**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;  
}

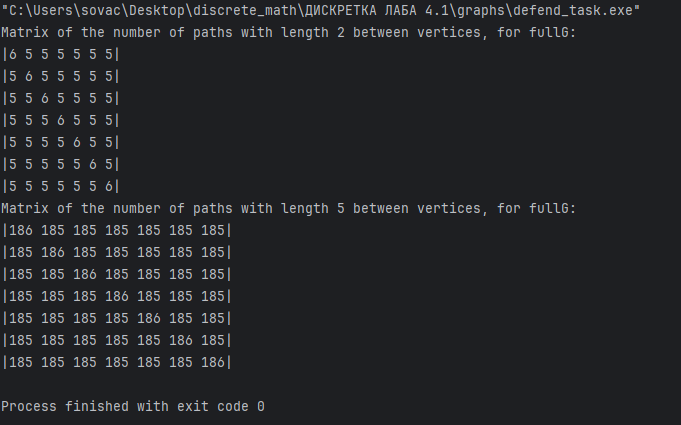
Данная версия функции isSequenceChain включена в исправленную версию лабораторной работы.

**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);  
}



В случае, когда длина маршрута равна 2, сумма по каждой строке (или столбцу) составит 6 + 5\*6 = 36; *(n-1)L* = (7-1)2 = 62 = 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 = 64 = 1296. В обоих случаях результаты работы программы сошлись с выведенной формулой.