

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
по дисциплине «Логика и основы алгоритмизации в инженерных задачах»
на тему «Разработка алгоритма и программного обеспечения для построения
эйлерова цикла в графе»

Удостоверено
27.11.20


Выполнил:

ст. группы 18ВВЗ
Дыкин И. С.

Принял:

д. т. н., профессор
Митрохин М.А.

Пенза 2020

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет вычислительной техники

Кафедра

Вычислительная техника

«УТВЕРЖДАЮ»

Зав. кафедрой

« 08 » сентября 2019 г.

ЗАДАНИЕ

на курсовое проектирование по курсу

Логика и основы алгоритмизации в инженерных задачах

Студенту Дыкину Ивану Сергеевичу Группы 18ВВ3

Тема проекта Построение эйлеровой цепи в графе.

Исходные данные (технические требования) на проектирование

Задание (вариант 5) Построить эйлерову цепь в графе. Изменить алгоритм для построения построения эйлеровой цепи для построения эйлерова цикла.

Объем работы по курсу

1. Расчетная часть

Ручной расчёт построения цепи.

2. Графическая часть

Графическое представление результата работы программы

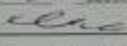
3. Экспериментальная часть
Разработка и тестирование программы

Срок выполнения проекта по разделам

1	Расчётная часть	к 15.11.2019
2	Графическая часть	к 22.11.2019
3	Экспериментальная часть	к 29.11.2019
4		к
5		к

Дата выдачи задания « 08 » сентября 2019 г.

Дата защиты проекта « 3 » декабря 2019 г.

Руководитель 

Задание получил « 08 » сентября 2019 г.

Студент Дьякин Иван Сергеевич 

Содержание	
Введение	3
1 Постановка задачи	4
2 Теоретическая часть	5
3 Описание программы	6
4 Ручной расчет	8
Заключение	13
Список используемой литературы	14
Приложение А	15

Введение

Теория графов - один из обширнейших разделов дискретной математики, широко применяется в решении экономических и управленческих задач, в программировании, химии, конструировании и изучении электрических цепей, коммуникации, психологии, социологии, лингвистике, других областях знаний. Теория графов систематически и последовательно изучает свойства графов, о которых можно сказать, что они состоят из множества вершин и множества ребер. Основателем теории графов считается Леонард Эйлер (1707-1882), решивший в 1736 году известную в то время задачу о кёнигсбергских мостах.

Эйлеров граф — граф, содержащий эйлеров цикл.

Эйлеров путь (эйлерова цепь) в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.

Эйлеров цикл — эйлеров путь, являющийся циклом, то есть замкнутый путь, проходящий через каждое ребро графа ровно по одному разу.

1 Постановка задачи

Необходимо разработать программу построения эйлерова цикла в графе на языке C/C++.

У пользователя должен быть выбор построить эйлеров цикл или цепь. В цикле пользователь должен иметь возможность выбрать начало цикла.

Управление в программе должно быть реализовано посредством клавиатуры. Программа должна обеспечивать контроль корректности ввода данных пользователем

Реализация приложения осуществляется посредством консольного приложения с использованием программного обеспечения Microsoft Visual Studio 2017.

2 Теоретическая часть

Эйлеров путь (эйлерова цепь) в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.

Эйлеров цикл — эйлеров путь, являющийся циклом, то есть замкнутый путь, проходящий через каждое ребро графа ровно по одному разу.

Эйлеров граф — граф, содержащий эйлеров цикл.

Полуэйлеров граф — граф, содержащий эйлеров путь.

В неориентированном графе Эйлеров цикл существует тогда и только тогда, когда граф связный или будет являться связным, если удалить из него все изолированные вершины, и в нём отсутствуют вершины нечётной степени.

Эйлеров путь в графе существует тогда и только тогда, когда граф связный и содержит не более двух вершин нечётной степени. Число вершин с нечётной степенью должно быть четным, значит эйлеров путь существует только тогда, когда это число равно нулю или двум. Причём когда оно равно нулю, эйлеров путь вырождается в эйлеров цикл.

Ориентированный граф содержит эйлеров цикл тогда и только тогда, когда он сильно связан или среди его компонент сильной связности только одна содержит ребра (а все остальные являются изолированными вершинами) и для каждой вершины графа её входящая степень равна её исходящей степени, то есть в вершину входит столько же ребер, сколько из неё и выходит.

Ориентированный граф содержит эйлеров путь, не являющийся циклом, тогда и только тогда, когда существуют две вершины такие что у одной вершины исходящая степень на единицу больше входящей степени, а у второй — исходящая степень на единицу меньше входящей степени. В таком случае эйлеров путь возможно построить из первой вершины.

3 Описание работы программы

После запуска программы пользователю даётся выбор вывод эйлерового цикла(1) или цепи(2).

Если пользователь выбирает вывод цепи, то на экран выводится матрица с заранее определённым размером и вершинами.

Если пользователь выберет вывод цикла то на экран выводится матрица с заранее определённым размером и вершинами, после этого у пользователя будет выбор из какой вершины строить цикл.

Представление алгоритма voidCycle() проверки на наличие цепи в формате псевдокода:

ЦИКЛ i от 0 до размера массива-1

Сложить все ячейки строки

ЕСЛИ сумма не равна нулю ТО флаг isCycle = false

Занести значение суммы ячеек в вспомогательный массив F

Представление алгоритма voidChain() проверки на наличие цикла в формате псевдокода:

ЦИКЛ i от 0 до размера массива-1

ЦИКЛ j от 0 до размера массива-1

ЕСЛИ в массиве F найдены 1 И -1 ИЛИ -1 И 1 ТО

Индекс единицы запомнить

Найденные ячейки F обнулить

Флаг f = true

Выход из цикла

ЕСЛИ f истинно ТО выход из цикла

ЦИКЛ i от 0 до размера массива-1

ЕСЛИ ячейка F равна 0 ТО isChain = false

Представление алгоритма voidChain() поиска вершин в формате псевдокода:

ЦИКЛ i от 0 до размера массива-1

ЕСЛИ существует ребро с индексами v и i ТО

оно удаляется

Search(i)

В массив Stack заносится v

4 Ручной расчет

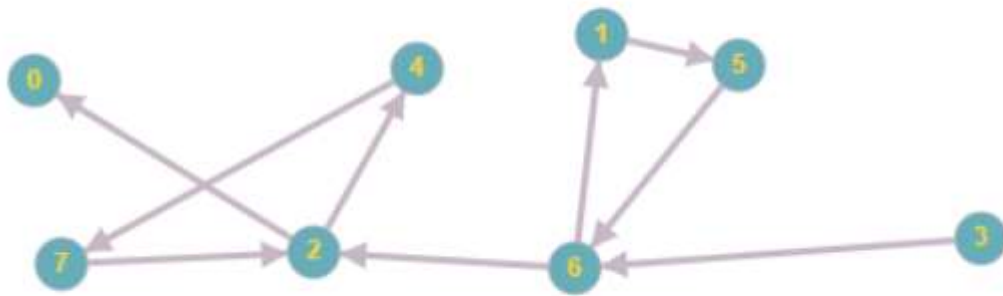


Рисунок 1- исходное состояние графа

Вызов Scan(3); Удаление ребра 3-6

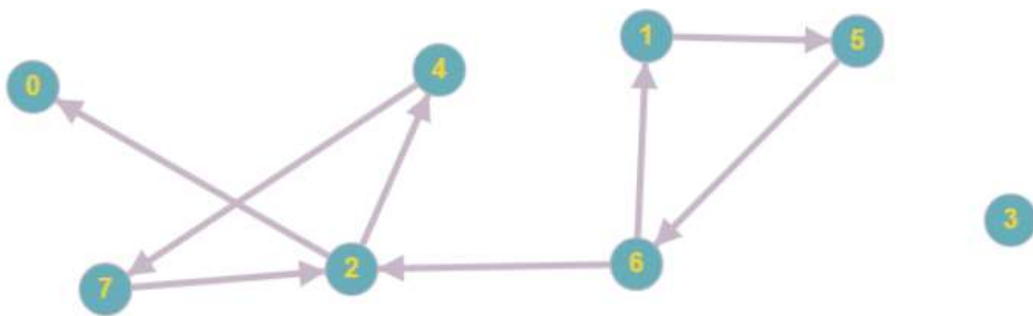


Рисунок 3- Граф после удаления ребра 3-6

Вызов Scan(6); Удаление ребра 6-1

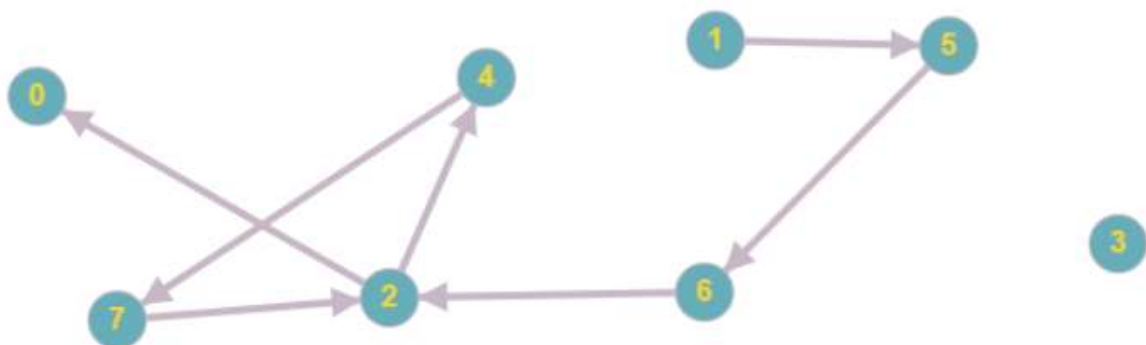


Рисунок 4- Граф после удаления ребра 6-1

Вызов `Scan(1)`; Удаление ребра 1-5

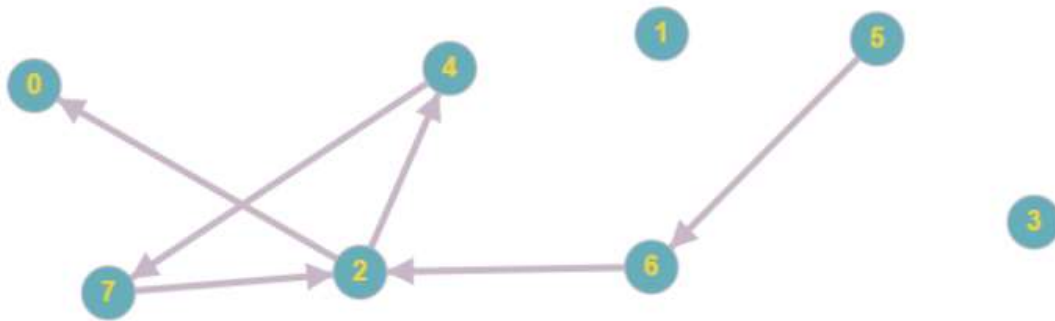


Рисунок 5- Граф после удаления ребра 1-5

Вызов `Scan(5)`; Удаление ребра 5-6

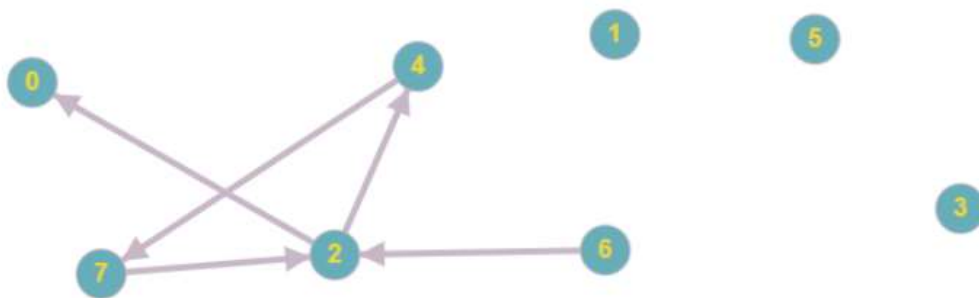


Рисунок 6- Граф после удаления ребра 5-6

Вызов `Scan(6)`; Удаление ребра 6-2

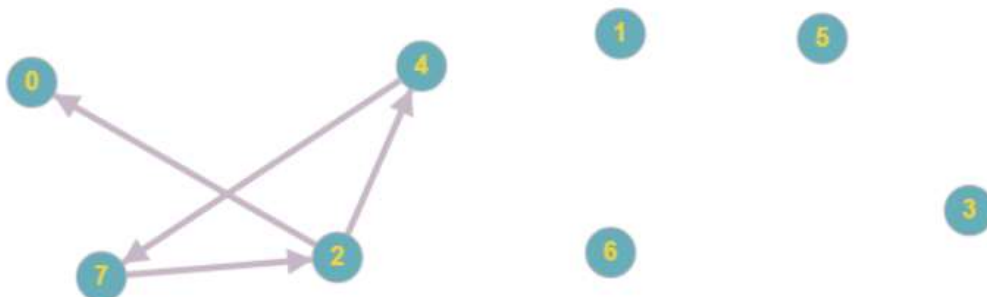


Рисунок 7- Граф после удаления ребра 6-2

Вызов `Scap(2)`; Удаление ребра 2-4

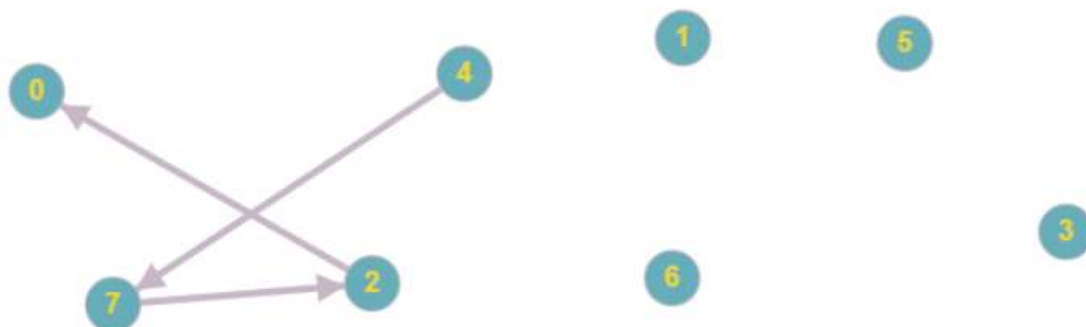


Рисунок 8- Граф после удаления ребра 2-4

Вызов `Scap(4)`; Удаление ребра 4-7

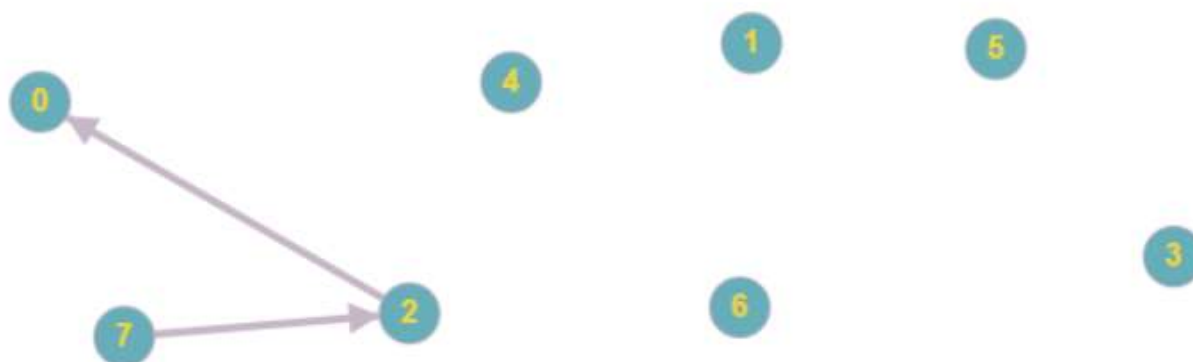


Рисунок 9- Граф после удаления ребра 4-7

Вызов `Scap(7)`; Удаление ребра 7-2

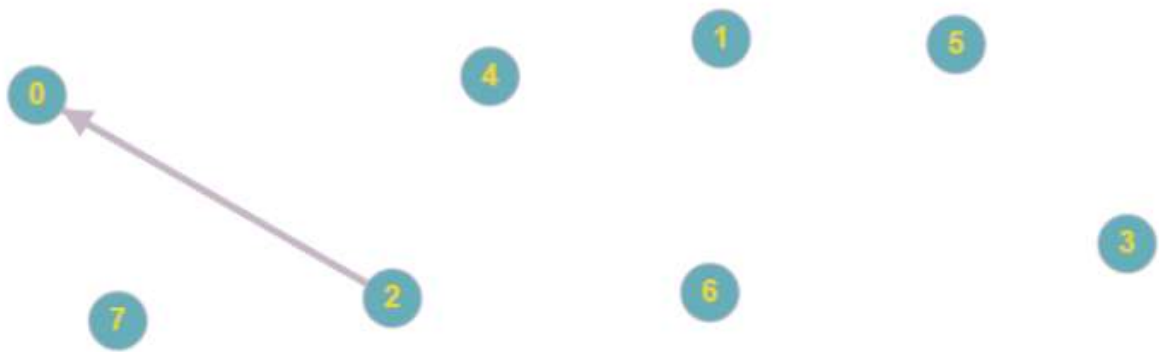


Рисунок 10- Граф после удаления ребра 7-2
Вызов Scan(2); Удаление ребра 2-0

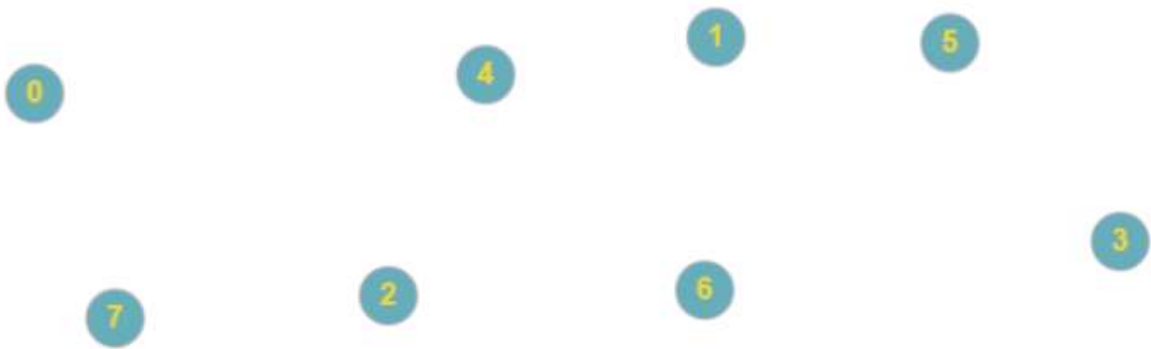


Рисунок 11- Граф после удаления ребра 2-0
Вызов Search(0); Матрица смежности обнулилась, Search(0) полностью выполнена; 2 заносится в массив;
Scan(2) полностью выполнена; 7 заносится в массив;
Scan(7) полностью выполнена; 4 заносится в массив;
Scan(4) полностью выполнена; 2 заносится в массив;
Scan(2) полностью выполнена; 6 заносится в массив;
Scan(6) полностью выполнена; 5 заносится в массив;
Scan(5) полностью выполнена; 1 заносится в массив;
Scan(1) полностью выполнена; 6 заносится в массив;

Scan(6)полностью выполнена;3 заносится в массив;

Scan(3)полностью выполнена; массив пути выглядит так: 0 2 7 4 2 6 5 1 6 3

Результат выглядит так: 3 6 1 5 6 2 4 7 2 0

```
Цикл(1) или Цепь(2):  
2  
0 0 -1 0 0 0 0 0  
0 0 0 0 0 1 -1 0  
1 0 0 0 1 0 -1 -1  
0 0 0 0 0 0 1 0  
0 0 -1 0 0 0 0 1  
0 -1 0 0 0 0 1 0  
0 1 1 -1 0 -1 0 0  
0 0 1 0 -1 0 0 0  
Эйлерова цепь возможна из вершины 3:  
3 6 1 5 6 2 4 7 2 0
```

Рисунок 12- Результат работы программы

Заключение

В ходе выполнения данной курсовой работы были получены навыки обработки графов на ПК, расчета основных параметров графов. Разработаны алгоритмы построения и обработки графов. Был изучен и реализован алгоритм построения эйлерового цикла.

Список использованных источников

1. Алгоритм Флери построения эйлерова цикла <https://natalibrilenova.ru/blog/> свободный.
2. Кристофидес Н. Теория графов. Алгоритмический подход (глава 9.5) [Электронный ресурс] — М.: Мир, 1978.
3. Библиотека MSDN [Электронный ресурс]. – Режим доступа: <http://msdn.microsoft.com/> свободный.
4. Википедия свободная энциклопедия. Эйлеров цикл [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/> свободный.
5. Зыков А.А. Основы теории графов – Москва, Наука, 1987г. - 381 с.
6. Керниган Б.В., Ричи Д.М. Язык Си – Санкт-Петербург, Невский диалект, 2003г. – 355 с.

ПРИЛОЖЕНИЕ А

Листинг программы

```
#include "pch.h"
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <cstdlib>
#include <Windows.h>
#include <time.h>
#define STACK_SIZE 100

int k = -1;
int Stack[STACK_SIZE];
int *G;
int *F;
int N;
int f1=-1000;
bool isCycle = true, isChain = true;

void Scan(int v)
{
    int i;
    for (i = 0; i < N; i++)
        if (*(G + N * v + i))
        {
            *(G + N * v + i) = *(G + N * i + v) = 0;
            Scan(i);
        }
    Stack[++k] = v;
}

void Cycle() {
    int s;
    for (int i = 0; i < N; i++)
    {
        s = 0;
        for (int j = 0; j < N; j++)
        {
            s += *(G + N * i + j);
        }
        if (s != 0)
            isCycle = false;
        *(F + i) = s;
    }
}

void Chain()
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (((*(F + i) == 1 && *(F + j) == -1) || (*(F + i) == -1 && *(F + j)
== 1)))
            {
                if (*(F + i) > 0)
                    f1 = i;
                else
                    f1 = j;
            }
        }
    }
}
```

```

        *(F + i) = 0;
        *(F + j) = 0;
    }
}

}

int main()
{
    srand(time(NULL));
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    int size = sizeof(int);
    int startV;
    int vybor;
    printf("Цикл(1) или Цепь(2):\n");
    scanf_s("%d", &vybor);
    N = 8;
    F = (int*)malloc(N*size);
    G = (int*)malloc(N*N*size);

    if (vybor == 1) {
        *(G + N * 0 + 0) = 0; // цикл
        *(G + N * 0 + 1) = -1;
        *(G + N * 0 + 2) = 0;
        *(G + N * 0 + 3) = 0;
        *(G + N * 0 + 4) = 0;
        *(G + N * 0 + 5) = 1;
        *(G + N * 0 + 6) = 0;
        *(G + N * 0 + 7) = 0;

        *(G + N * 1 + 0) = 1;
        *(G + N * 1 + 1) = 0;
        *(G + N * 1 + 2) = -1;
        *(G + N * 1 + 3) = 0;
        *(G + N * 1 + 4) = 1;
        *(G + N * 1 + 5) = -1;
        *(G + N * 1 + 6) = 0;
        *(G + N * 1 + 7) = 0;

        *(G + N * 2 + 0) = 0;
        *(G + N * 2 + 1) = 1;
        *(G + N * 2 + 2) = 0;
        *(G + N * 2 + 3) = -1;
        *(G + N * 2 + 4) = 0;
        *(G + N * 2 + 5) = 0;
        *(G + N * 2 + 6) = 0;
        *(G + N * 2 + 7) = 0;

        *(G + N * 3 + 0) = 0;
        *(G + N * 3 + 1) = 0;
        *(G + N * 3 + 2) = 1;
        *(G + N * 3 + 3) = 0;
        *(G + N * 3 + 4) = 0;
        *(G + N * 3 + 5) = 0;
        *(G + N * 3 + 6) = 0;
        *(G + N * 3 + 7) = -1;

        *(G + N * 4 + 0) = 0;
        *(G + N * 4 + 1) = -1;
        *(G + N * 4 + 2) = 0;
        *(G + N * 4 + 3) = 0;
    }
}

```

```

*(G + N * 4 + 4) = 0;
*(G + N * 4 + 5) = 0;
*(G + N * 4 + 6) = 1;
*(G + N * 4 + 7) = 0;

*(G + N * 5 + 0) = -1;
*(G + N * 5 + 1) = 1;
*(G + N * 5 + 2) = 0;
*(G + N * 5 + 3) = 0;
*(G + N * 5 + 4) = 0;
*(G + N * 5 + 5) = 0;
*(G + N * 5 + 6) = 0;
*(G + N * 5 + 7) = 0;

*(G + N * 6 + 0) = 0;
*(G + N * 6 + 1) = 0;
*(G + N * 6 + 2) = 0;
*(G + N * 6 + 3) = 0;
*(G + N * 6 + 4) = -1;
*(G + N * 6 + 5) = 0;
*(G + N * 6 + 6) = 0;
*(G + N * 6 + 7) = 1;

*(G + N * 7 + 0) = 0;
*(G + N * 7 + 1) = 0;
*(G + N * 7 + 2) = 0;
*(G + N * 7 + 3) = 1;
*(G + N * 7 + 4) = 0;
*(G + N * 7 + 5) = 0;
*(G + N * 7 + 6) = -1;
*(G + N * 7 + 7) = 0;
}
else {
    //N = 8;
    *(G + N * 0 + 0) = 0; // цепь
    *(G + N * 0 + 1) = 0;
    *(G + N * 0 + 2) = -1;
    *(G + N * 0 + 3) = 0;
    *(G + N * 0 + 4) = 0;
    *(G + N * 0 + 5) = 0;
    *(G + N * 0 + 6) = 0;
    *(G + N * 0 + 7) = 0;

    *(G + N * 1 + 0) = 0;
    *(G + N * 1 + 1) = 0;
    *(G + N * 1 + 2) = 0;
    *(G + N * 1 + 3) = 0;
    *(G + N * 1 + 4) = 0;
    *(G + N * 1 + 5) = 1;
    *(G + N * 1 + 6) = -1;
    *(G + N * 1 + 7) = 0;

    *(G + N * 2 + 0) = 1;
    *(G + N * 2 + 1) = 0;
    *(G + N * 2 + 2) = 0;
    *(G + N * 2 + 3) = 0;
    *(G + N * 2 + 4) = 1;
    *(G + N * 2 + 5) = 0;
    *(G + N * 2 + 6) = -1;
    *(G + N * 2 + 7) = -1;

    *(G + N * 3 + 0) = 0;
    *(G + N * 3 + 1) = 0;
    *(G + N * 3 + 2) = 0;
    *(G + N * 3 + 3) = 0;

```

```

*(G + N * 3 + 4) = 0;
*(G + N * 3 + 5) = 0;
*(G + N * 3 + 6) = 1;
*(G + N * 3 + 7) = 0;

*(G + N * 4 + 0) = 0;
*(G + N * 4 + 1) = 0;
*(G + N * 4 + 2) = -1;
*(G + N * 4 + 3) = 0;
*(G + N * 4 + 4) = 0;
*(G + N * 4 + 5) = 0;
*(G + N * 4 + 6) = 0;
*(G + N * 4 + 7) = 1;

*(G + N * 5 + 0) = 0;
*(G + N * 5 + 1) = -1;
*(G + N * 5 + 2) = 0;
*(G + N * 5 + 3) = 0;
*(G + N * 5 + 4) = 0;
*(G + N * 5 + 5) = 0;
*(G + N * 5 + 6) = 1;
*(G + N * 5 + 7) = 0;

*(G + N * 6 + 0) = 0;
*(G + N * 6 + 1) = 1;
*(G + N * 6 + 2) = 1;
*(G + N * 6 + 3) = -1;
*(G + N * 6 + 4) = 0;
*(G + N * 6 + 5) = -1;
*(G + N * 6 + 6) = 0;
*(G + N * 6 + 7) = 0;

*(G + N * 7 + 0) = 0;
*(G + N * 7 + 1) = 0;
*(G + N * 7 + 2) = 1;
*(G + N * 7 + 3) = 0;
*(G + N * 7 + 4) = -1;
*(G + N * 7 + 5) = 0;
*(G + N * 7 + 6) = 0;
*(G + N * 7 + 7) = 0;

}
for (inti = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        printf("%2d ", *(G + N * i + j));
    }
    printf("\n");
}
Cycle();
Chain();
if (isCycle)
{
    printf("Введите стартовую вершину цикла\n ");
    do {
        scanf_s("%d", &startV);
        if (startV > 7)
            printf("Такой вершины не существует введите вершину от 0 до
7\n");
    } while (startV > 7);

    Scan(startV);
    for (inti = k; i >= 0; i--)
        printf("%d ", Stack[i]);
}

```

```
elseif (isChain) {
    printf("Эйлера цепь возможна из вершины %d:\n", 3);
    startV = f1;
    Scan(startV);
    for (inti = k; i >= 0; i--)
        printf("%d ", Stack[i]);
}
_getch();
free(F);
free(G);
return 0;
}
```