

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**Содержание**

[Реферат 5](#_Toc122073280)

[Введение 6](#_Toc122073281)

[1. Постановка задачи 7](#_Toc122073282)

[2. Теоретическая часть задания 8](#_Toc122073283)

[3. Описание алгоритма программы 10](#_Toc122073284)

[4. Описание программы 18](#_Toc122073285)

[5. Тестирование 24](#_Toc122073286)

[6. Ручной расчет задачи 29](#_Toc122073287)

[Заключение 30](#_Toc122073288)

[Список литературы 31](#_Toc122073289)

[Приложение А. Листинг программы 32](#_Toc122073290)

# **Реферат**

Отчет 38 страниц, 10 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 поиска в глубину, на основе которых можно построить Эйлеров цикл. Рассмотрены условия существования Эйлерова цикла.

# **Введение**

Если граф имеет цикл (не обязательно простой), содержащий все ребра графа по одному разу, то такой цикл называется эйлеровым циклом, а граф называется эйлеровым графом. Если граф имеет цепь (не обязательно простую), содержащую все ребра графа по одному разу, то такая цепь называется эйлеровой цепью, а граф называется полуэйлеровым графом. Эйлерову цепь также иногда называют эйлеровым путем.

Для того чтобы найти Эйлеров цикл в графе, будем использовать поиск в глубину, но только здесь пройденными будут помечаться не вершины, а ребра графа.

В качестве среды разработки мною была выбрана среда разработки Microsoft Visual Studio 2019, язык программирования Си.

Целью данной курсовой работы является разработка программы на языке Си, определяющей удовлетворение графа условиям существования Эйлерова цикла, впоследствии – реализующей Эйлеров цикл на данном графе (при удовлетворении этих условий).

# **Постановка задачи**

Необходимо разработать алгоритм нахождения Эйлеровых циклов в неориентированном графе с использованием алгоритма поиска в глубину.

Граф задается матрицей смежности либо с клавиатуры (пользователь сам вводит значения матрицы), либо считывается из файла, либо задается случайно.

Для выбора задания матрицы реализуется текстовое меню. Матрица выводится на экран, после чего пользователь или выбирает вершину, с которой начнется Эйлеров цикл, или добавляет ребра в граф, если не выполнено условие существования Эйлерова цикла (если в данном графе есть такая возможность).

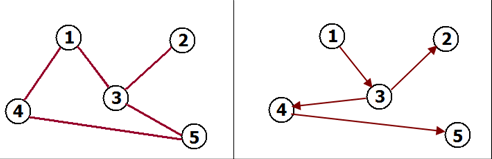
Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

# **Теоретическая часть задания**

Граф — это абстрактное представление множества объектов и связей между ними. Графом называют пару (V, E) где V это множество вершин, а E множество пар, каждая из которых представляет собой связь (эти пары называют рёбрами).

Граф может быть ориентированным или неориентированным. В ориентированном графе, связи являются направленными (то есть пары в E являются упорядоченными, например пары (a, b) и (b, a) это две разные связи). В свою очередь в неориентированном графе, связи ненаправленные, и поэтому если существует связь (a, b) то значит, что существует связь (b, a).

Рисунок 1 – Структура модулей программы



Существует два способа представления графа, в виде списков смежности и в виде матрицы смежности. Оба способа подходят для представления ориентированных и неориентированных графов.

*Матрица смежности.* Этот способ является удобным для представления плотных графов, в которых количество ребер (|E|) примерно равно количеству вершин в квадрате (|V|2).

В данном представлении заполняется матрица размером V x V следующим образом:

A[i][j] = 1 (Если существует ребро из i в j)

A[i][j] = 0 (Ребра между i и j не существует)

Для неориентированных графов матрица A является симметричной. Этот способ очень громоздкий, так как требует O (|V|2) памяти для хранения матрицы.

*Список смежности.* Данный способ представления больше подходит для разреженных графов, то есть графов, у которых количество ребер гораздо меньше чем количество вершин в квадрате (|E| << |V|2).

В данном представлении используется массив Adj содержащий |V| списков. В каждом списке Adj[v] содержатся все вершины u, так, что между v и u есть ребро. Память, требуемая для представления равна O (|E| + |V|) что является лучшим показателем, чем матрица смежности для разреженных графов.

Главный недостаток этого способа представления в том, что нет быстрого способа проверить, существует ли ребро (u, v).

Эйлеров путь в графе – это путь, проходящий по всем ребрам графа 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.

Эйлеров путь в графе существует тогда и только тогда, когда граф связный и содержит не более двух вершин нечётной степени. Ввиду [леммы о рукопожатиях](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BC%D0%BC%D0%B0_%D0%BE_%D1%80%D1%83%D0%BA%D0%BE%D0%BF%D0%BE%D0%B6%D0%B0%D1%82%D0%B8%D1%8F%D1%85), число вершин с нечётной степенью должно быть чётным. А значит, Эйлеров путь существует только тогда, когда это число равно нулю или двум. Причём, когда оно равно нулю, Эйлеров путь вырождается в Эйлеров цикл.

# **Описание алгоритма программы**

Ниже представлен псевдокод функций check\_cycle(), eulerian\_cycle().

Функция check\_cycle() сначала проверяет, удовлетворяет ли граф условиям существования Эйлерова цикла. Если нет, то предлагает пользователю выбрать – нужно ли достраивать (удалять) ребра в графе для того, чтобы в нем можно было реализовать Эйлеров цикл. Далее алгоритм запрашивает вершину, с которой он начнется (и закончится), после чего выведет полный проход по ребрам графа на консоль. Если вершина была введена некорректно, то за начальную вершину будет принята максимальная по значению вершина графа (на 1 меньше количества вершин графа).

**check\_cycle()**

1.для i=0 пока i<size делать i=i+1

2.для j=0 пока j<size делать j=j+1

3.если graph[i][j] == 1 то

4.count += graph[i][j]

5.конеццикла

6.если (count % 2 != 0) ИЛИ (count == 0)

7.flag=1

8.конеццикла

9.если flag==1

10.вывод "В данном графе эйлерова цикла не существует! Хотите достроить (удалить) ребра в графе для получения эйлерова цикла?\n+ да, - нет\n\n"

11.ввод choise2

12.если choise2 == 45 то

13.вернуть 0

14.если choise2 == 43 то

15.degree = (int\*)malloc(sizeof(int\*) \* (size));

16.для i=0 пока i<size делать i=i+1

17.degree[i] = 0;

18.конеццикла

19.для i=0 пока i<size делать i=i+1

20.для j=0 пока j<size делать j=j+1

21.если graph[i][j] == 1 то

22.degree[i]++;

23.конеццикла

24.если (degree[i] == size - 1) И ((size - 1) % 2 != 0) то

25.если o == -1 то

26.o = i;

27.иначе

28.graph[o][i] = 0;

29.graph[i][o] = 0;

30.вывод "Удалено ребро [o]-[i]"

31.degree[o]--;

32.degree[i]--;

33.o = -1;

34.конецусловия

35.конецусловия

36.если degree[i] == 0 то

37.если p == -1 то

38.p = i;

39.иначе

40.graph[p][i] = 1;

41.graph[i][p] = 1;

42.вывод "Добавлено ребро [p]-[i]\n"

43.degree[i]++;

44.degree[p]++;

45.p = -1;

46.конецусловия

47.конецусловия

48.конеццикла

49.для i=0 пока i<size делать i=i+1

50.если o != -1 то

51.если graph[o][i] == 1 то

52.graph[o][i] = 0;

53.graph[i][o] = 0;

54.вывод "Удалено ребро [o]-[i]\n"

55.degree[i]--;

56.degree[o]--;

57.o = -1;

58.конецусловия

59.конецусловия

60.если p != -1 то

61.если (graph[i][p] == 0) И (degree[i] % 2 != 0)

62.graph[p][i] = 1;

63.graph[i][p] = 1;

64.вывод "Добавлено ребро [p]-[i]\n"

65.degree[i]++;

66.degree[p]++;

67.p = -1;

68.конецусловия

69.конецусловия

70.если (degree[i] % 2 != 0) И (o == -1) то

71.если u == -1 то

72.u = i;

73.иначе

74.если graph[u][i] == 0 то

75.graph[u][i] = 1;

76.graph[i][u] = 1;

77.вывод "Добавлено ребро [u]-[i]\n"

78.degree[i]++;

79.degree[u]++;

80.u = -1;

81.конецусловия

82.конецусловия

83.конецусловия

84.конеццикла

85.для i=0 пока i<size делать i=i+1

86.если degree[i] % 2 != 0 то

87.если u == -1 то

88.u = i;

89.иначе

90.если graph[u][i] == 0

91.graph[u][i] = 1;

92.graph[i][u] = 1;

93.вывод "Добавлено ребро [u]-[i]\n"

94.degree[i]++;

95.degree[u]++;

96.u = -1;

97.конецусловия

98.конецусловия

99.конецусловия

100.если degree[i] == 0 то

101.для h=0 пока h<size делать h=h+1

102.если degree[h] != 0 то

103.для j=0 пока j<size делать j=j+1

104.если graph[h][j] == 1 то

105.graph[h][j] = 0;

106.graph[j][h] = 0;

107.вывод "Удалено ребро [h]- [j]\n"

108.graph[h][i] = 1;

109.graph[i][h] = 1;

110.graph[j][i] = 1;

111.graph[i][j] = 1;

112.вывод "Добавлено ребро [h]-[i]\n"

113.вывод "Добавлено ребро [i]-[j]\n"

114.degree[i]+=2;

115.break;

116.конецусловия

117.конеццикла

118.break;

119.конецусловия

120.конецццикла

121.конецусловия

122.конеццикла

123.вывод "\n"

124.вызвать функцию print\_graph(graph, size)

125.конецусловия

126.если (choise2 != 43) И (choise2 != 45) то

127.вывод "Неверный ввод!\n\n"

128.вернуть 0

129.конецусловия

130.конецусловия

131.вывод "В данном графе эйлеров цикл существует. Введите вершину, с которой начнется цикл: "

132.ввод cycle\_start

133.если (cycle\_start >= size) ИЛИ (cycle\_start < 0)

134.вывод "Введенной вершины в графе не существует! Вершиной начала цикла будет являться максимальная по значению вершина графа.\n"

135.cycle\_start = size - 1;

136.конецусловия

137.вывод "Эйлеров цикл: "

138.вызвать функцию eulerian\_cycle(graph, size, cycle\_start)

139.вывод "\n\n"

Функция eulerian\_cycle() реализует непосредственно алгоритм прохода по графу, нахождения в нем и вывода на экран Эйлерова цикла. Алгоритм основан на обходе графа в глубину – только здесь он идет не по вершинам, а по ребрам, отмечая их посещенными.

**eulerian\_cycle()**

1.для cycle\_next=0 пока cycle\_next<size делать cycle\_next=cycle\_next +1

2.если graph[cycle\_start][cycle\_next] == 1

3.graph[cycle\_start][cycle\_next] = 0

4.graph[cycle\_next][cycle\_start] = 0

5.вывод cycle\_start-cycle\_next

6.eulerian\_cycle(graph, size, cycle\_next);

7.конецусловия

8.конеццикла

Алгоритм имеет ограниченное применение: предназначен для работы с неориентированным графом. В силу того, что реализуется на матрице смежности – не подходит для взвешенного графа, для мультиграфа. Исключена возможность ребра-петли. Подходит как для связного, так и для несвязного графа (присутствие изолированных вершин возможно).

# **Описание программы**

Для написания данной программы использован язык программирования Си. Данная программа является многомодульной, поскольку состоит из нескольких функций: main, vvod\_file, print\_file, check\_cycle и др.

Работа программы начинается с запроса о выборе инициализации матрицы.

int main()

{

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

printf("Курсовая работа по предмету Логика и основы алгоритмизации в инженерных задачах\nТема: Реализация алгоритма нахождения эйлеровых циклов\nВыполнил: Давкин Максим, 21ВВ1.3\n\n");

int size = 0, choise1, \*\*graph = NULL;

while (true)

{

printf("Выберите нужное действие:\n1. Ввести матрицу смежности вручную\n2. Сгенерировать матрицу смежности (случайные значения)\n3. Cчитать матрицу смежности из файла\nEsc - Выход\n\n");

choise1 = \_getch();

if (choise1 == '1')

graph = vvod\_graph(&size);

if (choise1 == '2')

graph = generation\_graph(&size);

if (choise1 == '3')

graph = file\_graph(&size);

if (choise1 == 27)

exit(0);

if ((choise1 != '1') && (choise1 != '2') && (choise1 != '3') && (choise1 != 27)) {

printf("Неверный ввод!\n");

continue;

}

Если пользователь выбрал ввести матрицу смежности с консоли, то будет предложено ввести количество вершин графа, а далее вводить саму матрицу.

int\*\* vvod\_graph(int \*size)

{

printf("Введите количество вершин графа: ");

scanf("%d", size);

printf("Введите матрицу смежности, оставляя пробелы между соседними значениями:\n");

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

for (int i = 0; i < (\*size); i++)

{

graph[i] = (int\*)malloc(sizeof(int) \* (\*size));

}

for (int i = 0; i < (\*size); i++)

{

for (int j = 0; j < (\*size); j++)

{

scanf("%d", &graph[i][j]);

}

}

for (int i = 0; i < (\*size); i++)

for (int j = 0; j < (\*size); j++)

if ((graph[i][j] != graph[j][i]) || (graph[i][i] != 0)) {

printf("Неверный ввод матрицы!\n");

exit(0);

}

printf("Введенная матрица:\n");

return graph;

}

Если пользователь выбрал сгенерировать граф (матрицу смежности) случайно, то будет предложено ввести количество вершин графа.

int\*\* generation\_graph(int \*size)

{

printf("Введите количество вершин графа: ");

scanf("%d", size);

printf("\n");

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

for (int i = 0; i < (\*size); i++)

{

graph[i] = (int\*)malloc(sizeof(int) \* (\*size));

}

for (int i = 0; i < (\*size); i++)

{

for (int j = i; j < (\*size); j++)

{

if (i == j)

{

graph[i][j] = 0;

}

else {

graph[i][j] = rand() % 2;

graph[j][i] = graph[i][j];

}

}

}

return graph;

}

Если пользователь выбрал ввести матрицу смежности из файла, то необходимо, чтобы существовал текстовый файл формата txt, в нем содержалось количество вершин графа, ниже матрица смежности.

int\*\* file\_graph(int \*size)

{

char input\_graph[20];

printf("Файл должен содержать в себе количество вершин графа, ниже - матрицу смежности. Пример ввода: privet.txt\nВведите название файла: ");

scanf("%s", &input\_graph);

FILE\* f1 = fopen(input\_graph, "r");

if (f1 == NULL)

{

printf("Не удалось открыть файл!\n");

return NULL;

}

fscanf(f1, "%d", size);

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

for (int i = 0; i < (\*size); i++)

{

graph[i] = (int\*)malloc(sizeof(int) \* (\*size));

for (int j = 0; j < (\*size); j++)

{

fscanf(f1, "%d", &graph[i][j]);

}

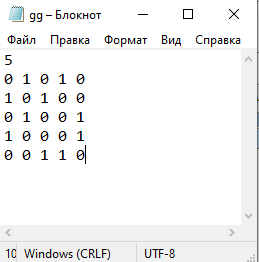
}

fclose(f1);

return graph;

}

Рисунок 2 – Пример файла с матрицей смежности



После выбора одного из вышеперечисленных вариантов матрица смежности будет выведена в консоль.

void print\_graph(int\*\* graph, int size)

{

printf(" ");

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

{

printf("[%d]", i);

}

printf("\n");

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

{

printf("[%d]", i);

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

{

printf(" %d ", graph[i][j]);

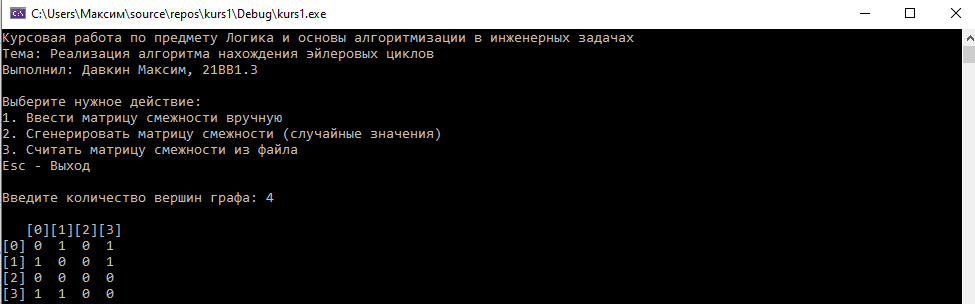
}

printf("\n");

}

}

Рисунок 3 – Пример выполнения формирования случайной матрицы



Далее происходит проверка на условие существования Эйлерова цикла. Если в данном графе он возможен – пользователю будет предложено ввести вершину, с которой начнется цикл. Если цикл отсутствует, то будет предложено достроить ребра в графе для соответствия графу условиям существования Эйлерова цикла.

int check\_cycle(int\*\* graph, int size)

{

int flag = 0;

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

{

int count = 0;

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

{

if (graph[i][j] == 1)

count += graph[i][j];

}

if ((count % 2 != 0) || (count == 0))

{

flag = 1;

}

}

if (flag == 1) {

int m = -1, nul = -1, chet = 0;

printf("В данном графе эйлерова цикла не существует! Хотите достроить ребра в графе для получения эйлерова цикла?\n+ да, - нет\n\n");

int choise2 = \_getch();

if (choise2 == 45)

return 0;

if (choise2 == 43) {

<…>

}

if ((choise2 != 45) && (choise2 != 43)) {

printf("Неверный ввод!\n\n");

return 0;

}

}

int cycle\_start = 0;

printf("В данном графе эйлеров цикл существует. Введите вершину, с которой начнется цикл: ");

scanf("%d", &cycle\_start);

if (cycle\_start >= size || cycle\_start < 0)

{

printf("Введенной вершины в графе не существует! Вершиной начала цикла будет являться максимальная по значению вершина графа.\n");

cycle\_start = size - 1;

}

printf("Эйлеров цикл: ");

eulerian\_cycle(graph, size, cycle\_start);

printf("\n\n");

}

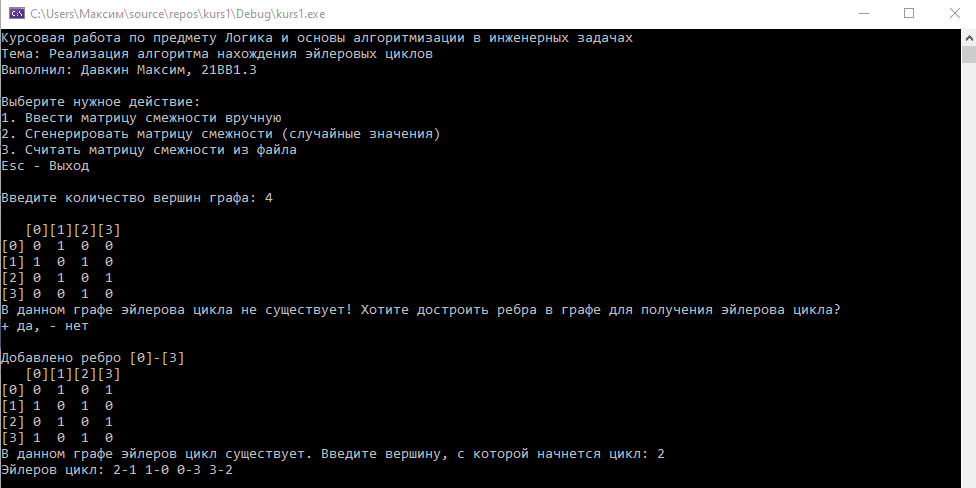


Рисунок 4 – Эйлеров цикл в преобразованной случайной матрице

# **Тестирование**

Средой разработки была выбрана программа Visual Studio 2019. Программа содержит в себе все необходимые средства для успешной отладки и тестирования модулей программы, а также всей программы в целом. Данная среда разработки выпущена относительно недавно, что позволяет проводить необходимые операции над разработанной программой наиболее производительно и эффективно.

Тестирование проводилось в рабочем порядке, на всех этапах разработки программы, в т.ч. после завершения написания кода. Тестирование позволило выявить множество проблем и недочетов по части внешнего вида матрицы, соответствия пунктов меню функциям и пр. Для эффективной работы программы были созданы проверки, не позволяющие проводить обработку некоторых частей программы в определенных условиях.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных данных.

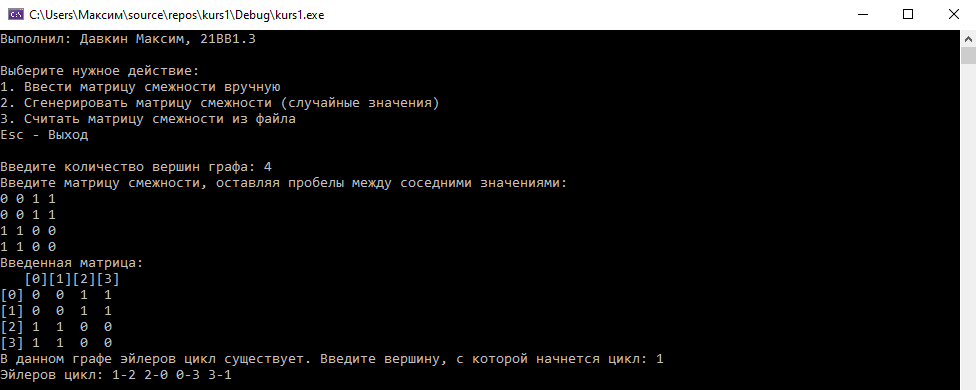


Рисунок 5 – Тестирование при ручном вводе данных

Рисунок 7 – Тестирование при считывании данных из файла

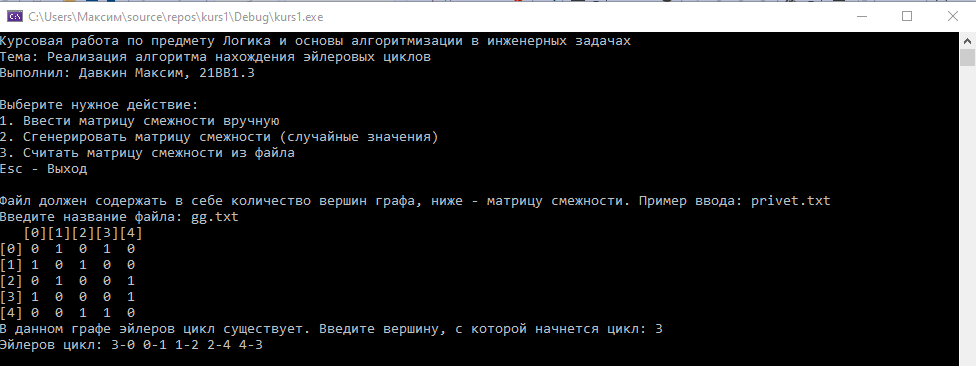


Рисунок 6 – Тестирование при случайной генерации данных

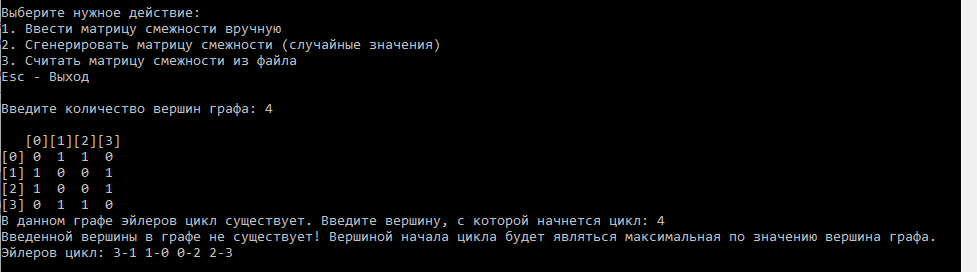
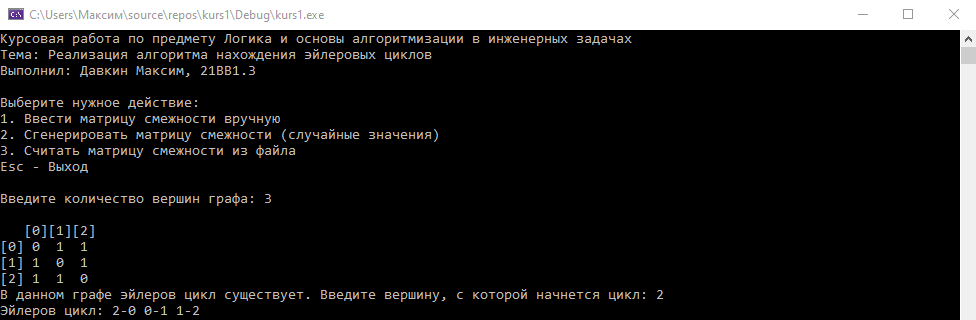


Рисунок 8 – Тестирование в графе с Эйлеровым циклом (4 вершины)

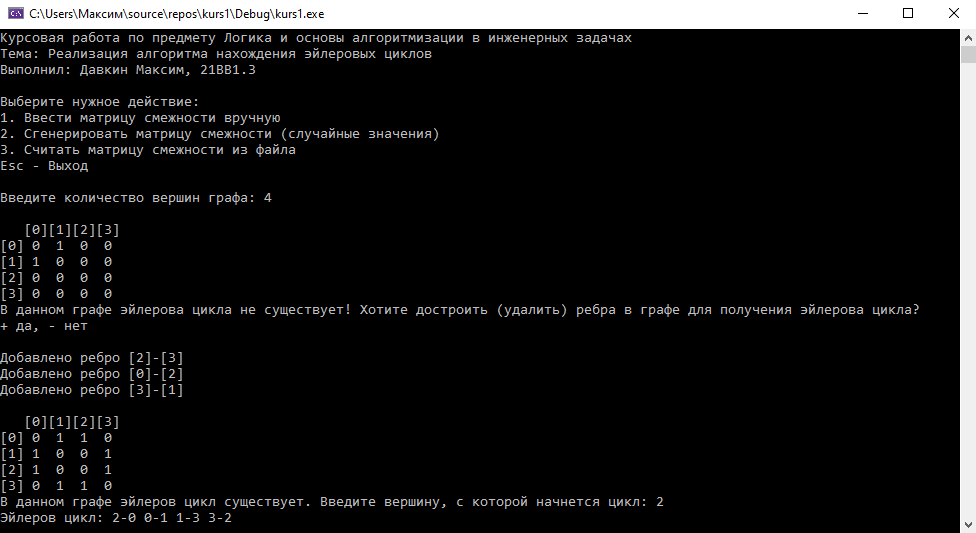


Рисунок 9 – Тестирование в графе без Эйлерова цикла (4 вершины)

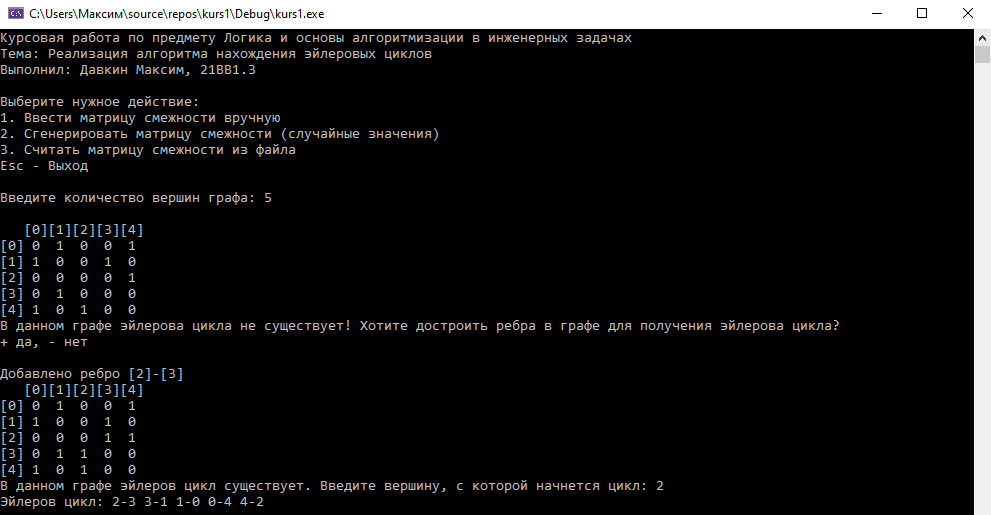


Рисунок 10 – Тестирование в графе без Эйлерова цикла (5 вершин)

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о выборе: ввести с консоли, ввести из файла или сгенерировать случайную матрицу | Верно |
| Выбор ввода с консоли | Вывод сообщения о количестве вершин графа и ввод матрицы | Верно |
| Выбор ввода из файла | Вывод сообщения о названии файла, ввод имени файла, вывод матрицы | Верно |
| Выбор случайной матрицы | Вывод сообщения о количестве вершин графа, вывод матрицы | Верно |
| Нахождение Эйлерова цикла в матрице с четырьмя вершинами, цикл в первоначальной матрице существует | Вывод сообщения о выполнении условия Эйлерова цикла, вывод сообщения о вершине начала цикла, вывод Эйлерова цикла | Верно |

Таблица 1 – Описание поведения программы при тестировании

Продолжение таблицы 1.

|  |  |  |
| --- | --- | --- |
| Нахождение Эйлерова цикла в матрице с четырьмя вершинами, цикла в первоначальной матрице не существует | Вывод сообщения о невыполнении условия существования Эйлерова цикла, вывод сообщения с предложением добавить ребра в матрицу, вывод преобразованной матрицы, вывод сообщения о вершине начала цикла, вывод Эйлерова цикла | Верно |
| Нахождение Эйлерова цикла в матрице с пятью вершинами | Вывод сообщения о невыполнении условия Эйлерова цикла, вывод сообщения с предложением добавить ребра в матрицу, вывод преобразованной матрицы, вывод сообщения о вершине начала цикла, вывод Эйлерова цикла | Верно |

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

# **Ручной расчет задачи**

Проведем проверку работы программы посредством ручных вычислений на примере графа с 5 вершинами (рис. 7). Приведем матрицу смежности.

Таблица 2 – матрица смежности

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |
| **[0]** | 0 | 1 | 0 | 1 | 0 |
| **[1]** | 1 | 0 | 1 | 0 | 0 |
| **[2]** | 0 | 1 | 0 | 0 | 1 |
| **[3]** | 1 | 0 | 0 | 0 | 1 |
| **[4]** | 0 | 0 | 1 | 1 | 0 |

Сначала проверяем условие существования Эйлерова цикла, для этого нужно, чтобы каждая вершина имела четную степень. На рисунке 11 видно, что из каждой вершины выходит четное количество ребер, а это значит, что условие выполняется и можно продолжать последующее вычисление Эйлерова цикла.

Было выбрано начать обход с вершины 3. Проверяется, есть ли путь из 3 в другие вершины. Есть. Т.к. проверка пути происходит с меньшей нумерации вершины, то следующей будет вершина 0, но перед тем, как перейти к ней данное ребро удаляется. [3-0].

Ищем путь из вершины 0. Наименьшая вершина, с которой связана 0 – 1. Ребро удаляется. [0-1]. Из вершины 1 путь в вершину 2. [1-2]. Единственное оставшееся ребро из вершины 2 – 4, оно удаляется, переходим к 4. [2-4]. Из вершины 4 возвращаемся к началу – к вершине 3, цикл замкнут.   
[4-3].

Таким образом, сравнивая полученный результат посредством ручного расчета с вычисленным результатом программы (рис. 7) можно сделать вывод, что программа работает верно.

# **Заключение**

В процессе создания проекта была разработана программа, реализующая алгоритм поиска Эйлерова цикла в графе на базе поиска в глубину.

В ходе выполнения курсовой работы были получены и расширены навыки разработки многомодульных программ на языке C, освоены приемы создания и редактирования матрицы смежности, ее интерпретации. Повторен и изучен расширенно алгоритм поиска в глубину. Закреплены приемы работы в компиляторе Visual Studio 2019.

Недостатком разработанной программы можно считать примитивный интерфейс, однообразное представление графа в виде матрицы смежности (а не в виде самого графа или списка смежности). Программа имеет небольшой, но достаточный для использования функционал возможностей.

# **Список литературы**

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.
2. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
3. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
4. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006

# **Приложение А. Листинг программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <malloc.h>

#include <locale.h>

#include <time.h>

int\*\* vvod\_graph(int\* size);

int\*\* generation\_graph(int\* size);

int\*\* file\_graph(int\* size);

void print\_graph(int\*\* graph, int size);

int check\_cycle(int\*\* graph, int size);

void eulerian\_cycle(int\*\* graph, int size, int cycle\_start);

int main()

{

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

printf("Курсовая работа по предмету Логика и основы алгоритмизации в инженерных задачах\nТема: Реализация алгоритма нахождения эйлеровых циклов\nВыполнил: Давкин Максим, 21ВВ1.3\n\n");

int size = 0, choise1, \*\* graph = NULL;

while (true)

{

printf("Выберите нужное действие:\n1. Ввести матрицу смежности вручную\n2. Сгенерировать матрицу смежности (случайные значения)\n3. Cчитать матрицу смежности из файла\nEsc - Выход\n\n");

choise1 = \_getch();

if (choise1 == '1')

graph = vvod\_graph(&size);

if (choise1 == '2')

graph = generation\_graph(&size);

if (choise1 == '3')

graph = file\_graph(&size);

if (choise1 == 27)

exit(0);

if ((choise1 != '1') && (choise1 != '2') && (choise1 != '3') && (choise1 != 27)) {

printf("Неверный ввод!\n");

continue;

}

print\_graph(graph, size);

check\_cycle(graph, size);

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

{

free(graph[i]);

}

free(graph);

}

}

int\*\* vvod\_graph(int\* size)

{

printf("Введите количество вершин графа: ");

scanf("%d", size);

printf("Введите матрицу смежности, оставляя пробелы между соседними значениями:\n");

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

for (int i = 0; i < (\*size); i++)

{

graph[i] = (int\*)malloc(sizeof(int) \* (\*size));

}

for (int i = 0; i < (\*size); i++)

{

for (int j = 0; j < (\*size); j++)

{

scanf("%d", &graph[i][j]);

}

}

for (int i = 0; i < (\*size); i++)

for (int j = 0; j < (\*size); j++)

if ((graph[i][j] != graph[j][i]) || (graph[i][i] != 0)) {

printf("Неверный ввод матрицы!\n");

exit(0);

}

printf("Введенная матрица:\n");

return graph;

}

int\*\* generation\_graph(int\* size)

{

printf("Введите количество вершин графа: ");

scanf("%d", size);

printf("\n");

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

for (int i = 0; i < (\*size); i++)

{

graph[i] = (int\*)malloc(sizeof(int) \* (\*size));

}

for (int i = 0; i < (\*size); i++)

{

for (int j = i; j < (\*size); j++)

{

if (i == j)

{

graph[i][j] = 0;

}

else {

graph[i][j] = rand() % 2;

graph[j][i] = graph[i][j];

}

}

}

return graph;

}

int\*\* file\_graph(int\* size)

{

char input\_graph[20];

printf("Файл должен содержать в себе количество вершин графа, ниже - матрицу смежности. Пример ввода: privet.txt\nВведите название файла: ");

scanf("%s", &input\_graph);

FILE\* f1 = fopen(input\_graph, "r");

if (f1 == NULL)

{

printf("Не удалось открыть файл!\n");

return NULL;

}

fscanf(f1, "%d", size);

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

for (int i = 0; i < (\*size); i++)

{

graph[i] = (int\*)malloc(sizeof(int) \* (\*size));

for (int j = 0; j < (\*size); j++)

{

fscanf(f1, "%d", &graph[i][j]);

}

}

fclose(f1);

return graph;

}

void print\_graph(int\*\* graph, int size)

{

printf(" ");

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

{

printf("[%d]", i);

}

printf("\n");

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

{

printf("[%d]", i);

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

{

printf(" %d ", graph[i][j]);

}

printf("\n");

}

}

int check\_cycle(int\*\* graph, int size)

{

int flag = 0;

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

{

int count = 0;

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

{

if (graph[i][j] == 1)

count += graph[i][j];

}

if ((count % 2 != 0) || (count == 0))

{

flag = 1;

}

}

if (flag == 1) {

int m = -1, nul = -1, chet = 0;

printf("В данном графе эйлерова цикла не существует! Хотите достроить (удалить) ребра в графе для получения эйлерова цикла?\n+ да, - нет\n\n");

int choise2 = \_getch();

if (choise2 == 45)

return 0;

if (choise2 == 43) {

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

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

{

degree[i] = 0;

}

int o = -1; // сохраняет вершину со степенью size-1

int p = -1; // сохраняет вершину с 0 степенью

int u = -1; // сохраняет вершину с нечетной степенью

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

{

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

{

if (graph[i][j] == 1) {

degree[i]++;

}

}

if ((degree[i] == size - 1) && ((size - 1) % 2 != 0)) {

if (o == -1) {

o = i;

}

else {

graph[o][i] = 0;

graph[i][o] = 0;

printf("Удалено ребро [%d]-[%d]\n", o, i);

degree[o]--;

degree[i]--;

o = -1;

}

}

if (degree[i] == 0) {

if (p == -1) {

p = i;

}

else {

graph[p][i] = 1;

graph[i][p] = 1;

printf("Добавлено ребро [%d]-[%d]\n", p, i);

degree[i]++;

degree[p]++;

p = -1;

}

}

}

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

{

if (o != -1) {

if (graph[o][i] == 1)

{

graph[o][i] = 0;

graph[i][o] = 0;

printf("Удалено ребро [%d]-[%d]\n", o, i);

degree[i]--;

degree[o]--;

o = -1;

}

}

if (p != -1) {

if ((graph[i][p] == 0) && (degree[i] % 2 != 0)) {

graph[p][i] = 1;

graph[i][p] = 1;

printf("Добавлено ребро [%d]-[%d]\n", p, i);

degree[i]++;

degree[p]++;

p = -1;

}

}

if ((degree[i] % 2 != 0) && (o == -1)) {

if (u == -1) {

u = i;

}

else {

if (graph[u][i] == 0)

{

graph[u][i] = 1;

graph[i][u] = 1;

printf("Добавлено ребро [%d]-[%d]\n", u, i);

degree[i]++;

degree[u]++;

u = -1;

}

}

}

}

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

{

if (degree[i] % 2 != 0) {

if (u == -1) {

u = i;

}

else {

if (graph[u][i] == 0)

{

graph[u][i] = 1;

graph[i][u] = 1;

printf("Добавлено ребро [%d]-[%d]\n", u, i);

degree[i]++;

degree[u]++;

u = -1;

}

}

}

if (degree[i] == 0)

{

for (int h = 0; h < size; h++) {

if (degree[h] != 0) {

for (int j = 0; j < size; j++) {

if (graph[h][j] == 1) {

graph[h][j] = 0;

graph[j][h] = 0;

printf("Удалено ребро [%d]-[%d]\n", h, j);

graph[h][i] = 1;

graph[i][h] = 1;

graph[j][i] = 1;

graph[i][j] = 1;

printf("Добавлено ребро [%d]-[%d]\n", h, i);

printf("Добавлено ребро [%d]-[%d]\n", i, j);

degree[i]+=2; break;

}

}

break;

}

}

}

}

printf("\n");

print\_graph(graph, size);

}

if ((choise2 != 45) && (choise2 != 43)) {

printf("Неверный ввод!\n\n");

return 0;

}

}

int cycle\_start = 0;

printf("В данном графе эйлеров цикл существует. Введите вершину, с которой начнется цикл: ");

scanf("%d", &cycle\_start);

if (cycle\_start >= size || cycle\_start < 0)

{

printf("Введенной вершины в графе не существует! Вершиной начала цикла будет являться максимальная по значению вершина графа.\n");

cycle\_start = size - 1;

}

printf("Эйлеров цикл: ");

eulerian\_cycle(graph, size, cycle\_start);

printf("\n\n");

}

void eulerian\_cycle(int\*\* graph, int size, int cycle\_start)

{

for (int cycle\_next = 0; cycle\_next < size; cycle\_next++)

{

if (graph[cycle\_start][cycle\_next] == 1)

{

graph[cycle\_start][cycle\_next] = 0;

graph[cycle\_next][cycle\_start] = 0;

printf("%d-%d ", cycle\_start, cycle\_next);

eulerian\_cycle(graph, size, cycle\_next);

}

}

}