения времени в конце программы

cout << (t2 - t1 + .0) / CLOCKS\_PER\_SEC << endl; //считаем время, затраченное на выполнение программы, и выводим его на экран

}

int DegV(int\*\* mas, int countV, int i) { //счетчик степени вершины i

int deg = 0;

for (int j = 0; j < countV; j++)

deg += mas[i][j]; //суммируем все единицы из строки i

return deg; //возвращаем степень вершины i

}

void Connection(int\*\* mas, bool\*& visited, int i, int countV) { //проверка на связность графа

if (visited[i]) return; //если вершина уже пройдена, выходим из функции

visited[i] = true; //иначе отмечаем вершину пройденной

for (int j = 0; j < countV; j++)

if (mas[i][j]) Connection(mas, visited, j, countV); //отмечаем пройденными все возможные вершины

}

bool Most(int\*\* mas, int countV, bool\*& visited) { //проверка на ребро-мост

for (int k = 0; k < countV; k++) visited[k] = false; //значение false означает, что вершина не пройдена

Connection(mas, visited, 0, countV); //проходим все возможные вершины, начиная с первой

for (int k = 0; k < countV; k++)

if (!visited[k]) return false; //если встретилась непройденная вершина, значит, мы попали на ребро-мост

return true;

}

bool Proverka\_Cycle(int\*\* mas, int countV) { //проверка на существование эйлерова цикла

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

int deg = DegV(mas, countV, i);

if (deg % 2 != 0) return false; //если у вершины нечетная степень, то эйлерова цикла быть не может

}

return true; //если все условия ложны, то возможно сущетсвование эйлерова цикла

}

bool Proverka\_Chain(int\*\* mas, int countV) { //проверка на существование эйлеровой цепи

int k = 0; //счетчик нечетных вершин

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

int deg = DegV(mas, countV, i);

if (deg % 2 != 0) k++; //если у вершины нечетная степень, то увеличиваем k на 1

if (k > 2) return false; //если вершин с нечетной степенью больше 2, то эйлеровой цепи быть не может

}

if (k == 2) return true; //если вершин с нечетной стпенью ровно 2, то возможно существование эйлеровой цепи

else return false; //если вершин с нечетной стпенью меньше 2, то эйлеровой цепи быть не может

}

void Poisk\_Fleury(int\*\* mas, int countV, int countE, bool\*& visited, ofstream& f\_out) { //построение цикла/цепи

int k = 0, //счетчик пройденных ребер в графе

i = 0, j = 0, //координаты ячейки матрицы смежности

step = 0, //переменная для перемещения сверху вниз/снизу вверх по матрице смежности

lastv = 0; //переменная для запоминания последней пройденной вершины

if (Proverka\_Cycle(mas, countV) == true) //если возможно наличие эйлерова цикла,

f\_out << "Cycle" << endl << i + 1; //записываем в файл "Cycle", и т.к. можно начать с любой вершины, начнем с первой

else { //если эйлерова цикла точно быть не может

if (Proverka\_Chain(mas, countV) == false) { //если и эйлеровой цепи точно быть не может,

f\_out << "No"; return; //то записываем в файл "No", выходим из функции

}

//если возможно наличие эйлеровой цепи:

int V1 = -1, V2 = -1; //номера вершин с нечетной степенью

for (int m = 0; m < countV; m++)

if (DegV(mas, countV, m) % 2 != 0) { //находим номера вершин с нечетной степенью

V1 < 0 ? V1 = m : V2 = m;

if (V1 > 0 && V2 > 0) break;

}

i = V1; f\_out << "Chain" << endl << i + 1; //записываем в файл "Chain", и т.к. нужно начинать с вершины с нечетной стпенью, начнем с первой из них

lastv = i; //запоминаем записанную вершину

}

while (k < countE) { //пока есть непройденные ребра

if (mas[i][j] && lastv == i) //если сущетсвует ребро между i и j и последняя записанная вершина равна i

if (DegV(mas, countV, i) == 1) { //если текущей вершине инцидентно всего одно ребро, то переходим по нему

f\_out << " " << j + 1; //записываем вершину, к которой пришли, в файл

mas[i][j] = mas[j][i] = 0; //отмечаем ребро пройденным

k++; i = lastv = j; //увеличиваем k на 1, запоминаем записанную вершину

}

else { //если текущей вершине инцидентно несколько ребер

mas[i][j] = mas[j][i] = 0; //удаляем ребро для проверки, не является ли оно мостом

if (Most(mas, countV, visited)) { //если ребро мостом не является

f\_out << " " << j + 1; //записываем вершину, к которой пришли, в файл

k++; i = lastv = j; //увеличиваем k на 1, запоминаем записанную вершину

}

else { //если ребро может являться мостом

mas[i][j] = mas[j][i] = 1; //возвращаем удаленное ребро

if (!Most(mas, countV, visited)) { //граф несвязный, но не из-за возвращенного ребра

f\_out << " " << j + 1; //записываем вершину, к которой пришли, в файл

mas[i][j] = mas[j][i] = 0; //отмечаем ребро пройденным

k++; i = lastv = j; //увеличиваем k на 1, запоминаем записанную вершину

}

}

}

if (step == 0) //идем сверху вниз по матрице смежности

if (j + 1 >= countV) //если мы находимся на последнем столбце

if (i + 1 < countV) { i++; j = 0; } //если мы не находимся на последней строке, то переходим на следующую строку, на первый столбец

else step = 1; //если мы дошли до конца матрицы, то начинаем проход по матрице снизу вверх

else j++; //если мы не на последнем столбце, то переходим на следующий столбец

else //идем снизу вверх по матрице смежности

if (j - 1 < 0) //если мы находимся на первом столбце

if (i - 1 >= 0) { i--; j = countV - 1; } //если мы не находимся на первой строке, то переходим на предыдущую строку, на последний столбец

else step = 0; //если мы дошли до начала матрицы, то начинаем проход по матрице сверху вниз

else j--; //если мы не на первом столбце, то переходим на предыдущий столбец

}

}