МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4.2

по дисциплине: «Дискретная математика»

по теме: **Циклы**

Выполнил/а: ст. группы ПВ-231

Чупахина София Александровна

Проверил: Рязанов Юрий Дмитриевич

Белгород, 2024

Содержание

Задание 1

Текст задания: Разработать и реализовать алгоритм генерации случайного графа, содержащего n вершин и m ребер.

Поступим как и в прошлой лабораторной работе — будем представлять граф в формате матрицы смежности, а структуру «граф» объявим просто как переименование структуры «бинарное отношение».

Если нам известно количество вершин n и ребер m, мы можем точно определить и количество единиц и нулей в матрице смежности. Всего матрица смежности содержит n^2 элементов, но поскольку мы работаем с неориентированными графами, можно рассматривать только элементы над главной диагональю: ведь элементы вида (x, y) и (y, x) отражают наличие одного и того же ребра между элементами x и y. (Саму главную диагональ мы также не рассматриваем, поскольку элемент не может быть соединен ребром сам с собой, если мы не говорим о псевдографах). Всего в области над главной диагональю $\frac{n^2-n}{2} = \frac{n\times(n-1)}{2}$ элементов. Действительно, каждый из n элементов может быть соединен ребром с одним из $n-1$ элементов, а чтобы исключить одинаковые пары, мы делим произведение $n \times (n-1)$ на 2.

Соответственно, если в функцию передается значение — количество ребер m, то в матрице смежности над главной диагональю будет m единиц и $\frac{n\times(n-1)}{2} - m$ нулей. В теле функции создадим граф с пустой матрицей смежности $n \times n$. для Потом запишем ожидаемое количество единиц в переменную ones_amount, а ожидаемое количество нулей — в переменную zeros_amount. Затем, проходя двумя вложенными циклами по элементам над главной диагональю, будем задавать значение каждому из них, сверяясь со значениями этих переменных. Если zeros_amount равно 0, то значение текущего элемента в матрице смежности может быть равно только 1; если ones_amount равно 0, то это значение может быть равно только 0; если обе переменные имеют ненулевое значение, то значение элемента определяется значением функции rand() (она возвращает псевдослучайное число, от которого в теле функции потом берется остаток от деления на 2). Текущий элемент матрицы приравнивается к выбранному значению (как и симметричный ему относительно главной диагонали), и в зависимости от того, какое значение было выбрано, уменьшается на 1 переменная zeros_amount или ones_amount. Таким образом, за один проход по матрице (вернее, по одной из ее половин) можно сгенерировать случайный граф с заданным количеством вершин и ребер.

Вызвав эту функцию несколько раз для разных значений n и m, можем убедиться, что результаты адекватны.

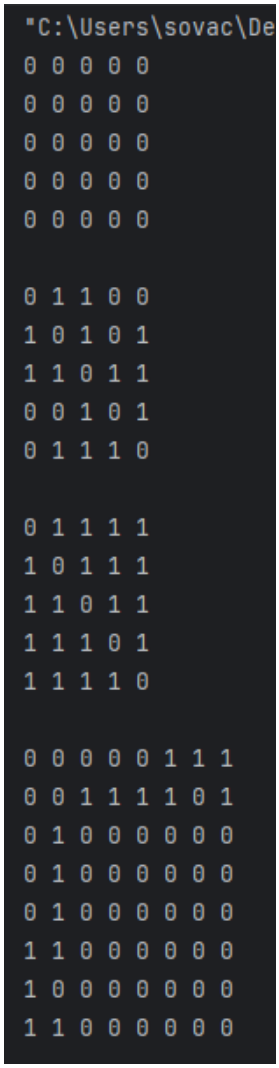
```c
#include "ДИСКРЕТКА../../ ЛАБА 3.1/bin_relations/bin_relation_definition_input_output.c"
#include "ДИСКРЕТКА../../ ЛАБА 3.1/bin_relations/bin_relations_operations.c"
#include "ДИСКРЕТКА../../ ЛАБА 3.1/bin_relations/bin_relations_properties.c"
#include <stdbool.h>
#include <sys/time.h>

typedef bin_relation graph;

```

```
9  graph graph_generate(int n, int m) {
10     graph result = bin_relation_createEmpty(n);
11     int ones_amount = m;
12     int zeros_amount = n*(n-1)/2 - m;
13     for (int i = 1; i <= n; i++) {
14         for (int j = i+1; j<= n; j++) {
15             int value = (ones_amount == 0) ? 0 : ((zeros_amount == 0)
    ? 1 : rand() % 2);
16             bin_relation_changeValue(&result, i, j, value);
17             bin_relation_changeValue(&result, j, i, value);
18             if (value == 1) {
19                 ones_amount--;
20             } else {
21                 zeros_amount--;
22             }
23         }
24     }
25     return result;
26 }
27
28 int main () {
29     graph G1 = graph_generate(5, 0);
30     bin_relation_matrixPrint(G1);
31     printf("\n");
32     graph G2 = graph_generate(5, 7);
33     bin_relation_matrixPrint(G2);
34     printf("\n");
35     graph G3 = graph_generate(5, 10);
36     bin_relation_matrixPrint(G3);
37     printf("\n");
38     graph G4 = graph_generate(8, 8);
39     bin_relation_matrixPrint(G4);
40     printf("\n");
41 }
```

graphs/graphs_generate_final.c



Задание 2

Текст задания: Написать программу, которая:

а) в течение десяти секунд генерирует случайные графы, содержа щие n вершин и m ребер;
б) для каждого полученного графа определяет, является ли он эй леровым или гамильтоновым;
в) подсчитывает общее количество сгенерированных графов и ко личество графов каждого типа.

Результат работы программы представить в виде таблицы (табл. 4).

Величину h подобрать такой, чтобы в таблице количество строк было в диапазоне от 20 до 30.

Для выполнения этой задачи нам понадобятся, в первую очередь, функции, определяющие, является ли переданный им граф гамильтоновым или эйлеровым. Для определения того, является ли граф гамильтоновым, будем методом поиска с возвращением искать в нем хотя бы

один гамильтонов цикл. Мы уже пользовались рекурсивным поиском для нахождения маршрутов, цепей и так далее; частично структура программы для определения «гамильтоновости» графа будет похожа на структуры ранее написанных программ.

Для выполнения этой задачи нам понадобится три функции. Первая — вспомогательная функция, проверяющая, включен ли некоторый элемент в последовательность; она понадобится для проверки сложных условий. Вторая — функция рекурсивного вызова graph_isHamilton_, принимабщая на вход непосредственно граф G, формируемый гамильтонов цикл W, индекс формируемой позиции cur_pos и предварительный результат result. Цепь ее вызовов формирует гамильтонов цикл, а за конкретный вызов задается значение элемента под индексом cur_pos (отсчет начинается с нуля) в последовательности вершин, отображающей этот цикл. Для этого перебираются все возможные вершины графа, и для каждой проверяется, смежна ли она с последней на текущий момент вершиной в последовательности и не была ли включена в последовательность до сих пор. Если это так, то элемент маршрута под индексом cur_pos приравнивается к найденному элементу. Затем проверяется, достигла ли последовательность нужной длины (количество вершин в ней на этом этапе должно быть равно количеству вершин в графе в целом). Если это так, то проверяется еще одно условие: соединены ли ребром текущий последний элемент последовательности и элемент, с которого она началась. При выполнении этого второго условия значение result приравнивается к 1. Если выполнено условие о длине, но не выполнено условие о смежности, то никаких действий не предпринимается. Если же условие о длине не выполнено, значит, в последовательность нужно добавлять новые вершины, чтобы она стала (возможно) гамильтоновым циклом. Тогда рекурсивная функция запускается для тех же данных, но с увеличенным на один индексом cur_pos. После прохода по всем вершинам для данного индекса cur_pos (и после всех более глубоких рекурсивных вызовов) функция возвращает значение result. И третья функция, graph_isHamilton — обертка для рекурсивной функции. Она создает последовательность, которая будет формироваться цепочкой рекурсивных вызовов, задает ее первый элемент (это всегда элемент 1: раз гамильтонов цикл включает в себя все вершины графа, то и вершина 1 будет включена в него, а поскольку это цикл, нет разницы, с какой вершины его начинать) и запускает рекурсивную функцию с cur_pos равным 1 (нулевой элемент уже сформирован) и предварительным результатом 0.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
bool isInSequence(int *W, int n, int value) {
    for (int j = 0; j < n; j++) {
        if (W[j] == value) {
            return true;
        }
    }
    return false;
}

bool graph_isHamilton_(graph G, int *W, int cur_pos, bool result) {
    for (int cur_el = 1; cur_el <= G.max_value && !result; cur_el++) {
        if (bin_relation_getValue(G, W[cur_pos-1], cur_el) &&
   !isInSequence(W, cur_pos, cur_el)) {
            W[cur_pos] = cur_el;
            if (cur_pos + 1 == G.max_value) {
                if (bin_relation_getValue(G, cur_el, W[0])) {
```

```
                    result = 1;
                }
            } else {
                result = graph_isHamilton_(G, W, cur_pos + 1, result);
            }
        }
    }
    return result;
}

bool graph_isHamilton(graph G) {
    int W[G.max_value];
    W[0] = 1;
    return graph_isHamilton_(G, W, 1, 0);
}
```

graphs/graphs_generate_final.c

Однако для определения того, является ли граф эйлеровым, метод поиска с возвращением не пойдет. Ведь в нем длина рассматриваемых последовательностей зависит от количества ребер, а не вершин, а сами вершины в последовательности могут повторяться. Получается, даже на графе с небольшим количеством вершин при возрастании количества ребер будет увеличиваться длина последовательности, а количество вариантов для рассмотрения будет куда больше, чем для поиска гамильтоновых циклов. Поэтому метод поиска с возвращением применим только для графов с небольшим количеством ребер, и рациональнее применять его тогда, когда нужно получить все эйлеровы циклы. Для проверки же того, является ли граф эйлеровым в принципе, воспользуемся соответствующей теоремой: связный граф является эйлеровым тогда и только тогда, когда все его вершины имеют четную степень.

Для того, чтобы определить степень вершины, достаточно в цикле найти сумму всех элементов соответствующего ей ряда матрицы смежности; степень будет четной, если эта сумма делится нацело на два. Циклом пройдясь по всем вершинам, можно определить, имеют ли они все четную степень. Но как определить связность графа? Если граф связан, значит, из каждой его вершины существует путь в любую другую вершину, неважно какой длины этот путь. Получить матрицу, отображающую, есть ли между вершинами путь заданной блины, можно, возведя матрицу смежности графа в степень, равную этой длине. Но матрица, отображающую, есть ли между вершинами путь в целом — транзитивное замыкание бинарного отношения, которое задается матрицей смежности графа. Скопируем функцию, возвращающую транзитивное замыкание, из лабораторной работы 3.2. В теле функции graph_isEuler найдем отношение транзитивного замыкания для матрицы смежности данного графа, а потом будет с помощью цикла проходить по рядам обеих матриц и искать сумму ряда, определяя, таким образом: 1) четна ли степень текущей вершины, 2) все ли элементы отношения транзитивного замыкания равны единице, то есть связан ли каждый элемент исходного графа с каждым. Если для какого-то из рядов хотя бы одно из условий нарушено, значит, граф не является эйлеровым.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
bin_relation transitive_closure(bin_relation A) {
    bin_relation C =A;
    bin_relation C2 = bin_relation_composition(C,C);
```

```
    while(!bin_relation_inclusion(C2,C)) {
        C = bin_relation_union(C,C2);
        C2 = bin_relation_composition(C,C);
    }
    return C;
}

bool graph_isEuler(graph G) {
    bin_relation transitive = transitive_closure(G);
    for (int i = 1; i <= G.max_value; i++) {
        int sum_of_transitive_row = 0;
        int sum_of_original_row = 0;
        for (int j = 1; j <= G.max_value; j++) {
            sum_of_transitive_row += bin_relation_getValue(transitive,
   i, j);
            sum_of_original_row += bin_relation_getValue(G, i, j);
        }
        if (sum_of_original_row % 2 != 0 || sum_of_transitive_row !=
   G.max_value) {
            return 0;
        }
    }
    return 1;
}
```

graphs/graphs_generate_final.c

Теперь, когда имеются функции, определяющие нужные свойства графа, займемся непосредственно генерацией. Сначала создадим функцию для генерации однородных (с одинаковым количеством и вершин, и ребер) графов и определения их свойств в течение десяти секунд. Перед первой генерацией в теле функции задаются временные метки start и end — экземпляры структуры timeval из библиотеки <sys.time>. Затем, пока разница между ними не станет больше десяти секунд (эта разница находится с помощью вспомогательной функции), происходит генерация графа с заданным количеством вершин и ребер и определяются его свойства. Если граф эйлеров или гамильтонов, увеличиваются на один значения счетчиков по переданным адресам; вне зависимости от свойств, увеличивается на 1 и счетчик сгенерированных графов в целом. После генерации и определения свойств временная метка end обновляется.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
typedef struct timeval Timeval;

Timeval getDifference(Timeval start, Timeval end) {
    Timeval difference;
    difference.tv_sec = end.tv_sec - start.tv_sec - (end.tv_usec <
   start.tv_usec);
    difference.tv_usec = (end.tv_usec + (end.tv_usec < start.tv_usec)
   * 1000000)-start.tv_usec;
    return difference;
```

```
}

void graph_generateBunch(int n, int m, int *hamilton_counter, int
   *euler_counter, int *all_counter) {
    Timeval start, end;
    gettimeofday(&start, NULL);
    gettimeofday(&end, NULL);
    while (getDifference(start, end).tv_sec < 10) {
        graph cur_graph = graph_generate(n, m);
        if (graph_isEuler(cur_graph)) {
            (*euler_counter)++;
        }
        if (graph_isHamilton(cur_graph)) {
            (*hamilton_counter)++;
        }
        (*all_counter)++;
        gettimeofday(&end, NULL);
    }
}
```

graphs/graphs_generate_final.c

Наконец, последняя функция будет определять шаг, с которым, при заданном количестве вершин графа, будет меняться количество его ребер, и с заданным шагом запускать функцию генерации однородных массивов и выводить на экран значения счетчиков для данного количества вершин и ребер.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
void makeTable(int n) {
    int start_m = n;
    int end_m = n*(n-1)/2;
    int diff_m = end_m - start_m;
    int h = 1;
    while (diff_m / h > 30) {
        h+=1;
    }
    if (diff_m / h < 20) {
        h-=1;
    }
    while (start_m <= end_m) {
        int hamilton = 0;
        int euler = 0;
        int all = 0;
        graph_generateBunch(n, start_m, &hamilton, &euler, &all);
        printf("%d %d %d %d %d\n", n, start_m, euler, hamilton, all);
        start_m += h;
    }
```

```
}
```

graphs/graphs_generate_final.c

Остается только ввести с клавиатуры значение n и запустить функцию makeTable в теле main.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
int main () {
    int vertex_amount;
    scanf("%d", &vertex_amount);
    makeTable(vertex_amount);
}
```

graphs/graphs_generate_final.c

Задание 3

Текст задания: Выполнить программу при $n = 8, 9, 10$ и сделать выводы.

Запустим программу трижды, вводя с клавиатуры значения 8, 9, 10, и перенесем полученные данные в таблицы ниже.

Количество вершин	**Количество ребер**	**Количество графов**		
		эйлеровых	**гамильтоновых**	**всех**
8	8	329	329	2706164
8	9	296	2961	2170248
8	10	5758	18781	1664216
8	11	8234	70605	1331577
8	12	7380	162288	994541
8	13	5763	258642	845660
8	14	5488	375519	753836
8	15	5333	474302	725616
8	16	5074	538992	716389
8	17	3936	604536	732807
8	18	5724	577744	679661
8	19	4064	611978	692214
8	20	4922	710585	776501
8	21	12396	794662	853735
8	22	11960	704613	750801
8	23	5070	3590176	3590176
8	24	274	4484313	4484313
8	25	0	4809515	4809515
8	26	0	5497014	5497014
8	27	0	5546394	5546394
8	28	0	6016808	6016808

Количество вершин	**Количество ребер**	**Количество графов**		
		эйлеровых	**гамильтоновых**	**всех**
9	9	0	0	2250347
9	10	569	0	1866770
9	11	494	2226	1475801
9	12	2212	8453	1103489
9	13	1141	26403	828961
9	14	2282	56498	603688
9	15	1465	100133	471878
9	16	1590	144860	383106
9	17	964	181004	339669
9	18	1336	208463	311177
9	19	781	228743	300877
9	20	1129	254534	306271
9	21	1141	264422	303443
9	22	1188	240478	270309
9	23	624	221317	244116
9	24	1265	213434	231075
9	25	680	188389	202509
9	26	505	194239	205769
9	27	928	240918	251222
9	28	757	249320	258594
9	29	709	191054	197922
9	30	1299	2246412	2246412
9	31	1718	3413076	3413076
9	32	17195	3860149	3860149
9	33	17632	4410086	4410086
9	34	0	4357571	4357571
9	35	0	4518128	4518128
9	36	3890206	3890206	3890206

Количество вершин	**Количество ребер**	**Количество графов**		
		эйлеровых	**гамильтоновых**	**всех**
10	10	0	0	1822419
10	11	0	0	1556894
10	12	0	2	1256609
10	13	0	947	955767
10	14	625	3598	732693
10	15	480	8138	523783
10	16	182	18236	375074
10	17	351	33016	272801

10	18	220	47578	202974
10	19	461	58483	163571
10	20	508	78002	145503
10	21	238	90230	133135
10	22	114	96745	126689
10	23	111	101363	121978
10	24	252	105400	120304
10	25	239	91157	102380
10	26	313	92284	100954
10	27	94	79453	86057
10	28	103	74341	79372
10	29	154	66605	70394
10	30	121	62977	65982
10	31	123	54785	57256
10	32	129	52656	54679
10	33	96	45336	46862
10	34	87	49618	50960
10	35	100	55529	56756
10	36	255	55409	56483
10	37	228	43807	44639
10	38	354	1097930	1097930
10	39	0	2287160	2287160
10	40	0	3026333	3026333
10	41	0	3456340	3456340
10	42	0	3603233	3603233
10	43	0	3978845	3978845
10	44	0	3880880	3880880
10	45	0	4122492	4122492

Анализируя данные в таблицах, можем выделить следующие закономерности:

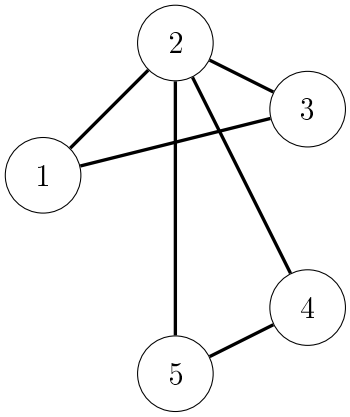
- В большинстве случаев, среди множества случайных графов с одинаковым количеством ребер и вершин эйлеровых графов в несколько десятков раз меньше, чем гамильтоновых,
- Нельзя сказать, что количества эйлеровых и гамильтоновых графов изменяются по общему закону; можно найти строки, где количество гамильтоновых графов растет, а эйлеровых — падает, и наоборот,
- Среди графов с равным количеством вершин и ребер существуют и эйлеровы, и гамильтоновы, причем множества эйлеровых и гамильтоновых графов совпадают (это видно на примере, где $n = m = 8$), однако с увеличением числа вершин шансы того, что такие графы будут сгенерированы, сильно падают, потому количество таких графов для $n = 9$ и $n = 10$ в таблице равно 0.
- Для ряда случаев, когда количество ребер близится к максимальному, все составленные графы будут гамильтоновыми. В целом с увеличением числа ребер процент гамильтоновых графов среди всех скорее возрастает, тогда как процент эйлеровых графов ведет себя более непредсказуемо.
- Для ряда случаев, когда количество ребер близится к максимальному, невозможно составить эйлеров граф; размер этой зоны зависит от того, четно или нечетно количество

вершин (для нечетного количества она меньше). От этого же зависит соотношение количества сгенерированных эйлеровых и гамильтоновых графов при максимальном количестве ребер (для полного графа). По сути, когда мы генерируем графы с максимальным количеством ребер, мы все время будем генерировать один и тот же граф; он всегда будет гамильтоновым, поэтому количество сгенерированных в целом и гамильтоновых графов совпадает. Если число вершин нечетно, то полный граф будет эйлеровым, тогда и их количество будет равно общему; если числ овершин четно, то полный граф не является эйлеровым, и число сгененрированных эйлеровых графов будет равно 0.

Задание 4

Текст задания: Привести пример диаграммы графа, который является эйлеровым, но не гамильтоновым. Найти в нем все эйлеровы циклы.

Зададим следующий граф из пяти вершин и шести ребер.



Вручную пройдя по всем ребрам, мы не сможем найти для него ни одного гамильтонова цикла, но сможем найти как минимум один эйлеров цикл: $[1, 3, 2, 4, 5, 2, 1]$. Но сможем ли мы найти все эйлеровы циклы самостоятельно, а потом доказать, что кроме найденных циклов, других не существует? А сможем ли доказать отсутствие гамильтоновых циклов для этого графа? Чтобы не заниматься этим, напишем программы, выводящие на экран (при их наличии) все эйлеровы и гамильтоновы циклы. Будем использовать для этого уже знакомый нам метод рекурсивного поиска.

Функция graph_allHamiltonCycles_, по сути, очень похожа на функцию graph_isHamilton_, однако в ней отсутствует аргумент result: вместо того, чтобы приравнивать его к 1, когда подходящая условиям гамильтонова цикла последовательность найдена, последовательность просто выводится на экран с помощью функции printSequence. Обертка этой функции, graph_allHamiltonCycles, перебирает все возможные элементы, каждый из них назначает на роль первого в созданной последовательности W и для каждого из этих случаев вызывает функцию graph_isHamilton_. Так мы получаем все гамильтоновы циклы, а не только начинающиеся с определенной вершины.

```
1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 void printSequence(int *W, int n) {
3     printf("(");
```

```
    for (int cur_ind = 0; cur_ind < n; cur_ind++) {
        printf("%d, ", W[cur_ind]);
    }
    printf("\b\b)\t");
}

void graph_allHamiltonCycles_(graph G, int *W, int cur_pos) {
    for (int cur_el = 1; cur_el <= G.max_value; cur_el++) {
        if (bin_relation_getValue(G, W[cur_pos-1], cur_el) &&
   !isInSequence(W, cur_pos, cur_el)) {
            W[cur_pos] = cur_el;
            if (cur_pos + 1 == G.max_value) {
                if (bin_relation_getValue(G, cur_el, W[0])) {
                    W[cur_pos+1] = W[0];
                    printSequence(W, cur_pos + 2);
                }
            } else {
                graph_allHamiltonCycles_(G, W, cur_pos + 1);
            }
        }
    }
}

void graph_allHamiltonCycles(graph G) {
    int W[G.max_value + 1];
    for (int cur_start_el = 1; cur_start_el <= G.max_value;
   cur_start_el++) {
        W[0] = cur_start_el;
        graph_allHamiltonCycles_(G, W, 1);
    }
}
```

graphs/graphs_cycles_searching.c

К рекурсивному поиску эйлеровых циклов будет применяться похожая логика, но при формировании позиции будет проверяться не то, входила ли уже в последовательность такая вершина, а то, входило ли в множество пройденных ребер ребро, образованное рассматриваемой вершиной и предыдущей. После того, как текущая вершина включается в последовательность, в множество пройденных ребер, представленное бинарным отношением, включаются пары, составленные из текущего и предыдущего элемента. Достигла ли последовательность нужной длины, определяется также не количеством вершин графа, а количеством ребер в исходном графе (оно считается с помощью вспомогательной функции). И точно также в качестве первого элемента последовательности в теле функции-обертки поочередно выбираются все вершины графа.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
int count_edges(graph G) {
    int result = 0;
```

```
    for (int i = 1; i <= G.max_value; i++) {
        for (int j = i+1; j<= G.max_value; j++) {
            result += bin_relation_getValue(G, i, j);
        }
    }
    return result;
}

void graph_allEulerCycles_(graph G, int *W, bin_relation E, int cur_pos) {
    for (int cur_el = 1; cur_el <= G.max_value; cur_el++) {
        if (bin_relation_getValue(G, W[cur_pos-1], cur_el) && !bin_relation_getValue(E, cur_el, W[cur_pos-1])) {
            bin_relation_changeValue(&E, cur_el, W[cur_pos-1], 1);
            bin_relation_changeValue(&E, W[cur_pos-1], cur_el, 1);
            W[cur_pos] = cur_el;
            if (cur_pos + 1 == count_edges(G)) {
                if (bin_relation_getValue(G, cur_el, W[0]) && !bin_relation_getValue(E, cur_el, W[0])) {
                    W[cur_pos+1] = W[0];
                    printSequence(W, cur_pos + 2);
                }
            } else {
                graph_allEulerCycles_(G, W, E, cur_pos + 1);
            }
            bin_relation_changeValue(&E, cur_el, W[cur_pos-1], 0);
            bin_relation_changeValue(&E, W[cur_pos-1], cur_el, 0);
        }
    }
}

void graph_allEulerCycles(graph G) {
    bin_relation E = bin_relation_createEmpty(G.max_value);
    int W[count_edges(G) + 1];
    for (int cur_start_el = 1; cur_start_el <= G.max_value; cur_start_el++) {
        W[0] = cur_start_el;
        graph_allEulerCycles_(G, W, E, 1);
    }
}
```

graphs/graphs_cycles_searching.c

Если мы зададим матрицу смежности этого графа и применим к нему функции graph_allHamiltonCycles и graph_allEulerCycles, вывод подтвердит наши догадки: для данного графа не найдено ни одного гамильтонова цикла, но найдено множество эйлеровых.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
```

```
graph graph_createFromMatrix(matrix M) {
    graph result = bin_relation_createEmpty(M.nRows);
    for (int i = 1; i <= M.nRows; i++) {
        for (int j = 1; j <= M.nRows; j++) {
            bin_relation_changeValue(&result, i, j, M.values[i-1][j - 1]);
        }
    }
    return result;
}

int main () {
    matrix M1 = createMatrixFromArray((int[25]) {
            0, 1, 1, 0, 0,
            1, 0, 1, 1, 1,
            1, 1, 0, 0, 0,
            0, 1, 0, 0, 1,
            0, 1, 0, 1,  0
    }, 5, 5);
    graph G1 = graph_createFromMatrix(M1);
    printf("\n\nAll Hamilton cycles for G1:\n");
    graph_allHamiltonCycles(G1);
    printf("\nAll Euler cycles for G1:\n");
    graph_allEulerCycles(G1);

    matrix M2 = createMatrixFromArray((int[25]) {
            0, 0, 1, 0, 1,
            0, 0, 0, 1, 1,
            1, 0, 0, 1, 0,
            0, 1, 1, 0, 1,
            1, 1, 0, 1,  0
    }, 5, 5);
    graph G2 = graph_createFromMatrix(M2);
    printf("\n\nAll Hamilton cycles for G2:\n");
}
```

graphs/graphs_cycles_searching.c

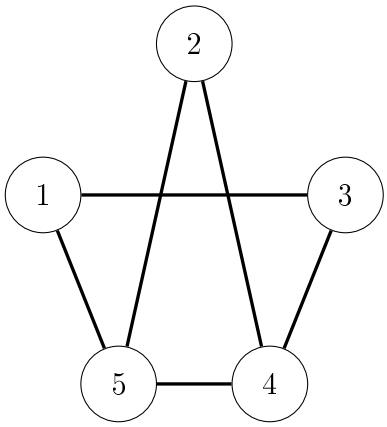
```
All Hamilton cycles for G1:

All Euler cycles for G1:
(1, 2, 4, 5, 2, 3, 1)   (1, 2, 5, 4, 2, 3, 1)   (1, 3, 2, 4, 5, 2, 1)   (1, 3, 2, 5, 4, 2, 1)   (2, 1, 3, 2, 4, 5, 2)
(2, 1, 3, 2, 5, 4, 2)   (2, 3, 1, 2, 4, 5, 2)   (2, 3, 1, 2, 5, 4, 2)   (2, 4, 5, 2, 1, 3, 2)   (2, 4, 5, 2, 3, 1, 2)
(2, 5, 4, 2, 1, 3, 2)   (2, 5, 4, 2, 3, 1, 2)   (3, 1, 2, 4, 5, 2, 3)   (3, 1, 2, 5, 4, 2, 3)   (3, 2, 4, 5, 2, 1, 3)
(3, 2, 5, 4, 2, 1, 3)   (4, 2, 1, 3, 2, 5, 4)   (4, 2, 3, 1, 2, 5, 4)   (4, 5, 2, 1, 3, 2, 4)   (4, 5, 2, 3, 1, 2, 4)
(5, 2, 1, 3, 2, 4, 5)   (5, 2, 3, 1, 2, 4, 5)   (5, 4, 2, 1, 3, 2, 5)   (5, 4, 2, 3, 1, 2, 5)
```

Задание 5

Текст задания: Привести пример диаграммы графа, который является гамильтоновым, но не эйлеровым. Найти в нем все гамильтоновы циклы.

Для того, чтобы задать граф, являющийся гамильтоновым, но не эйлеровым, можно задать любую перестановку его вершин и соединить стоящие рядом, а также первую и последнюю вершины, обеспечив наличие в графе хотя бы одного гамильтонова цикла, а потом добавить хотя бы одно ребро, нарушив, таким образом, условие о четности степеней вершин. В графе из пяти вершин зададим перестановку $[1, 3, 4, 2, 5]$, а потом дополнительно проведем ребро между вершинами 4 и 5.



Если мы зададим матрицу смежности этого графа и применим к нему функции graph_allHamiltonCycles и graph_allEulerCycles, вывод подтвердит наши догадки: для данного графа не найдено ни одного эйлерова цикла, но найдено несколько гамильтоновых.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА

int main () {
    matrix M2 = createMatrixFromArray((int[25]) {
            0, 0, 1, 0, 1,
            0, 0, 0, 1, 1,
            1, 0, 0, 1, 0,
            0, 1, 1, 0, 1,
            1, 1, 0, 1,  0
    }, 5, 5);
    graph G2 = graph_createFromMatrix(M2);
    printf("\n\nAll Hamilton cycles for G2:\n");
    graph_allHamiltonCycles(G2);
    printf("\nAll Euler cycles for G2:\n");
    graph_allEulerCycles(G2);
}
```

graphs/graphs_cycles_searching.c

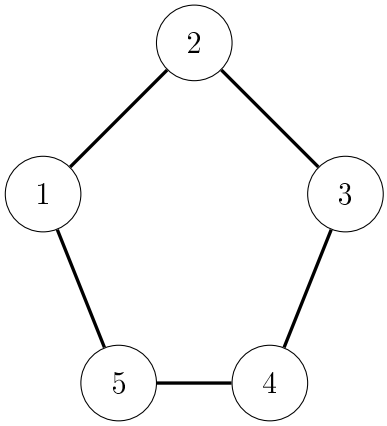
```
All Hamilton cycles for G2:
(1, 3, 4, 2, 5, 1)      (1, 5, 2, 4, 3, 1)      (2, 4, 3, 1, 5, 2)      (2, 5, 1, 3, 4, 2)      (3, 1, 5, 2, 4, 3)
(3, 4, 2, 5, 1, 3)      (4, 2, 5, 1, 3, 4)      (4, 3, 1, 5, 2, 4)      (5, 1, 3, 4, 2, 5)      (5, 2, 4, 3, 1, 5)

All Euler cycles for G2:
```

Задание 6

Текст задания: Привести пример диаграммы графа, который является эйлеровым и гамильтоновым. Найти в нем все эйлеровы и гамильтоновы циклы.

Для того, чтобы задать граф, являющийся и гамильтоновым, и эйлеровым, можно задать любую перестановку его вершин и соединить стоящие рядом, а также первую и последнюю вершины, обеспечив наличие в графе хотя бы одного гамильтонова цикла. Если не добавлять в граф ни одного ребра после этого, то полученный гамильтонов цикл будет проходить по всем ребрам графа, а значит, будет являться также и эйлеровым. В графе из пяти вершин зададим перестановку $[1, 2, 3, 4, 5]$ и соединим соответствующие вершины.



Если мы зададим матрицу смежности этого графа и применим к нему функции graph_allHamiltonCycles и graph_allEulerCycles, вывод подтвердит наши догадки: для данного графа существуют как эйлеровы, так и гамильтоновы циклы.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА

int main () {
    matrix M3 = createMatrixFromArray((int[25]) {
            0, 1, 0, 0, 1,
            1, 0, 1, 0, 0,
            0, 1, 0, 1, 0,
            0, 0, 1, 0, 1,
            1, 0, 0, 1, 0
    }, 5, 5);
    graph G3 = graph_createFromMatrix(M3);
    printf("\n\nAll Hamilton cycles for G3:\n");
    graph_allHamiltonCycles(G3);
```

```
    printf("\nAll Euler cycles for G3:\n");
    graph_allEulerCycles(G3);
}
```

graphs/graphs_cycles_searching.c

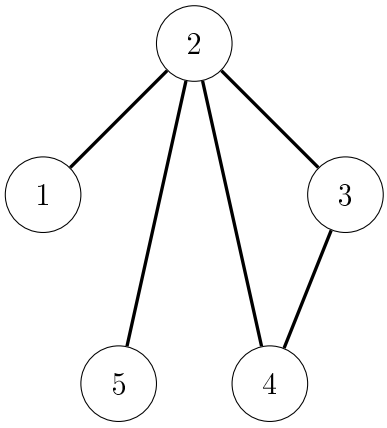
```
All Hamilton cycles for G3:
(1, 2, 3, 4, 5, 1)      (1, 5, 4, 3, 2, 1)      (2, 1, 5, 4, 3, 2)      (2, 3, 4, 5, 1, 2)      (3, 2, 1, 5, 4, 3)
(3, 4, 5, 1, 2, 3)      (4, 3, 2, 1, 5, 4)      (4, 5, 1, 2, 3, 4)      (5, 1, 2, 3, 4, 5)      (5, 4, 3, 2, 1, 5)

All Euler cycles for G3:
(1, 2, 3, 4, 5, 1)      (1, 5, 4, 3, 2, 1)      (2, 1, 5, 4, 3, 2)      (2, 3, 4, 5, 1, 2)      (3, 2, 1, 5, 4, 3)
(3, 4, 5, 1, 2, 3)      (4, 3, 2, 1, 5, 4)      (4, 5, 1, 2, 3, 4)      (5, 1, 2, 3, 4, 5)      (5, 4, 3, 2, 1, 5)
```

Задание 7

Текст задания: Привести пример диаграммы графа, который не является ни эйлеровым, ни гамильтоновым.

Для того, чтобы задать граф, не являющийся ни гамильтоновым, ни эйлеровым, можно расположить ребра между вершинами графа таким образом, чтобы существовала вершина, смежная только с одной вершиной. Тогда при формировании маршрута, при прохождении через эту вершину, продвинуться дальше не будет возможности, не пройдя еще раз по тому единственному ребру, которое вело к единственной смежной с ней вершине. Значит, будет нарушено условие и эйлерова, и гамильтонова цикла. В графе из пяти вершин проведем ребра так, чтобы вершины 1 и 5 имели только одну смежную вершину.



Если мы зададим матрицу смежности этого графа и применим к нему функции graph_allHamiltonCycles и graph_allEulerCycles, вывод подтвердит наши догадки: для данного графа не найдено ни одного эйлерова, равно как и ни одного гамильтонова цикла.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА

int main () {
    matrix M4 = createMatrixFromArray((int[25]) {
            0, 1, 0, 0, 0,
            1, 0, 1, 1, 1,
```

```
7               0, 1, 0, 1, 0,
8               0, 1, 1, 0, 0,
9               0, 1, 0, 0, 0
10      }, 5, 5);
11      graph G4 = graph_createFromMatrix(M4);
12      printf("\n\nAll Hamilton cycles for G4:\n");
13      graph_allHamiltonCycles(G4);
14      printf("\nAll Euler cycles for G4:\n");
15      graph_allEulerCycles(G4);
16  }
```

graphs/graphs_cycles_searching.c

Вывод

Среди маршрутов — последовательностей инцидентных ребер и вершин в графе, которые чаще расматриваются как последовательности смежных вершин — можно выделить отдельный класс: циклы. Дополнительные свойства маршрутов позволяют выделять среди них цепи, простые цепи, циклы и простые циклы. У циклов можно выделять разные свойства: например, свойство простоты — цикл называется простым, если ни одна из вершин в нем, кроме первой и последней, не повторяется. В данной же работе обосый интерес для нас представляли понятия гамильтонова и эйлерова цикла. Гамильтоновым циклом называется простой цикл, куда входят все вершины графа. Эйлеровым циклом называется цикл, в котором могут повторяться вершины, но в который входят без повторений все ребра графа. Если в графе существует хотя бы один гамильтонов или эйлеров цикл, он называется соотвественно гамильтоновым или эйлеровым.

В ходе лабораторной работы написали ряд функций, которые как определяют, является ли граф гамильтоновым или эйлеровым, так и находят в нем все гамильтоновы и эйлеровы циклы; путем генерации большого числа графов определили, как много гамильтоновых и эйлеровых графов содержится среди графов с тем или иным количеством ребер и вершин; привели примеры графов, которые обладают только одним из этих свойств, двумя свойствами сразу или ни одним из них; применили эти функции, чтобы найти все гамильтоновы и эйлеровы циклы в составленных графах.