Министерство науки и высшего образования РФ ФГБОУ ВПО

«Пермский государственный национальный исследовательский университет»

Кафедра прикладной математики и информатики

Отчет о выполнении индивидуального задания по теме

**«Нахождение эйлерова цикла/цепи в графе (алгоритм Флери)»**

***по дисциплине «Дискретная математика»***

|  |
| --- |
| **Выполнила:**  Студентка 1-го курса механико-математического факультета  Шмырина Владислава Олеговна  группа ПМИ-1,2-2021 |

Пермь 2022

**Постановка задачи**

задание: Нахождение эйлерова цикла/цепи в графе (алгоритм 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: cycle/chain - в зависимости от того, что было найдено цикл или путь.

и сам цикл/путь соответственно

**Основная функция + описание алгоритма**

1. Создаем двумерный список вершин, матрица смежности
2. Считываем входные данные из файла в матрицу
3. Происходит подсчет степеней вершин и поиск начальной вершины
4. Ищем ненулевую степень у вершин с которой мы и будем начинать ход
5. определение эйлеровости графа и поиск нечетных вершин

Переменные

(

v1 - это 1 переменная для цепи, v2 - 2 переменная для цепи (вход и выход для цепи)

Bad - переменная, которая возвращает 0 если граф не эйлеровый

)

6. если есть нечетные вершины в графе, строим дополнительное ребро для построения цикла

7. поиск цикла в графе

8. проверяем граф на связность

9. вывод и определение типа графа

**Код программы с описанием данных**

#include "iostream" //манипулятор основных процессов

#include "iomanip" //манипулятор вывода

#include "vector" //динамич массивы-контейнеры

#include "list" //двусвязные списки

#include "fstream" //файлы

#include "stack" //список-стек

#include <ctime>

using namespace std;

int main()

{

// ввод графа

setlocale(LC\_ALL, "Russian");

ifstream f1("input.txt", ios::in);

ofstream f2("output.txt", ios::out);

int n; //число строк/столбцов/вершин

unsigned int start\_time = clock(); //начало выполнения программы(отсюда происходит отсчет времени)

f1 >> n;

vector < vector<int> > g(n, vector<int>(n)); //двумерный список вершин, матрица смежности

for (int i = 0; i < n; i++) //чтение графа в матрицу смежности

{

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

{

f1 >> g[i][j];

}

}

//подсчет степеней вершин и поиск начальной вершины

vector<int> deg(n); //подсчет степеней вершин каждой вершины

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

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

deg[i] += g[i][j];

int first = 0;

while (!deg[first]) //ищем ненулевую степень у вершин с которой мы и будем начинать ход

{

++first;

}

//определение эйлеровости графа и поиск нечетных вершин

int v1 = -1, v2 = -1; //v1 - это 1 переменная для цепи, v2 - 2 переменная для цепи (вход и выход для цепи)

bool bad = false; //переменная, кот-ая возвращает 0 если граф не эйлеровый

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

if (deg[i] & 1) //если вершина не четна

if (v1 == -1)

v1 = i;

else

if (v2 == -1)

v2 = i;

else

bad = true; //если нечетных вершин больше двух - ни цепь, ни цикл не построить

//если есть нечетные вершины в графе, строим дополнительное ребро для построения цикла

if (v1 != -1) //если в графе цепь

{

++g[v1][v2], ++g[v2][v1]; //строим дополнительное ребро для построения и в итоге будут все четные вершины => эйлеров цикл будет сущ-ть

}

//поиск цикла в графе

stack <int> st; //создаем стек

st.push(first); //добавим первую вершину в стек

vector <int> res;

if (!bad) //если граф эйлеровый

{

while (!st.empty()) //пока стек не пустой, ищем цикл в графе

{ //при цепи: при ребре-мосте (при нечетн вершинах, цепях соотв-но), зацикливается цикл при не пройденном до конца графе, путей из вершины тогда не находится, i = n, поэтому стек начинает перезаписывать до своего обнуления в массив вывода, но наоборот(учитывается пройденное-ребро мост по алгоритму флери, поэтому цикл переворачивается, а соединенные ребром нечетные вершины (с цифрой 2, если ребро между ними уже есть, оно учитывается и как бы уже есть в цикле, поэтому ошибки не происходит, два ребра разбиваются, один в стек, другой все еще в графе), поэтому цикл не зацикливается дальше и идет правильно, все ребра проходятся, цепь найдена, после прохода всего графа после этой рокировки все нормально вставляется в выходной массив

//при цикле - программа спокойно находит любой цикл

int v = st.top(); //вызываем верхний элемент стека, это номер вершины и с помощью нее мы выбираем куда идти по строке где есть 1

int i;

for (i = 0; i < n; ++i) //если цикл доходит до конца то из-за ++i i = n

{

if (g[v][i]) //если ребро есть - завершение

{

break;

}

}

if (i == n) //если цикл доходит до конца(строка пуста), вершина записывается в цепочку-массив ребер для вывода

{

res.push\_back(v); //вставляем новый элемент в конец

st.pop(); //удаляем верхний элемент

}

else //при эйлеровых цепях между нечетными вершинами достраивается ребро тогда получается цепь а в последствии мы просто убираем это ребро и получается цепь

{

--g[v][i];

--g[i][v]; //удаляем ребро чтобы не зациклился ход по матрице и во избежание повторок

st.push(i); //добавляем элемент в верх стека и удаляем ребро в графе

}

}

// если в графе цепь, удаляем лишнее ребро в результирующей цепи вершин

if (v1 != -1)

for (size\_t i = 0; i + 1 < res.size(); ++i) //проискиваем в цикле по выводному массиву где у нас одна вершина будет равна нечетной первой а следующая за ней 2 нечетн или наоборот

if (res[i] == v1 && res[i + 1] == v2 || res[i] == v2 && res[i + 1] == v1)

{ //size\_t - беззнаковый целый тип данных

vector<int> res2;

for (size\_t j = i + 1; j < res.size(); ++j) //переступаем через "лишнее" вставленное для нечетных вершин ребро в цепи и записываем весь путь после этого ребра в новый массив

res2.push\_back(res[j]);

for (size\_t j = 1; j <= i; ++j) //а теперь берем все вершины перед лишним ребром, но его опять же не берем

res2.push\_back(res[j]);

res = res2; //изменяем результирующий массив буквально на одно значение, окончательно убираем лишнее ребро

break;

}

//доп проверка на связность

for (int i = 0; i < n; ++i) //дополнительная проверка на то, связный ли граф (при оставшихся ребрах - нет)

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

{

if (g[i][j]) //учитываем случай если вдруг цикл поиска ошибся и какое-то ребро осталось, значит граф не эйлеровый

{

bad = true;

}

}

if (bad)

f2 << "No" << endl;

}

// вывод и определение типа графа

else //Если граф не эйлеровый, то выводим как есть, не тратим время зря при неэйлеровом графе

f2 << "No" << endl;

if (!bad) //если переменная определяющая, что граф эйлеровый, равна !1, то вывод соответв-ий

{

if (v1 != -1) //если нечетные вершины есть

{

f2 << "Chain" << endl;

for (size\_t i = 0; i < res.size(); ++i) //вывод прохода по цепи, если она есть

f2 << res[i] + 1 << " ";

}

else

{

f2 << "Cycle" << endl;

for (size\_t i = 0; i < res.size() - 1; ++i) //вывод прохода по циклу, если он есть, без печати возвращения в начало

f2 << res[i] + 1 << " ";

}

}

unsigned int end\_time = clock(); // конец выполнения программы

unsigned int time = end\_time - start\_time; // итоговое время выполнения программы

cout << "\nВремя выполнения программы: " << time << " мс.";

}

**Результаты тестирования**

Результаты данных тестов были проверены с помощью online-сервиса https://programforyou.ru/graph-redactor . Изображения графов взяты с этого же источника. Как уже говорилось в алгоритме, мною использовалась функция clock() для определения длительности выполнения программы. Сравнивая скорость выполнения программы на графах различной длины, можно заметить, что она работает достаточно быстро. Время выполнения не сильно заметно (по сравнению с другими тестами) увеличивается при проверке на больших графах (тест № 5). Стоит заметить, что при разных запусках одного и того же теста время выполнения программы менялось, в таблице представлены данные, приближенные к максимальным из проведенных. Для графов с маленьким количеством вершин время выполнения программы варьировалось в основном от 1 до 5 мс. Ознакомиться с проведенными экспериментами можно в ниже представленной таблице.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № теста | Входные данные | Изображение графа | Входные данные, результат | | Вывод о корректности(правильности) ответа | Информация о времени выполнения (мс) |
|  |  |  | программа | Online-сервис |  |  |
|  | | | | | | |
| 1 | 3  0 1 1  1 0 1  1 1 0 |  | Cycle  1 3 2 | Cycle  1 3 2 | Ответ верный | 2мс |
| 2 | 4  0 1 1 1  1 0 0 0  1 0 0 0  1 0 0 0 |  | No | No | Ответ верный | 1 мс |
| 3 | 8  0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0 0 0 |  | Chain  7 4 8 3 7 6 4 3 6 2 1 5 2 | Chain  7 4 8 3 7 6 4 3 6 2 1 5 2 | Ответ верный | 2 мс |
| 4 | 12  0 0 0 0 0 0 0 0 0 1 0 0  0 0 0 0 0 0 1 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 1 0 0 0 0 0 0  0 0 0 0 1 0 0 0 0 0 0 0  0 1 0 0 0 0 0 1 0 0 0 0  0 0 0 0 0 0 1 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1 0 0  1 0 0 0 0 0 0 0 1 0 0 0  0 0 0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0 1 0 |  | No | No | Ответ верный | 1 мс |
| 5 | 31  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0  0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0  0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 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  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0  0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0  0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |  | No | No | Ответ верный | 5 мс |