Текст задания: Написать функцию линейной временной сложности, определяющую, является ли заданная последовательность вершин цепью в графе.

Решение: Линейной сложности можно добиться, если сменить способ представления множества ребер. Пусть в данной реализации за отслеживание того, входило ли некое ребро в заданную последовательность, отвечает не последовательность вершин, рассматриваемых парами, а бинарное отношение на множестве вершин графа. Если в нем элемент (x, y) равен 1, значит, ребро, соединяющее элементы x и y, уже входило в последовательность. После включения в последовательность, где ранее последней вершиной была вершина a, очередной вершины b, необходимо установить в 1 как элемент (a, b), так и элемент (b, a). Доступ к элементам бинарного отношения является прямым, поэтому мы сможем проверять, входило ли данное ребро в последовательность, не задействуя дополнительный вложенный цикл.

```
bool isSequenceChain(graph G, int *s, int n) {
    bool result = s[0] <= G.max_value;
    graph passed_edges = bin_relation_createEmpty(G.max_value);
    for (int vert_ind = 1; vert_ind < n && result; vert_ind++) {
        bool cond1 = s[vert_ind] > G.max_value;
        bool cond2 = !bin_relation_getValue(G, s[vert_ind],
s[vert_ind-1]);
        bool cond3 = bin_relation_getValue(passed_edges,
s[vert_ind], s[vert_ind-1]);
        if (cond1 || cond2 || cond3) {
            result = false;
        } else {
            bin_relation_changeValue(&passed_edges, s[vert_ind],
s[vert_ind-1], 1);
            bin_relation_changeValue(&passed_edges, s[vert_ind-
1], s[vert_ind], 1);
        }
    }
    return result;
}
```

Данная версия функции isSequenceChain включена в исправленную версию лабораторной работы.

Текст задания: Задан граф, в котором n вершин и любые две не равные вершины образуют ребро. Сколько существует маршрутов длины L в этом графе, которые начинаются заданной вершиной?

Решение: Если любые две не равные вершины в графе образуют ребро, такой граф называется полным. В полном графе, какую бы начальную вершину мы не выбрали, ей будет смежна любая из *n-1* вершин, не равных начальной. Если мы начнем формировать маршрут, первой вершиной которого будет заданная вершина, а второй – одна из *n-1* вершин, то для выбранной вершины смежными также будут являться *n-1* вершин (в их множество также будет включена и начальная). И для каждой следующей включенной в маршрут вершины можно будет выбрать следующую вершину как одну из *n-1*. Таким образом, для полного графа количество маршрутов длины *L* из заданной вершины будет равно $(n-1)^L$.

Данное утверждение можно проверить с помощью программы для задания №5. Программа формирует матрицу количества маршрутов заданной длины из одной вершины в другую, но сложив значения по определенной строке или столбцу, мы сможем узнать, сколько маршрутов заданной длины, начинающихся с заданной вершины, содержится в графе в целом. Создадим для решения этого задания отдельный файл, в нем для полного графа с количеством вершин $n = 7$ сформируем такие матрицы для маршрутов длиной 2 и 4.

```
#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 "../matrix/matrix.c"

#include <stdbool.h>

typedef bin_relation graph;

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;
}

void getMatrixOfRoutesAmount(graph G, int len, matrix *M) {
    matrix result = getMemMatrix(G.max_value, G.max_value);
```

```
    for (int row_ind = 0; row_ind < G.max_value; row_ind++) {
        for (int col_ind = 0; col_ind < G.max_value; col_ind++) {
            result.values[row_ind][col_ind] =
bin_relation_getValue(G, row_ind+1, col_ind+1);
            M->values[row_ind][col_ind] =
bin_relation_getValue(G, row_ind+1, col_ind+1);
        }
    }
    for (int degree = 1; degree < len; degree++) {
        for (int row_ind = 0; row_ind < G.max_value; row_ind++) {
            for (int col_ind = 0; col_ind < G.max_value;
col_ind++) {
                M->values[row_ind][col_ind] = 0;
                for (int summand_ind = 0; summand_ind <
G.max_value; summand_ind++) {
                    M->values[row_ind][col_ind] +=
result.values[row_ind][summand_ind] * bin_relation_getValue(G,
summand_ind + 1, col_ind + 1);
                }
            }
        }
        for (int row_ind = 0; row_ind < G.max_value; row_ind++) {
            for (int col_ind = 0; col_ind < G.max_value;
col_ind++) {
                result.values[row_ind][col_ind] = M-
>values[row_ind][col_ind];
            }
        }
    }
}

int main () {
    matrix fullM = createMatrixFromArray(
            (int[]) {
                0, 1, 1, 1, 1, 1, 1,
              1, 0, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 1, 1, 1,
              1, 1, 1, 0, 1, 1, 1,
              1, 1, 1, 1, 0, 1, 1,
              1, 1, 1, 1, 1, 0, 1,
              1, 1, 1, 1, 1, 1, 0
            },7, 7
```

```
    );

    graph fullG = graph_createFromMatrix(fullM);

    matrix routes = getMemMatrix(7, 7);
    printf("Matrix of the number of paths with length 2 between
vertices, for fullG:\n");
    getMatrixOfRoutesAmount(fullG, 2, &routes);
    outputMatrix(routes);
    printf("Matrix of the number of paths with length 5 between
vertices, for fullG:\n");
    getMatrixOfRoutesAmount(fullG, 4, &routes);
    outputMatrix(routes);
}
```

```
"C:\Users\sovac\Desktop\discrete_math\ДИСКРЕТКА ЛАБА 4.1\graphs\defend_task.exe"
Matrix of the number of paths with length 2 between vertices, for fullG:
|6 5 5 5 5 5 5|
|5 6 5 5 5 5 5|
|5 5 6 5 5 5 5|
|5 5 5 6 5 5 5|
|5 5 5 5 6 5 5|
|5 5 5 5 5 6 5|
|5 5 5 5 5 5 6|
Matrix of the number of paths with length 5 between vertices, for fullG:
|186 185 185 185 185 185 185|
|185 186 185 185 185 185 185|
|185 185 186 185 185 185 185|
|185 185 185 186 185 185 185|
|185 185 185 185 186 185 185|
|185 185 185 185 185 186 185|
|185 185 185 185 185 185 186|

Process finished with exit code 0
```

В случае, когда длина маршрута равна 2, сумма по каждой строке (или столбцу) составит 6 + 5*6 = 36; $(n-1)^L = (7-1)^2 = 6^2 = 36$. А в случае, когда длина маршрута равна 4, сумма по каждой строке (или столбцу) составит 186 + 185*6 = 186 + 1110 = 1296; $(n-1)^L = (7-1)^4 = 6^4 = 1296$. В обоих случаях результаты работы программы сошлись с выведенной формулой.