

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

Пояснительная записка

к курсовой работе по курсу «Логика
и основы алгоритмизации»
на тему «Реализация алгоритма нахождения Эйлеровых циклов».

Выполнил:

Студент группы 21ВВ1

Жбанников Д.Н.

Принял:

Акифьев И.В.

сумел объяснить

Отлично

23.12.22г.


Пенза 2022

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ 

«___» _____ 20__

ЗАДАНИЕ

на курсовое проектирование по курсу

Логика и основы алгоритмизации в ИЗ
Студенту Кудачикову Дмитрию Николаевичу Группа 21ВВ1.1
Тема проекта "Реализация алгоритма нахождения Эйлеровых циклов"

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения
в соответствии с данными задания курсового проекта.
Пояснительная записка должна содержать:

1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма поставленной задачи;
4. Пример ручного расчёта задачи и вычисления
(на некотором участке работы алгоритма);
5. Описание самой программы;
6. Тесты;
7. Список литературы;
8. Листинг программы;
9. Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчёт работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схема

3. Экспериментальная часть

Тестирование программ

Результаты работы программ на тестовых данных

Срок выполнения проекта по разделам

- 1 Изучение теоретической части курсового
- 2 Разработка алгоритмов программ
- 3 Разработка программ
- 4 Тестирование и завершение разработки программ
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания "20" сентября 2022г

Дата защиты проекта " " "

Руководитель

Акингел И. В. eff

Задание получил

"20" сентября

2022г.

Студент

Иванников Д.Н. mff

Оглавление

Реферат	5
Введение	6
1. Постановка задачи.....	7
2. Теоретическая часть задания	8
3. Описание алгоритма программы	10
4. Описание программы	15
5. Тестирование	18
6. Ручной расчет задачи	21
Заключение.....	23
Список литературы.....	24
Приложение А. Листинг программы.....	25

Реферат

Отчет 25 страниц, 12 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ЭЙЛЕРОВ ЦИКЛ, ЭЙЛЕРОВ ПУТЬ, ПОИСК В ГЛУБИНУ, РЕКУРСИЯ

Цель исследования – разработка программы, способная находить Эйлеров цикл (путь), используя алгоритм поиска в глубину.

В работе рассмотрены правила нахождения Эйлерова цикла, также присутствует возможность определения отсутствия данного цикла.

Добавлена функция записи в файл всех конечных значений.

Введение

Эйлеров путь – называется такой путь в графе, который проходит по каждому ребру, причем ровно один раз. Эйлеров цикл – замкнутый Эйлеров путь. Граф называется Эйлеровым, если он содержит Эйлеров цикл.

Для нахождения Эйлерова цикла понадобится применить алгоритм поиска в глубину, который позволяет построить обход неориентированного графа, при котором посещаются все вершины, доступные из произвольной вершины.

В качестве среды для разработки программы была выбрана Microsoft Visual Studio 2017, язык программирования C/C++.

Целью данной курсовой работы является разработка программы на языке Си, которая определяет является ли граф Эйлеровым и, если это так – находит Эйлеров путь в заданном графе.

1. Постановка задачи

Требуется разработать программу, которая определит Эйлеров цикл в ориентированном графе, используя алгоритм поиска в глубину.

Исходный граф в программе задается матрицей смежности с указанием веса каждого ребра. Программа должна работать так, чтобы пользователь вводил количество вершин графа для случая, а затем вводить расстояние между ними.

После инициализации матрицы – следует ее вывести. После вывода необходимо вывести информацию о матрице: условие существования Эйлера цикла и сам путь, если первое утверждение было положительным.

Устройство ввода – клавиатура.

Символы ввода – цифры 0-9

2. Теоретическая часть задания

Граф – совокупность точек, соединенных линиями. Точки называются вершинами, а линии ребрами. Степенью входа вершины – количество входящих в нее ребер, а степень выхода – количество исходящих ребер.

Графы можно разделить на ориентированные (рис. 1) и неориентированные (рис. 2). Отличие отображения ориентированного и неориентированного графов – обозначение ребер. В ориентированном графе ребро может иметь направление движения, которое обозначается стрелкой.

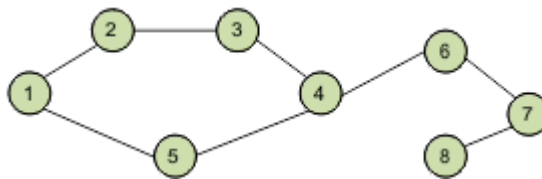


Рисунок 1 - Неориентированный граф

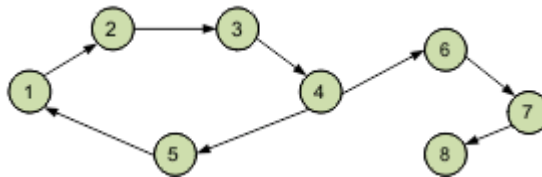


Рисунок 2 - Ориентированный граф

Графы также можно разделить на связные (рис. 1 также является связным) и несвязные (рис. 3).

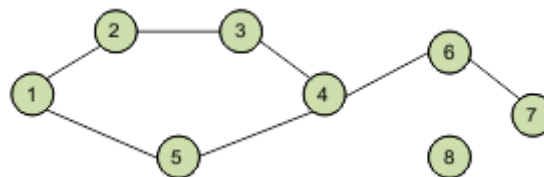


Рисунок 3 - Несвязный граф

Граф может быть представлен несколькими способами:

1. Матрица смежности;
2. Матрица инцидентности;
3. Список смежности или инцидентности;
4. Список ребер.

При представлении графа в виде матрицы смежности – информация о ребрах хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Эйлеров путь в графе – это путь, проходящий по всем ребрам графа и притом только по одному разу. Эйлеров цикл – Эйлеров путь, являющимся циклом, то есть замкнутый путь, проходящий через каждое ребро графа ровно по одному разу. Эйлеров граф – граф содержащий Эйлеров путь.

Согласно теореме, доказанной Эйлером, Эйлеров цикл существует в неориентированном графе тогда и только тогда, когда граф связный или будет являться связным, если удалить из него все изолированные вершины, и в нем отсутствуют вершины нечетной степени.

Для поиска Эйлерова цикла (пути) можно использовать алгоритм обхода графа – поиска в глубину. Поиск в глубину предполагает продвижение в глубь до тех пор, пока это возможно. Невозможность продвижения означает, что следующим шагом будет переход на последнюю, имеющую несколько вариантов движения, ранее посещенную вершину.

3. Описание алгоритма программы

int min:

1. для $i = 0 ; i < n$ делать $i = i+1$
 2. если не $\text{flag}[i]$, $\text{result} = i$
3. для $i = 0 ; i < n$ делать $i = i+1$
 4. если $l[\text{result}] > l[i]$ и не $\text{flag}[i]$, $\text{result} = i$

void main:

1. вызов функции `global`
2. ввод количества вершин
3. ввод размеров ребер
4. вывод матрицы смежности
5. для $i = 0 ; i < n$ делать $i = i+1$
 6. для $j = 0 ; j < n$ делать $j = j+1$
 7. если $c[i][j] = 0$, $c[i][j] = 65535$
- (условие для бесконечности)
8. для $i = 0 ; i < n$ делать $i = i+1$
9. $z = 0$
10. для $j = 0 ; j < n$ делать $j = j+1$
 11. если $c[i][j] \neq 65535$
 12. $z = z+1$
13. если $z \% 2 = 1$
 14. $\text{Mas}[k][0] = i$

15. $Mas[0][k] = i$

16. $k = k + 1$

17. для $m = 1; m < k$ делать $m = m + 1$

18. $xn = Mas[0][m]$

19. для $j = m; j < k-1$ делать $j = j+1$

20. $xk = Mas[j + 1][0];$

21. если $xn \neq xk$

22. вызов функции `deikstra()`

23. вызов функции `eiler()`

void dekstra:

1. для $i = 0; i < n$ делать $i = i+1$

2. $flag[i] = 0$

3. $l[i] = 65535$

4. $l[xn] = 0;$

5. $flag[xn] = 1;$

6. $p = xn;$

7. `_itoa_s(xn + 1, s, 10)`

(преобразование числа в символы)

8. для $i = 0; i < n$ делать $i = i+1$

9. копирование строки 'X', на которую ссылается `path[i]`

10. добавление копии строки `s` в конец строки `path[i]`

11. делать

12. для $i = 0; i < n$ делать $i = i+1$

13. если $((c[p][i] \neq 65535) \text{ и } (\text{не } flag[i])) \ \&\& \ (i \neq p))$

14. для ($l[i] > l[p] + c[p][i]$)

15. `_itoa_s(i + 1, s, 10);`

16. копирование строки `path[p + 1]`, на которую ссылается `path[i + 1]`

17. добавление копии строки "-X" в конец строки `path[i + 1]`

18. добавление копии строки `s` в конец строки `path[i + 1]`

19. $[i] = \text{minim}(l[i], l[p] + c[p][i])$

20. $p = \text{min}(n)$

21. `flag[p] = 1`

22. пока $p \neq x_k$

23. если $[p] \neq 65535$

24. `mas[p].ves = (int)l[p]`

25. $i = 0$

26. пока `path[p + 1][i] != '\0'`

27. `mas[ch].put[i] = path[p + 1][i]`

28. $i++$

29. `mas[ch].x1 = x_k`

30. `mas[ch].x2 = x_n`

31. вывод веса графа

32. вывод пути графа

33. иначе

34. $ch = ch + 1$

void eiler:

1. $\text{sum} = 0$

2. для $i = 0 ; i < n$ делать $i = i + 1$

3. для $j = 0 ; j < n$ делать $j = j + 1$

4. $a[i + 1][j + 1] = (\text{int})c[i][j]$

5.если $a[i + 1][j + 1] = 65535$

6. $a[i + 1][j + 1] = 0$

7. $\text{count} = 0$

8. для $i = 0 ; i < n$ делать $i = i + 1$

9.если $\text{flag}[i] = 0$

10. $\text{count} = \text{count} + 1$

10.если $\text{count} > 1$

11.вызов функции $\text{no}()$

12. для $i = 0 ; i < n$ делать $i = i + 1$

13.если $\text{vert}[i] \% 2 = 1$

14. вызов функции $\text{no}()$

15. $w = 0$

16. вызов функции $\text{poisk}(1)$

17. $\text{way}[0] = \text{way}[1]$

18. для $i = 0 ; i < w$ делать $i = i + 1$

19.если $c[\text{way}[i] - 1][\text{way}[i + 1] - 1] == 65535$

20.для $j = 0; j < \text{ch}; j++$

21.если $((\text{mas}[j].x1 + 1 == \text{way}[i] \text{ и } \text{mas}[j].x2 + 1 == \text{way}[i + 1]) |$
 $(\text{mas}[j].x2 + 1 == \text{way}[i] \text{ и } \text{mas}[j].x1 + 1 == \text{way}[i + 1]))$

22.Вывод пути

23. $\text{sum} = \text{sum} + \text{mas}[p].\text{ves}$

24. $\text{way}[0] = \text{mas}[p].x1$

25.иначе вывести путь

26.если way[0] != way[w]

27.если c[way[1] - 1][way[w] - 1] не = 65535

28.вывести way[1]

29. sum = sum + (int)c[way[i] - 1][way[w] - 1]

30.если

31. для j = 0 ; j < ch делать j = j+1

32. mas[p].x1 == way[1] и mas[p].x2 == way[w - 1]

33.вывести путь

34. sum = sum + mas[p].ves

35. для i = 0 ; i <= w делать i = i+1

36.если c[way[i] - 1][way[i + 1] - 1] не = 65535

37. sum = (int)c[way[i] - 1][way[i + 1] - 1] + sum

37.вывести вес пути sum

void no:

1.вывести "Эйлеров цикл не существует!"

2. exit(0)

void poisk:

1.для j = 1; j <= n; j = j + 1

2.для a[i][j] не = 0

3. a[i][j] = 0

4. a[j][i] = 0

5.вызов функции poisk(j)

6.w = w + 1

7. way[w] = i

4. Описание программы

Для реализации программы использован язык программирования Си/ Си++. Проект создан в виде консольного приложения.

Работа программы начинается с листа ознакомления, после чего идет подтверждение начала работы.

```
пгу
Кафедра ВТ
КУРСОВАЯ РАБОТА
На тему
Реализация алгоритма нахождения
Эйлеровых циклов
Выполнил:
студент группы 21вв1
Жбанников Д.Н.
Принял:
Акифьев И.В.
Пенза 2022
Нажмите 1, если хотите продолжить : 1
```

Рисунок 4 – титульный лист проекта

Далее начинается основная программа.

Пользователь должен выбрать количество вершин в графе. Затем ему предстоит указать расстояние между каждой вершиной.

```

Нажмите 1, если желаете продолжить: 1
введите количество вершин в графе: 4
введите расстояние x1 до x2:1
введите расстояние x1 до x3:2
введите расстояние x1 до x4:3
введите расстояние x2 до x3:4
введите расстояние x2 до x4:2
введите расстояние x3 до x4:2

```

Рисунок 5 – условие, задающие граф

В случае, если пользователь ввел параметр для графа, которого нет, то программа завершит свою работу выдав соответствующее сообщение. Иначе, если был выбран один из предложенных вариантов, то после выполнения, текущая матрица смежности будет выведена в консоль.

Ниже приведен пример начального запроса и дальнейшие действия с ним.

```

введите количество вершин в графе: 4
введите расстояние x1 до x2:2
введите расстояние x1 до x3:3
введите расстояние x1 до x4:4
введите расстояние x2 до x3:1
введите расстояние x2 до x4:5
введите расстояние x3 до x4:6
  x1 x2 x3 x4

x1  0  2  3  4
x2  2  0  1  5
x3  3  1  0  6
x4  4  5  6  0

```

Рисунок 6 – вывод матрицы смежности

После чего происходит проверка на условие существования Эйлера цикла и выводится соответствующее сообщения. Если результат был положительным, то выводит найденный Эйлеров путь


```

2 вес
X1-X2 путь
3 вес
X1-X3 путь
4 вес
X1-X4 путь
1 вес
X2-X3 путь
5 вес
X2-X4 путь
6 вес
X3-X4 путь
X2-X3-X4-X1-X3-X2-X1-X2
веспути 65552

```

Рисунок 4 – вывод Эйлерового пути

Если условия пользователя не соответствуют нужным, то на экран будет выведено данное сообщение.

```

Нажмите 1, если желаете продолжить: 1
введите количество вершин в графе:
4
введите расстояние x1 до x2:0
введите расстояние x1 до x3:0
введите расстояние x1 до x4:0
введите расстояние x2 до x3:2
введите расстояние x2 до x4:3
введите расстояние x3 до x4:4
  X1 X2 X3 X4
X1  0  0  0  0
X2  0  0  2  3
X3  0  2  0  4
X4  0  3  4  0
Эйлеров цикл не существует!

```

Рисунок 8 – цикла не существует

5. Тестирование

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывод результата выполнения.

```
Нажмите 1, если желаете продолжить: 1
введите количество вершин в графе: 4
введите расстояние x1 до x2:2
введите расстояние x1 до x3:3
введите расстояние x1 до x4:4
введите расстояние x2 до x3:1
введите расстояние x2 до x4:5
введите расстояние x3 до x4:6
  X1 X2 X3 X4

X1  0  2  3  4
X2  2  0  1  5
X3  3  1  0  6
X4  4  5  6  0

2 вес
X1-X2 путь
3 вес
X1-X3 путь
4 вес
X1-X4 путь
1 вес
X2-X3 путь
5 вес
X2-X4 путь
6 вес
X3-X4 путь
X2-X3-X4-X1-X3-X2-X1-X2
вспути 65552
```

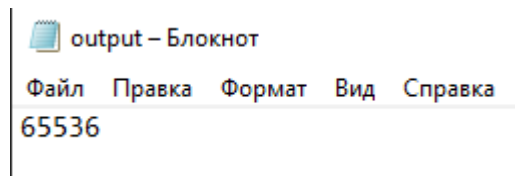
Рисунок 9 - Тестирование при вводе 4 вершин

```

2 вес
X1-X2 путь
4 вес
X1-X3 путь
5 вес
X1-X6-X4 путь
4 вес
X1-X6-X5 путь
3 вес
X1-X6 путь
4 вес
X2-X3 путь
6 вес
X2-X5-X4 путь
3 вес
X2-X5 путь
4 вес
X2-X6 путь
6 вес
X3-X6-X4 путь
5 вес
X3-X6-X5 путь
4 вес
X3-X6 путь
3 вес
X4-X5 путь
2 вес
X4-X6 путь
1 вес
X5-X6 путь

```

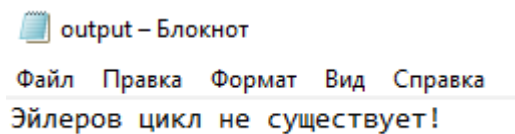
Рисунок 10 - Тестирование при вводе 6 вершин



output - Блокнот

Файл Правка Формат Вид Справка

65536



output - Блокнот

Файл Правка Формат Вид Справка

Эйлеров цикл не существует!

Рисунок 11 – проверка вывод результата в файл

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод сообщения о выборе: ввести с консоли.	Верно
Выбор ввода с консоли	Вывод сообщения о количестве вершин графа и ввод ребер	Верно
Вывод матрицы с 6 вершинами	Вывод сообщения о выполнении условия Эйлера цикла и вывод Эйлера пути	Верно
Вывод матрицы с 4 вершинами	Вывод сообщения о выполнении условия Эйлера цикла и вывод Эйлера пути	Верно
Вывод матрицы из файла с 10 вершинами	Вывод сообщения о выполнении условия Эйлера цикла и вывод Эйлера пути	Верно
Ввод 4 вершин, не соответствующих Эйлерову циклу	Вывод сообщения о невыполнении условия Эйлера цикла	Верно

В результате тестирования было выявлено, что программа корректно отрабатывает и выводит верные результаты.

6. Ручной расчет задачи

Проведем проверку работы программы ручным расчетом на примере графа с 10 вершинами (рис. 9). Для простоты понимания приведем графическое представление графа вместе с матрицей смежности (рис. 12).

	0	1	2	3	4	5	6	7	8	9
0						1	1			
1					1			1		
2								1	1	
3						1		1	1	1
4		1				1	1		1	
5	1			1	1		1			
6	1				1	1		1		
7		1	1	1			1			
8			1	1	1					1
9				1					1	

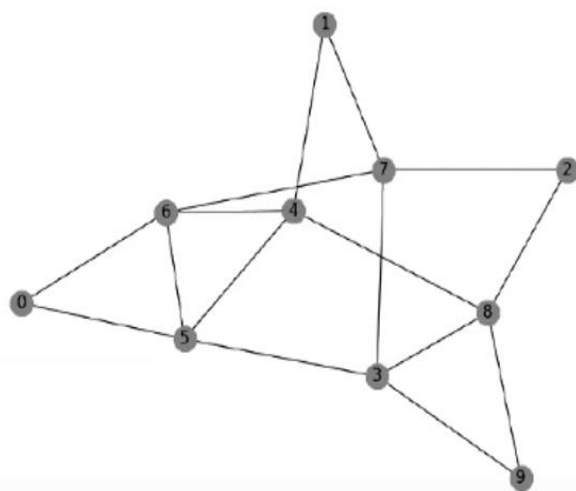


Рисунок 5 - Графическое представление графа для ручного расчета

Сперва проверим условие существования Эйлера цикла, для этого нужно, чтобы каждая вершина имела четную степень. Из рисунка 12 видно, что из каждой вершины выходит четное количество ребер, а это значит, что условие выполняется и можно продолжать последующее вычисление Эйлера цикла (пути).

// Начнем обход с вершины 6 в последующие. Проверяем, есть ли путь из 6 в другие вершины, то дальше идем. Т.к. проверка пути происходит с меньшей нумерации вершины, то следующей будет вершина 0, но перед тем, как перейти к ней – нужно очистить этот путь, чтобы его снова не использовать.

Аналогично ищем путь из вершины 0. Это будет вершина 5. Повторяем те же операции, что описаны выше и переходим к вершине 3. Из вершины 3 передвигаемся в вершину 7. Из вершины 7 в вершину 1, далее в вершину 4, следующая вершина 5. Из вершины 5 в вершину 6, после чего в вершину 4, после в вершину 8. Из вершины 8 есть путь в вершину 2, из 2 в 7, и наконец вернулись в вершину 6, но остались еще не посещенные ребра, поэтому рекурсивно возвращаемся к вершине у которой еще остались пути. Это будет вершина 8. Из вершины 8 есть путь к вершине 3. Вершина

3 содержит путь к вершине 9, а вершина 9 содержит путь до 8. И вот теперь из вершины 8 можно дойти через вершины 2 и 7 до вершины 6.

Теперь можно рекурсивно возвращаясь к началу выводить посещенные вершины и результат будет следующим: {6, 7, 2, 8, 9, 3, 8, 4, 6, 5, 4, 1, 7, 3, 5, 0, 6}.

Сравнивая полученный результат посредством ручного расчета с вычисленным результатом программы (рис. 9) можно сделать вывод, что программа работает верно. //

Заключение

В процессе создания проекта разработана программа, реализующая алгоритм поиска в глубину для поиска Эйлера цикла (пути) графа в Microsoft Visual Studio 2017.

При выполнении курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, также получен опыт создания графической части программы. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания языка программирования Си, в том числе Си ++.

Недостатком разработанной программы можно считать интерфейс, который не отображает интуитивно понятного представления графа из-за того, что вся информация выводится в консоль. Также, возможно выбран не лучший способ ввода размера ребер, возможно в дальнейшем стоило бы сделать это в виде графической части, например, используя ресурсы Си #.

Список литературы

1. wikipedia.org
2. CyberForum.ru
3. Язык программирования С [2009] Керниган, Ритчи
4. Expert C Programming Deep C Secrets Peter van der Linden

Приложение А. Листинг программы.

```
//С-матрица смежности,с расстояниями
//xn-начальная точка
//xk-конечная точка
#include "locale.h"

#include<iostream>
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include "locale.h"
#define word unsigned int
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(suppress : 4996)
std::ofstream fout("output.txt");
using namespace std;
void deikstra();
void no();
void komponenta(int i);
void eiler();
void poisk(int i);
int a[50][50];
int i, j, p, xn, xk, z, k = 1, Mas[100][100], ch = 0;
int vert[10000]; //степень вершин
int way[10000]; //Эйлеров цикл
int flag[10000]; //компоненты связности
int x, y, w;
int n, m; // m - число дуг, n - число вершин
int count; // число компонент связности
word c[50][50], l[50];
char s[80], path[80][50];
struct st {
    char put[50];
    int x1;
    int x2;
    int ves;
};
st mas[100];
int min(int n)
{
    int i, result;
    for (i = 0; i < n; i++)
        if (!(flag[i])) result = i;
    for (i = 0; i < n; i++)
        if ((l[result] > l[i]) && (!flag[i])) result = i;
    return result;
}
word minim(word x, word y)
{
    if (x < y) return x;
    return y;
}
void global()
{
    int startik;
    setlocale(LC_ALL, "Rus");
    printf("_____ \n");
    printf("_____ ПГУ _____ \n");
    printf("_____ \n");
    printf("_____ Кафедра ВТ _____ \n");
```

```

printf("_____\n");
printf("_____КУРСОВАЯ РАБОТА_____\n");
printf("_____\n");
printf("_____На тему_____\n");
printf("_____\n");
printf("_____Реализация алгоритма нахождения_____\n");
printf("_____\n");
printf("_____Эйлеровых циклов_____\n");
printf("_____\n");
printf("_____\n");
printf("_____\n");
printf("_____Выполнил:_____\n");
printf("_____\n");
printf("_____студент группы 21вв1_____\n");
printf("_____\n");
printf("_____Жбанников Д.Н._____\n");
printf("_____\n");
printf("_____Принял:_____\n");
printf("_____\n");
printf("_____Акифьев И.В._____\n");
printf("_____\n");
printf("_____\n");
printf("_____\n");
printf("_____Пенза 2022_____\n");
printf("_____\n");
printf("_____\n");

printf("Нажмите 1, если желаете продолжить: ");
scanf_s("%d", &startik);
if (startik == 1)
{
    st();
}

_getch();
}
void main(int argc, char* argv[])
{
    global();
    setlocale(LC_ALL, "Russian");
    cout << "введите количество вершин в графе: ";
    cin >> n;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) c[i][j] = 0;
    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
        {
            cout << "введите расстояние x" << i + 1 << " до x" << j + 1 << ":\n";
            cin >> c[i][j]; //записываем расстояния в матрицу c
        }
    cout << "\n";
    for (i = 0; i < n; i++) cout << " X" << i + 1;
    cout << endl << endl;
    for (i = 0; i < n; i++)
    {
        printf("X%d", i + 1);
        for (j = 0; j < n; j++)
        {
            printf("%6d", c[i][j]);
            c[j][i] = c[i][j];
        }
        printf("\n\n");
    }
}

```

```

    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            if (c[i][j] == 0)
                c[i][j] = 65535;
    } //бесконечность
    for (i = 0; i < n; i++)
    {
        z = 0;
        for (j = 0; j < n; j++) {
            if (c[i][j] != 65535)
                z++;
        }
        if (z % 2 == 1) //ищем вершины с нечетными степенями
        {
            Mas[k][0] = i; //сохраняем номера этих вершин
            Mas[0][k] = i;
            k++;
        }
    }
    for (int m = 1; m < k; m++) {
        xn = Mas[0][m];
        for (j = m; j < k - 1; j++)
        {
            xk = Mas[j + 1][0];
            if (xn != xk)
                deikstra();
        }
    }
    eiler();
    getchar();
}

void deikstra()
{
    for (i = 0; i < n; i++)
    {
        flag[i] = 0;
        l[i] = 65535;
    }

    l[xn] = 0;
    flag[xn] = 1;
    p = xn;
    _itoa_s(xn + 1, s, 10); //преобразование числа в символы
    for (i = 1; i <= n; i++)
    {
        strcpy_s(path[i], "X");
        strcat_s(path[i], s);
    }
    do
    {
        for (i = 0; i < n; i++)
            if ((c[p][i] != 65535) && (!flag[i]) && (i != p))
            {
                if (l[i] > l[p] + c[p][i])
                {
                    _itoa_s(i + 1, s, 10);
                    strcpy_s(path[i + 1], path[p + 1]);
                    strcat_s(path[i + 1], "-X");
                    strcat_s(path[i + 1], s);
                }
                l[i] = minim(l[i], l[p] + c[p][i]);
            }
    }
}

```

```

        p = min(n);
        flag[p] = 1;
    } while (p != xk);
    if (l[p] != 65535)
    {
        mas[p].ves = (int)l[p];
        i = 0;
        while (path[p + 1][i] != '\0') {
            mas[ch].put[i] = path[p + 1][i];
            i++;
        }
        mas[ch].x1 = xk;
        mas[ch].x2 = xn;
        printf("%i вес\n", mas[p].ves);
        printf("%s путь \n", mas[ch].put);
    }
    else
        printf( "Эйлеров цикл!");
    ch++;
}
void euler()
{
    int sum = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
            a[i + 1][j + 1] = (int)c[i][j];
            if (a[i + 1][j + 1] == 65535)
                a[i + 1][j + 1] = 0;
        }
    int count = 0;
    for (int i = 1; i < n; i++)
    {
        if (flag[i] == 0)
            count++;
        if (count > 1)
            no(); // граф несвязен
            //komponenta(i);
    }
    for (int i = 1; i < n; i++)
        if (vert[i] % 2 == 1)
            no(); // есть вершины нечётной степени

    w = 0;
    poisk(1);
    way[0] = way[1];
    for (int i = 1; i <= w; i++) {
        if ((int)c[way[i] - 1][way[i + 1] - 1] == 65535) {
            for (j = 0; j < ch; j++) {
                if ((mas[j].x1 + 1 == way[i] && mas[j].x2 + 1 == way[i + 1]) ||
                    (mas[j].x2 + 1 == way[i] && mas[j].x1 + 1 == way[i + 1])) {
                    printf("%s-", mas[j].put);
                    sum = sum + mas[j].ves;
                    way[0] = mas[j].x1;
                }
            }
        }
        else
            printf("X%i-", way[i]);
    }
    if (way[0] != way[w]) {
        if ((int)c[way[1] - 1][way[w] - 1] != 65535) {
            printf("X%i", way[1]);
            sum = sum + (int)c[way[i] - 1][way[w] - 1];
        }
        else {

```

```

        for (j = 0; j < ch; j++) {
            if (mas[p].x1 == way[1] && mas[p].x2 == way[w - 1]) {
                printf("%s-", mas[p].put);
                sum = sum + mas[p].ves;
            }
        }
    }
}
for (int i = 0; i <= w; i++) {
    if ((int)c[way[i] - 1][way[i + 1] - 1] != 65535)
        sum = (int)c[way[i] - 1][way[i + 1] - 1] + sum;
}
printf("\n");
printf("вспути %i ", sum);
fout << sum;
}
//-----
void no() {
    printf("Эйлеров цикл не существует!");
    fout << "Эйлеров цикл не существует!";
    exit(0);
    _getch();
}
void poisk(int i) {
    int j;
    for (int j = 1; j <= n; j++)
        if (a[i][j] != 0) {
            a[i][j] = 0;
            a[j][i] = 0;
            poisk(j);
        }
    w++;
    way[w] = i;
}
}

```