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 №4.1

по дисциплине: «Дискретная математика»

по теме: **Графы, маршруты**

Выполнил/а: ст. группы ПВ-231

Чупахина София Александровна

Проверил: Рязанов Юрий Дмитриевич

Белгород, 2024

# Содержание

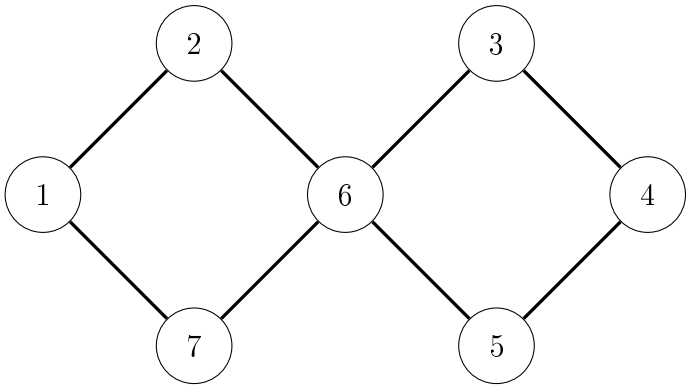


# Вариант 6

**а)** Матрица смежности графа $G_1$:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Диаграмма графа $G_2$:




**б)** Последовательности вершин:

1. (1, 7, 6, 3, 5)
2. (6, 5, 3, 6, 7, 1)
3. (7, 6, 3, 2, 1, 7)
4. (4, 3, 5, 3, 6, 5)
5. (1, 2, 7, 6, 2, 1)

# Задание 1

**Текст задания:** Представить графы $G_1$ и $G_2$ (см. «Варианты заданий», п.а) матрицей смежности, матрицей инцидентности, диаграммой.

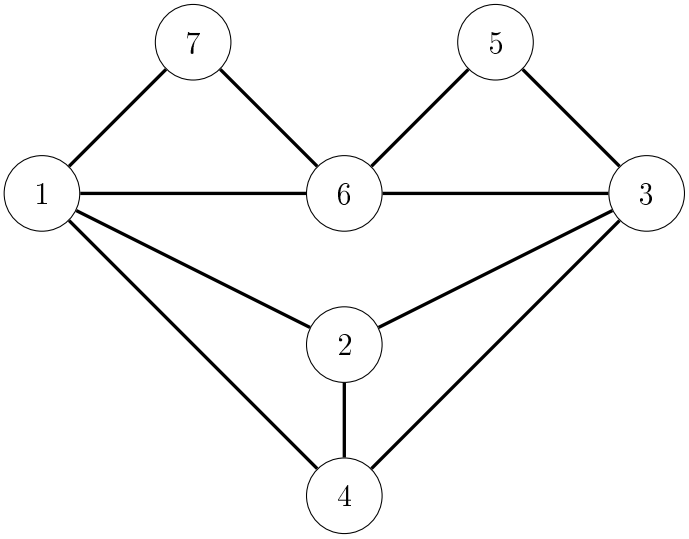
Для графа $G_1$ мы уже имеем матрицу смежности. Проанализировав ее можем сделать вывод, что эта матрица симметрична, а значит, глав является неориентированным — вершины связаны ребрами без учета направлений, и элементами множества можно считать двухэлементные подмножества множества $V$, а не упорядоченные пары на множестве $V$.

Составим для этого графа матрицу инцидентности. Строки такой матрицы соответствуют вершинам графа: их, как мы можем видеть по размеру матрицы смежности, 7. Значит, 7 рядов будет в матрице инцидентности. Столбцы же соответствуют ребрам графа, и их количество равно количеству единиц в части матрицы над главной диагональю: раз граф неориентированный и порядок вершин, составляющих ребро, не важен, то единицы под главной диагональю дублируют ребра графа, уже отображенные в верхней части матрицы. Единиц над главной диагональю 11, и столько же столбцов будет в матрице инцидентности.

Поскольку граф неориентированный, элементами матрицы будут являться только нули и единицы: единица означает, что вершина, соответствующая текущей строке, является одним из концов ребра, соответствующего текущему столбцу, то есть инцидента этому ребру; ноль означает, что данная вершина не является концом данного ребра. Заполним матрицу значениями; рассматривая последовательно каждую единицу матрицы смежности на позиции $(i, j)$, запишем единицы в строки $i$ и $j$ в столбце $k$, где $k$ — номер рассматриваемой единицы. Результатом будет следующая матрица:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Перейдем к составлению диаграммы этого графа, опираясь на его матрицу смежности. Как уже было упомянуто, этот граф имеет семь вершин, потому диаграмма будет включать в себя 7 окружностей. Проходя по матрице смежности (вернее, по ее части выше главной диагонали), будем соединять линиями окружности под номерами $i$ и $j$, если в матрице смежности элемент на позиции $(i, j)$ равен 1.




Для графа $G_2$ мы уже имеем диаграмму, но не имеем матрицы смежности и инцидентности. Заметим, что, поскольку диаграмма не имеет в своем составе стрелок (то есть направленных дуг), этот граф также является неориентированным, и все связанные с этим выводы для графа $G_1$ справедливы и для этого графа. Составляя матрицу смежности, зададим пустую матрицу размером $7 \times 7$ (поскольку у диаграммы также 7 вершин) и, проходя по каждому ее элементу, позицию которого обозначим как $(i, j)$, будем обращать его в 1, если на диаграмме вершины под номерами $i$ и $j$ соединены. Результатом будет следующая матрица:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

После получения матрицы смежности, можем получить матрицу инцидентности по тому же алгоритму, что и для графа $G_1$. Матрица будет содержать 7 строк (размер матрицы смежности и количество вершин на диаграмме) и 8 столбцов (количество единиц над главной диагональю в матрице смежности и количество ребер на диаграмме). Заполняя ее значениями, для каждой единицы, расположенной над главной диагональю матрицы смежности, будем записывать единицы в строки $i$ и $j$ в столбце $k$, где $k$ — номер рассматриваемой единицы, $(i, j)$ — ее позиция в матрице смежности. Результатом будет следующая матрица:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

# Задание 2

**Текст задания:** Определить, являются ли последовательности вершин (см. «Варианты заданий», п.б) маршрутом, цепью, простой цепью, циклом, простым циклом в графах $G_1$ и $G_2$ (см. «Варианты заданий», п.а).

Будем по очереди рассматривать заданные последовательности вершин и их свойства в обоих графах.

**№1: (1, 7, 6, 3, 5)**

Рассмотрим свойства этой последовательности в графе $G_1$.

Маршрут — такая последовательность вершин, в которой каждые две соседние вершины являются смежными. Смотря на диаграмму графа $G_1$, мы можем видеть, что: 1) вершины 1 и 7 соединены, 2) вершины 7 и 6 соединены, 3) вершины 6 и 3 соединены, 4) вершины 3 и 5 соединены. Если вершины соединены на диаграмме, значит, они смежны. Следовательно, данная последовательность является маршрутом.

Цепь — маршрут, в котором все ребра различны. Мы уже выяснили, что данная последовательность — маршрут. В его состав будут входить 4 ребра, инцидентные вершинам 1 и 7, 7 и 6, 6 и 3, 3 и 5 соответственно. Двухэлементные подмножества, образованные этими парами вершин, попарно неравны, значит, и ребра графа будут попарно различны. Следовательно, данная последовательность является цепью.

Простая цепь — цепь, в котором все вершины различны. Мы уже выяснили, что данная последовательность — цепь. В последовательности вершин (1, 7, 6, 3, 5) нет повторов. Следовательно, данная последовательность является также и простой цепью.

Цикл — цепь, в которой совпадают начальная и конечная вершины. Несмотря на то, что данная последовательность является цепью, начальная и конечная вершины (1 и 5) в ней не совпадают. Значит, она не является циклом.

Простой цикл — простая замкнутая цепь, являющаяся циклом. Данная последовательность является простой цепью, но не является циклом, следовательно, она не является простым циклом.

Анализируя эту последовательность для графа $G_2$, нам достаточно доказать, что для него она является маршрутом. Остальные свойства зависят от характеристик самой последовательности (повторяются ли в ней вершины или их пары, равны ли начальная и конечная вершины), а не графа. Смотря на диаграмму графа $G_2$, мы можем видеть, что вершины 1 и 7, 7 и 6, 6 и 3 соединены, однако не соединены вершины 3 и 5. Значит, данная последовательность не является маршрутом. А так как она не является маршрутом, она не может являться цепью, простой цепью, циклом и простым циклом.

**№2: (6, 5, 3, 6, 7, 1)**

Рассмотрим свойства этой последовательности в графе $G_1$.

Смотря на диаграмму графа $G_1$, мы можем видеть, что: 1) вершины 6 и 5 соединены, 2) вершины 5 и 3 соединены, 3) вершины 3 и 6 соединены, 4) вершины 6 и 7 соединены, 5) вершины 7 и 1 соединены. Если вершины соединены на диаграмме, значит, они смежны. Следовательно, данная последовательность является маршрутом. В его состав будут входить 5

ребер, инцидентных вершинам 6 и 5, 5 и 3, 3 и 6, 6 и 7, 7 и 1 соответственно. Двухэлементные подмножества, образованные этими парами вершин, попарно неравны, значит, и ребра графа будут попарно различны. Следовательно, данная последовательность является цепью. Однако в последовательности вершин (6, 5, 3, 6, 7, 1) дважды повторяется вершина 6. Следовательно, данная последовательность не является простой цепью. Несмотря на то, что данная последовательность является цепью, начальная и конечная вершины (6 и 1) в ней не совпадают. Значит, она не является циклом. А раз данная последовательность не является циклом, следовательно, простым циклом она также не является.

Является ли эта последовательность маршрутом для графа $G_2$? Смотря на диаграмму графа $G_2$, мы можем видеть, что вершины 6 и 5 соединены, однако не соединены вершины 5 и 3. Значит, данная последовательность не является маршрутом. А так как она не является маршрутом, она не может являться цепью, простой цепью, циклом и простым циклом.

**№3: (7, 6, 3, 2, 1, 7)**

Рассмотрим свойства этой последовательности в графе $G_1$.

Смотря на диаграмму графа $G_1$, мы можем видеть, что: 1) вершины 7 и 6 соединены, 2) вершины 6 и 3 соединены, 3) вершины 3 и 2 соединены, 4) вершины 2 и 1 соединены, 5) вершины 1 и 7 соединены. Если вершины соединены на диаграмме, значит, они смежны. Следовательно, данная последовательность является маршрутом. В его состав будут входить 5 ребер, инцидентных вершинам 7 и 6, 6 и 3, 3 и 2, 2 и 1, 1 и 7 соответственно. Двухэлементные подмножества, образованные этими парами вершин, попарно неравны, значит, и ребра графа будут попарно различны. Следовательно, данная последовательность является цепью. В последовательности вершин (7, 6, 3, 2, 1, 7) дважды повторяется вершина 7, однако она выполняет роль начальной и конечной в этой последовательности, что дает нам право заявить, что данная последовательность является замкнутой простой цепью, то есть одновременно простой цепью и циклом. А из этого следует, что она является также и простым циклом.

Является ли эта последовательность маршрутом для графа $G_2$? Смотря на диаграмму графа $G_2$, мы можем видеть, что вершины 7 и 6, 6 и 3 соединены, однако не соединены вершины 3 и 2. Значит, данная последовательность не является маршрутом. А так как она не является маршрутом, она не может являться цепью, простой цепью, циклом и простым циклом.

**№4: (4, 3, 5, 3, 6, 5)**

Рассмотрим свойства этой последовательности в графе $G_1$.

Смотря на диаграмму графа $G_1$, мы можем видеть, что: 1) вершины 4 и 3 соединены, 2) вершины 3 и 5 соединены, 3) вершины 5 и 3 соединены, 4) вершины 3 и 6 соединены, 5) вершины 6 и 5 соединены. Если вершины соединены на диаграмме, значит, они смежны. Следовательно, данная последовательность является маршрутом. В его состав будут входить 5 ребер, инцидентных вершинам 4 и 3, 3 и 5, 5 и 3, 3 и 6, 6 и 5 соответственно. Однако двухэлементные подмножества, образованные этими парами вершин (3, 5) и (5, 3), одинаковы, значит, маршрут дважды проходит по одному ребру. Следовательно, данная последовательность не является цепью. А раз так, она не может являться простой цепью, циклом и простым циклом.

Является ли эта последовательность маршрутом для графа $G_2$? Смотря на диаграмму графа $G_2$, мы можем видеть, что вершины 4 и 3 соединены, однако не соединены вершины 3 и 5. Значит, данная последовательность не является маршрутом. А так как она не является

маршрутом, она не может являться цепью, простой цепью, циклом и простым циклом.

**№5: (1, 2, 7, 6, 2, 1)**

Рассмотрим свойства этой последовательности в графе $G_1$.

Смотря на диаграмму графа $G_1$, мы можем видеть, что: 1) вершины 1 и 2 соединены, 2) вершины 2 и 7 не соединены. Если вершины не соединены на диаграмме, значит, они не смежны. Следовательно, условие маршрута нарушается, и данная последовательность не является маршрутом. А раз так, она не может являться цепью, простой цепью, циклом и простым циклом.

Является ли эта последовательность маршрутом для графа $G_2$? Смотря на диаграмму графа $G_2$, мы можем видеть, что вершины 1 и 2 соединены, однако не соединены вершины 2 и 7. Значит, данная последовательность не является маршрутом. А так как она не является маршрутом, она не может являться цепью, простой цепью, циклом и простым циклом.

# Задание 3

**Текст задания:** Написать программу, определяющую, является ли заданная последовательность вершин (см. «Варианты заданий», п.б) маршрутом, цепью, простой цепью, циклом, простым циклом в графах G1 и G2 (см. «Варианты заданий», п.а).

Решим для начала, в какой форме мы будем представлять такую структуру, как граф. Как мы уже выяснили, есть три способа представления графа: матрица смежности, матрица инцидентности и диаграмма, и все три задают граф однозначно. Матрица смежности, по сути, является отношением на множестве вершин и показывает, какие упорядоченные пары вершин входят в множество дуг, то есть смежны друг другу. Мы говорим об упорядоченных парах, потому что с помощью матрицы смежности, заполненной только нулями и единицами, можно представлять и ориентированные графы; если граф неориентирован, то информация о ребрах, по сути, дублируется в матрице смежности. При этом, какие бы объекты ни отображали вершины графа, их всегда можно отобразить на отрезок множества натуральных чисел [1; max], где max — наибольший возможный номер вершины. В течение семестра мы много работали именно с бинарными отношениями на множестве идущих подряд натуральных чисел на отрезке [1; max]. Поэтому мы можем задать структуру «граф» просто как переименованную структуру «бинарное отношение».

В течение работы нам понадобиться работать с матрицами (не только бинарными) отдельно от структуры «граф», хотя бы просто для удобной инициализации. Заранее приведем модуль с заданной структурой «матрица» и базовыми функциями для нее — часть модуля, созданного в прошлом семестре в рамках курса ОП.

```c
#ifndef INC_MATRIX_H
#define INC_MATRIX_H

#include <stdbool.h>
#include <stdint.h>

typedef struct {
```

```
8       int **values; // элементы матрицы
9       int nRows; // количество рядов
10      int nCols; // количество столбцов
11  } matrix;
12  typedef struct {
13      int rowIndex;
14      int colIndex;
15  } position;
16
17  //размещает в динамической памяти и возвращает матрицу размером nRows
        на nCols
18  matrix getMemMatrix(int nRows, int nCols);
19  //осуществляет ввод матрицы и записывает данные по адресу m
20  void inputMatrix(matrix *m);
21  // выводит на экран матрицу m
22  void outputMatrix(matrix m);
23  //возвращает позицию последнего элемента со значением value, если идти
        по матрице сверху вниз и по ее строкам слева направо
24  position getValuePos(matrix m, int value);
25  //возвращает матрицу - произведение матриц m1 и m2
26  matrix mulMatrices(matrix m1, matrix m2);
27  //возвращает матрицу размера nRows на nCols, построенную из элементов
        массива a
28  matrix createMatrixFromArray(const int *values, size_t nRows, size_t
        nCols);
29
30  #endif
```

matrix/matrix.h

```
1   #include "matrix.h"
2   #include <malloc.h>
3   #include <stdio.h>
4   #include <assert.h>
5   #ifndef INC_MATRIX_C
6   #define INC_MATRIX_C
7
8   matrix getMemMatrix(int nRows, int nCols) {
9       int **values = (int **) malloc(sizeof(int*) * nRows);
10      for (int row_ind = 0; row_ind < nRows; row_ind++) {
11          values[row_ind] = (int *) malloc(sizeof(int) * nCols);
12      }
13      return (matrix){values, nRows, nCols};
14  }
15
16  void inputMatrix(matrix *m) {
17      for (int row_ind = 0; row_ind < m->nRows; row_ind++) {
18          for (int col_ind = 0; col_ind < m->nCols; col_ind++) {
```

```
            scanf("%d", &(m->values[row_ind][col_ind]));
        }
    }
}

void outputMatrix(matrix m) {
    if (m.nRows != 0 && m.nCols != 0) {
        for (int row_ind = 0; row_ind < m.nRows; row_ind++) {
            printf("|");
            for (int col_ind = 0; col_ind < m.nCols; col_ind++) {
                printf("%d ", m.values[row_ind][col_ind]);
            }
            printf("\b|\n");
        }
    }
}

position getValuePos(matrix m, int value) {
    position pos = (position) {0, 0};
    for (int row_index = 0; row_index < m.nRows; row_index++) {
        for (int col_index = 0; col_index < m.nCols; col_index++) {
            if (m.values[row_index][col_index] == value) {
                pos.rowIndex = row_index;
                pos.colIndex = col_index;
                break;
            }
        }
    }
    return pos;
}

matrix mulMatrices(matrix m1, matrix m2) {
    assert(m1.nCols == m2.nRows);
    matrix result = getMemMatrix(m1.nRows, m2.nCols);
    for (int row_ind = 0; row_ind < result.nRows; row_ind++) {
        for (int col_ind = 0; col_ind < result.nCols; col_ind++) {
            result.values[row_ind][col_ind] = 0;
            for (int summand_ind = 0; summand_ind < m1.nRows;
   summand_ind++) {
                result.values[row_ind][col_ind] +=
   m1.values[row_ind][summand_ind] * m2.values[summand_ind][col_ind];
            }
        }
    }
    return result;
}

```

```
64 matrix createMatrixFromArray(const int *values, size_t nRows, size_t
   nCols) {
65     matrix m = getMemMatrix(nRows, nCols);
66     int k = 0;
67     for (int row_ind = 0; row_ind < nRows; row_ind++) {
68         for (int col_ind = 0; col_ind < nCols; col_ind++) {
69             m.values[row_ind][col_ind] = values[k++];
70         }
71     }
72     return m;
73 }
74
75 #endif
```

matrix/matrix.c

После задания структуры «граф» перейдем к функциям, определяющим свойства последовательности вершин для конкретного графа. При определении того, является ли последовательность маршрутом, мы определяем, входит ли в матрицу графа каждая пара, образованная соседними вершинами последовательности, то есть действительно ли все соседние вершины последовательности смежны. Для цепей, помимо этого, мы задаем массив, хранящий подряд значения вершин, инцидентных каждому последующему ребру, и для каждого вновь образованного ребра мы сравниваем инцидентные ему вершины с инцидентными вершинами всех предыдущих, проверяя, нет ли среди них равных пар. Для простых цепей мы создаем массив вершин и для каждой новой вершины определяем, была ли она включена в этот массив ранее (исключая ситуацию, когда повторяющиеся вершины — первая и последняя в последовательности; так функция сможет учесть и замкнутые простые цепи). Ну и для циклов и простых циклов вызываются функции определения, является ли последовательность цепью или простой цепью соответственно, а дополнительно сравниваются первый и последний элементы, то есть проверяется условие цикличности.

Все эти функции можно последовательно вызвать в объединяющей их функции printSequenceProp которая и будет выводить результаты на экран. А затем в теле main задаются (с помощью матриц) сами графы, последовательности, которым ставятся в соответствие их длины, и с помощью цикла эти последовательности (и их длины) поочередно перебираются, и на экран выводятся их свойства как относительно графа $G_1$, так и относительно графа $G_2$.

```
1 #include "ДИСКРЕТКА../../ ЛАБА
   3.1/bin_relations/bin_relation_definition_input_output.c"
2 #include "ДИСКРЕТКА../../ ЛАБА
   3.1/bin_relations/bin_relations_operations.c"
3 #include "ДИСКРЕТКА../../ ЛАБА
   3.1/bin_relations/bin_relations_properties.c"
4 #include "../matrix/matrix.c"
5
6 #include <stdbool.h>
7
8 typedef bin_relation graph;
9
```

```
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;
}

bool isSequenceRoute(graph G, int *s, int n) {
    bool result = s[0] <= G.max_value;
    for (int vert_ind = 1; vert_ind < n && result; vert_ind++) {
        if (s[vert_ind] > G.max_value || !bin_relation_getValue(G, s[vert_ind], s[vert_ind-1])) {
            result = false;
        }
    }
    return result;
}

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;
}

bool isSequenceSimpleChain(graph G, int *s, int n) {
    bool result = s[0] <= G.max_value;
    int V_array[n];
    V_array[0] = s[0];
```

```
51      for (int vert_ind = 1; vert_ind < n && result; vert_ind++) {
52          if (s[vert_ind] > G.max_value || !bin_relation_getValue(G,
    s[vert_ind], s[vert_ind-1])) {
53              result = false;
54          } else {
55              V_array[vert_ind] = s[vert_ind];
56              for (int vert_ind2 = 0; vert_ind2 < vert_ind-1;
    vert_ind2++) {
57                  if (V_array[vert_ind2] == V_array[vert_ind] &&
    !(vert_ind2 == 0 && vert_ind == n-1)) {
58                      result = false;
59                  }
60              }
61          }
62      }
63      return result;
64  }
65
66  bool isSequenceCycle(graph G, int *s, int n) {
67      return isSequenceChain(G, s, n) && s[0] == s[n-1];
68  }
69
70  bool isSequenceSimpleCycle(graph G, int *s, int n) {
71      return isSequenceSimpleChain(G, s, n) && s[0] == s[n-1];
72  }
73
74  void printSequenceProperties(graph G, int *s, int n) {
75      printf("This sequence is: ");
76      if (isSequenceRoute(G, s, n)) {
77          printf("route, ");
78      } else {
79          printf("not route, ");
80      }
81      if (isSequenceChain(G, s, n)) {
82          printf("chain, ");
83      } else {
84          printf("not chain, ");
85      }
86      if (isSequenceSimpleChain(G, s, n)) {
87          printf("simple chain, ");
88      } else {
89          printf("not simple chain, ");
90      }
91      if (isSequenceCycle(G, s, n)) {
92          printf("cycle, ");
93      } else {
94          printf("not cycle, ");
```

```
    }
    if (isSequenceSimpleCycle(G, s, n)) {
        printf("simple cycle.\n");
    } else {
        printf("not simple cycle.\n");
    }
}

int main () {
    matrix M1 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 1, 0, 1, 1,
                1, 0, 1, 1, 0, 0, 0,
                0, 1, 0, 1, 1, 1, 0,
                1, 1, 1, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            },7, 7
    );
    matrix M2 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 0, 1, 0, 1, 0, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 1, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            }, 7, 7
    );

    graph G1 = graph_createFromMatrix(M1);
    graph G2 = graph_createFromMatrix(M2);

    int sequence1[5] = {1, 7, 6, 3, 5};
    int sequence2[6] = {6, 5, 3, 6, 7, 1};
    int sequence3[6] = {7, 6, 3, 2, 1, 7};
    int sequence4[6] = {4, 3, 5, 3, 6, 5};
    int sequence5[6] = {1, 2, 7, 6, 2, 1};

    int *sequences[5] = {sequence1, sequence2, sequence3, sequence4,
   sequence5};
    int lens[5] = {5, 6, 6, 6, 6};

    for (int i = 0; i < 5; i++) {
        printf("Sequence No %d, for G1:\n", i+1);
```

```
141         printSequenceProperties(G1, sequences[i], lens[i]);
142         printf("Sequence No %d for G2:\n", i+1);
143         printSequenceProperties(G2, sequences[i], lens[i]);
144         printf("\n");
145     }
146 }
```

graphs/graphs_main.c

```
"C:\Users\sovac\Desktop\discrete_math\ДИСКРЕТКА ЛАБА 4.1\graphs\graphs_main.exe"
Sequence No 1, for G1:
This sequence is: route, chain, simple chain, not cycle, not simple cycle.
Sequence No 1 for G2:
This sequence is: not route, not chain, not simple chain, not cycle, not simple cycle.

Sequence No 2, for G1:
This sequence is: route, chain, not simple chain, not cycle, not simple cycle.
Sequence No 2 for G2:
This sequence is: not route, not chain, not simple chain, not cycle, not simple cycle.

Sequence No 3, for G1:
This sequence is: route, chain, simple chain, cycle, simple cycle.
Sequence No 3 for G2:
This sequence is: not route, not chain, not simple chain, not cycle, not simple cycle.

Sequence No 4, for G1:
This sequence is: route, not chain, not simple chain, not cycle, not simple cycle.
Sequence No 4 for G2:
This sequence is: not route, not chain, not simple chain, not cycle, not simple cycle.

Sequence No 5, for G1:
This sequence is: not route, not chain, not simple chain, not cycle, not simple cycle.
Sequence No 5 for G2:
This sequence is: not route, not chain, not simple chain, not cycle, not simple cycle.
```

Как мы можем видеть, заключения на экране совпадают с выводами, сделанными в задании 2.

# Задание 4

**Текст задания:** Написать программу, получающую все маршруты заданной длины, выходящие из заданной вершины. Использовать программу для получения всех маршрутов заданной длины в графах $G_1$ и $G_2$ (см. «Варианты заданий», п.а).

Для выполнения этой задачи нам понадобится четыре функции. Первая — вспомогательная функция вывода последовательности вершин на экран. Вторая — функция рекурсивного вызова allRoutesWithLenFromVert_, формирующая маршрут заданной длины и за конкретный вызов задающая значение элемента под индексом cur_pos в ней (отсчет начинается с нуля). Для этого перебираются все возможные вершины графа, и для каждой проверяется, смеж-

на ли она с последней на текущий момент вершиной в последовательности (то есть включена ли такая пара в матрицу смежности). Если это так, то элемент маршрута под индексом cur_pos приравнивается к найденному элементу. Затем проверяется, достигла ли последовательность заданной длины: если достигла, то она выводится на экран (с помощью функции printSequence). Иначе рекурсивная функция запускается для тех же данных, но с увеличенным на один индексом cur_pos. Третья функция, allRoutesWithLenFromVert — обертка для рекурсивной функции. Она создает последовательность, которая будет формироваться цепочкой рекурсивных вызовов, задает ее первый элемент (он передается функции как аргумент) и запускает рекурсивную функцию с cur_pos равным 1 (нулевой элемент уже сформирован). Наконец, четвертая функция, allRoutesWithLen, перебирает все возможные вершины графа и вызывая allRoutesWithLenFromVert для каждой из них. Таким образом, на экран выводятся все маршруты заданной длины, а не только начинающиеся с определенной вершины.

В качестве примера вызовем эту функцию сначала для графа $G_1$ и длины 2 (в маршрутах будет 2 ребра и, соответственно, 3 вершины), потом — для графа $G_2$ и длины 1 (1 ребро, 2 вершины).

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

void allRoutesWithLenFromVert_(graph G, int len, 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)) {
            W[cur_pos] = cur_el;
            if (cur_pos == len) {
                printSequence(W, len+1);
            } else {
                allRoutesWithLenFromVert_(G, len, W, cur_pos+1);
            }
        }
    }
}

void allRoutesWithLenFromVert(graph G, int len, int vert) {
    int W[G.max_value];
    W[0] = vert;
    allRoutesWithLenFromVert_(G, len, W, 1);
}

void allRoutesWithLen(graph G, int len) {
    for (int cur_start_el = 1; cur_start_el <= G.max_value;
   cur_start_el++) {
```

```
        allRoutesWithLenFromVert(G, len, cur_start_el);
        printf("\n");
    }
}

int main () {
    matrix M1 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 1, 0, 1, 1,
                1, 0, 1, 1, 0, 0, 0,
                0, 1, 0, 1, 1, 1, 0,
                1, 1, 1, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            },7, 7
    );
    matrix M2 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 0, 1, 0, 1, 0, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 1, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            }, 7, 7
    );

    graph G1 = graph_createFromMatrix(M1);
    graph G2 = graph_createFromMatrix(M2);

    printf("All routes with len 1, for G1:\n");
    allRoutesWithLen(G1, 1);
    printf("All routes with len 2, for G2:\n");
    allRoutesWithLen(G2, 2);
    printf("\n");
}
```

graphs/graphs_main.c

```
All routes with len 1, for G1:
(1, 2)  (1, 4)  (1, 6)  (1, 7)
(2, 1)  (2, 3)  (2, 4)
(3, 2)  (3, 4)  (3, 5)  (3, 6)
(4, 1)  (4, 2)  (4, 3)
(5, 3)  (5, 6)
(6, 1)  (6, 3)  (6, 5)  (6, 7)
(7, 1)  (7, 6)
All routes with len 2, for G2:
(1, 2, 1)       (1, 2, 6)       (1, 7, 1)       (1, 7, 6)
(2, 1, 2)       (2, 1, 7)       (2, 6, 2)       (2, 6, 3)       (2, 6, 5)       (2, 6, 7)
(3, 4, 3)       (3, 4, 5)       (3, 6, 2)       (3, 6, 3)       (3, 6, 5)       (3, 6, 7)
(4, 3, 4)       (4, 3, 6)       (4, 5, 4)       (4, 5, 6)
(5, 4, 3)       (5, 4, 5)       (5, 6, 2)       (5, 6, 3)       (5, 6, 5)       (5, 6, 7)
(6, 2, 1)       (6, 2, 6)       (6, 3, 4)       (6, 3, 6)       (6, 5, 4)       (6, 5, 6)       (6, 7, 1)       (6, 7, 6
)
(7, 1, 2)       (7, 1, 7)       (7, 6, 2)       (7, 6, 3)       (7, 6, 5)       (7, 6, 7)
```

# Задание 5

**Текст задания:** Написать программу, определяющую количество маршрутов заданной длины между каждой парой вершин графа. Использовать программу для определения количества маршрутов заданной длины между каждой парой вершин в графах $G_1$ и $G_2$ (см. «Варианты заданий», п.а).

Воспользуемся упомянутым в лекции свойством, что матрица, в которой элемент на позиции $(i, j)$ указывает, сколько маршрутов заданной длины существует между вершинами $i$ и $j$ для некоторого графа, равна матрице смежности этого графа, возведенной в степень, равную заданной длине. Нам остается лишь реализвать возведение матрицы в степень. В библиотеке для работы с матрицами есть функция умножения матриц, однако в нашем случае использовать ее будет затруднительно, потому что матрица смежности в нашем случае задается не структурой matrix, а массивом целых чисел (отдельное значение задается его конкретным битом). Поэтому возьмем за основу этот алгоритм, но слегка изменим его.

Результирующая матрица будет записываться по переданному функции адресу M. Сначала может показаться, что нам достаточно задать этой матрице начальное значение так, чтобы она была равна исходной матрице смежности, а потом умножать ее на матрицу смежности нужное количество раз. Но если мы будем последовательно перезаписывать каждое значение в результирующей матрице при умножении, то при вычислении значений следующих элементов мы будем опираться на уже перезаписанные, и результаты будут искажены. Поэтому необходимо создать матрицу, хранящую промежуточный результат умножения, и при перезаписи значений результирующей матрицы опираться на нее.

Таким образом, в функции можно выделить три блока: 1) создание матрицы промежуточного результата и приравнивание ее к исходной матрице смежности, 2) перезапись результирующей матрицы как произведения матрицы смежности и матрицы промежуточного результата, 3) обновление матрицы промежуточного результата и приравнивание ее к результирующей. Шаги 2 и 3 выполняются в цикле до тех пор, пока не будет достигнута степень, равная длине маршрутов, матрицу количества которых мы хотим получить. Для доступа к значениям результирующей матрицы и матрицы промежуточного результата используются ключ values и два индекса, для доступа к значениям матрицы смежности — функция bin_relation_getValue, причем важно учесть, что в обычных матрицах индексы элементов идут с нуля, а в матрице

смежности мы используем для обращения значения элементов, идущие с единицы.

В качестве примера вызовем эту функцию дважды для графа $G_2$, сначала для длины 2 (в маршрутах будет 2 ребра и, соответственно, 3 вершины), потом — для длины 3 (3 ребра, 4 вершины).

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
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 M1 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 1, 0, 1, 1,
                1, 0, 1, 1, 0, 0, 0,
                0, 1, 0, 1, 1, 1, 0,
                1, 1, 1, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
```

```
            },7, 7
    );
    matrix M2 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 0, 1, 0, 1, 0, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 1, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            }, 7, 7
    );

    graph G1 = graph_createFromMatrix(M1);
    graph G2 = graph_createFromMatrix(M2);

    matrix routes = getMemMatrix(7, 7);
    printf("Matrix of the number of paths with length 2 between vertices, for G2:\n");
    getMatrixOfRoutesAmount(G2, 2, &routes);
    outputMatrix(routes);
    printf("Matrix of the number of paths with length 3 between vertices, for G2:\n");
    getMatrixOfRoutesAmount(G2, 3, &routes);
    outputMatrix(routes);
}
```

graphs/graphs_main.c

```
Matrix of the number of paths with length 2 between vertices, for G2:
|2 0 0 0 0 2 0|
|0 2 1 0 1 0 2|
|0 1 2 0 2 0 1|
|0 0 0 2 0 2 0|
|0 1 2 0 2 0 1|
|2 0 0 2 0 4 0|
|0 2 1 0 1 0 2|
Matrix of the number of paths with length 3 between vertices, for G2:
|0 4 2 0 2 0 4|
|4 0 0 2 0 6 0|
|2 0 0 4 0 6 0|
|0 2 4 0 4 0 2|
|2 0 0 4 0 6 0|
|0 6 6 0 6 0 6|
|4 0 0 2 0 6 0|
```

Вручную посчитать все пути из каждой вершины в каждую и проверить, таким образом, истинность этих матриц не представляется возможным, но в принципе результат соответствует здравому смыслу. Например, для длины 2 для каждой вершины количество маршрутов, начинающихся и заканчивающихся в ней самой, равно количеству вершин, с которыми она смежна: формируя маршруты, мы можем перейти в любую из этих вершин и вернуться обратно, и длина маршрута будет составлять ровно 2. Между смежными вершинами же (для этого конкретного графа) нельзя проложить ни одного маршрута длины 2: одно ребро уйдет, чтобы достигнуть этой вершины, второе — заканчивает маршрут где угодно, но не в этой вершине, а «обходных» путей длины 2 к смежным вершинам нет. И наоборот, для длины 3 ни для одной вершины не существует маршрута, начинающегося и заканчивающегося ей самой, зато между смежными вершинами можно проложить не один маршрут. Также можно заметить, что полученные матрицы тоже симметрична — количество путей из одной вершины в другую будет одинаковым, если поменять местами начало и конец.

# Задание 6

**Текст задания:** Написать программу, определяющую все маршруты заданной длины между заданной парой вершин графа. Использовать программу для определения всех маршрутов заданной длины между заданной парой вершин в графах $G_1$ и $G_2$ (см. «Варианты заданий», п.а).

По сути, эта задача сводится к задаче 4: поиску всех маршрутов заданной длины, начинающихся с конкретной вершины — только теперь в список аргументов добавляется еще одна вершина, которой маршрут должен заканчиваться, и после достижения требуемой длины маршрута необходимо проверить еще одно условие: равна ли последняя вершина полученного маршрута той, которая передана функции как аргумент. Если нет, полученный маршрут не

выводится на экран.

В качестве примера вызовем эту функцию дважды для графа $G_2$, сначала для длины 2, начальной вершины 3 и конечной вершины 5, потом — длины 3, начальной вершины 6 и конечной вершины 5.

```
ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
void allRoutesWithLenFromVertToVert_(graph G, int len, int vert2, 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)) {
            W[cur_pos] = cur_el;
            if (cur_pos == len) {
                if (cur_el == vert2) {
                    printSequence(W, len+1);
                }
            } else {
                allRoutesWithLenFromVertToVert_(G, len, vert2, W,
   cur_pos+1);
            }
        }
    }
}

void allRoutesWithLenFromVertToVert(graph G, int len, int vert, int
   vert2) {
    int W[G.max_value];
    W[0] = vert;
    allRoutesWithLenFromVertToVert_(G, len, vert2, W, 1);
}

int main () {
    matrix M1 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 1, 0, 1, 1,
                1, 0, 1, 1, 0, 0, 0,
                0, 1, 0, 1, 1, 1, 0,
                1, 1, 1, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            },7, 7
    );
    matrix M2 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 1, 0, 1, 0,
```

```
                0, 0, 1, 0, 1, 0, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 1, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            }, 7, 7
    );

    graph G1 = graph_createFromMatrix(M1);
    graph G2 = graph_createFromMatrix(M2);

    printf("\nAll routes with len 2 from vertex 3 to vertex 5, for G2:\n");
    allRoutesWithLenFromVertToVert(G2, 2, 3, 5);
    printf("\nAll routes with len 3 from vertex 6 to vertex 5, for G2:\n");
    allRoutesWithLenFromVertToVert(G2, 3, 6, 5);
    printf("\n");
}
```

graphs/graphs_main.c

```
All routes with len 2 from vertex 3 to vertex 5, for G2:
(3, 4, 5)       (3, 6, 5)
All routes with len 3 from vertex 6 to vertex 5, for G2:
(6, 2, 6, 5)    (6, 3, 4, 5)    (6, 3, 6, 5)    (6, 5, 4, 5)    (6, 5, 6, 5)    (6, 7, 6, 5)
```

Несложно вручную проследить эти маршруты на графе и убедиться, что они верны. Кстати, можем заметить, что в каждом случае количество выведенных маршрутов совпадает с тем, которое записано в матрицах в задании 5 на соответствующих позициях (строка 3, столбец 5 и строка 6, столбец 5 соответственно).

# Задание 7

**Текст задания:** Написать программу, получающую все простые максимальные цепи, выходящие из заданной вершины графа. Использовать программу для получения всех простые максимальных цепей, выходящих из заданной вершины в графах $G_1$ и $G_2$» (см. «Варианты заданий», п.а).

Алгоритм имеет общие черты с таковыми в задачах 4 и 6, однако для него нам потребуется не только хранить простую цепь W, но и сверяться с тем, какие вершины в него уже входили. Можно использовать для этого отдельное множество встреченных вершин V, а можно использовать последовательность, хранящуюся непосредственно в простой цепи W — ведь если она будет формироваться по правилам, то вершины в ней не будут повторяться; к тому же, если массивы V и W формируются последовательно, то они на каждом шаге, пока условие цепи еще выполняется, будут, по сути, равны. Поэтому логичнее будет оптимизировать программу и использовать только W.

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 allMaxSimpleChainsFromVert_, формирующая максимальную простую цепь и за конкретный вызов задающая значение элемента под индексом cur_pos в ней (отсчет начинается с нуля). 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 снова перебираются все возможные вершины, и для каждой, если она смежна с последней, проверяется, входила ли она в последовательность ранее. Если для всех вершин это условие выполняется, то максимальная простая цепь считается сформированной и выводится на экран (с помощью функции printSequence). Иначе рекурсивная функция запускается для тех же данных, но с увеличенным на один индексом cur_pos. И третья функция, allMaxSimpleChainsFromVert — обертка для рекурсивной функции. Она создает последовательность, которая будет формироваться цепочкой рекурсивных вызовов, задает ее первый элемент (он передается функции как аргумент) и запускает рекурсивную функцию с cur_pos равным 1 (нулевой элемент уже сформирован).

В качестве примера вызовем эту функцию сначала для графа $G_1$ и вершины 2, потом — для графа $G_2$ и вершины 2.

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

void allMaxSimpleChainsFromVert_(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;
            bool final_flag = true;
            for (int k = 1; k <= G.max_value; k++) {
                final_flag *= !bin_relation_getValue(G, k, cur_el) ||
  isInSequence(W, cur_pos, k);
            }
            if (final_flag) {
                printSequence(W, cur_pos+1);
            } else {
                allMaxSimpleChainsFromVert_(G, W, cur_pos+1);
            }
        }
    }
```

```c
}

void allMaxSimpleChainsFromVert(graph G, int vert) {
    int W[G.max_value], V[G.max_value];
    W[0] = vert;
    allMaxSimpleChainsFromVert_(G, W, 1);
}

int main () {
    matrix M1 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 1, 0, 1, 1,
                1, 0, 1, 1, 0, 0, 0,
                0, 1, 0, 1, 1, 1, 0,
                1, 1, 1, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            },7, 7
    );
    matrix M2 = createMatrixFromArray(
            (int[]) {
                0, 1, 0, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 0, 1, 0, 1, 0, 0,
                0, 0, 0, 1, 0, 1, 0,
                0, 1, 1, 0, 1, 0, 1,
                1, 0, 0, 0, 0, 1, 0
            }, 7, 7
    );

    graph G1 = graph_createFromMatrix(M1);
    graph G2 = graph_createFromMatrix(M2);

    printf("\nAll ax simple chains from vertex 2, for G1:\n");
    allMaxSimpleChainsFromVert(G1, 2);
    printf("\nAll ax simple chains from vertex 2, for G2:\n");
    allMaxSimpleChainsFromVert(G2, 2);
}
```

graphs/graphs_main.c

```
All ax simple chains from vertex 2, for G1:
(2, 1, 4, 3, 5, 6, 7)   (2, 1, 4, 3, 6, 5)      (2, 1, 4, 3, 6, 7)      (2, 1, 6, 3, 4) (2, 1, 6, 3, 5) (2, 1, 6, 5, 3,
4)      (2, 1, 6, 7)    (2, 1, 7, 6, 3, 4)      (2, 1, 7, 6, 3, 5)      (2, 1, 7, 6, 5, 3, 4)   (2, 3, 4, 1, 6, 5)
(2, 3, 4, 1, 6, 7)      (2, 3, 4, 1, 7, 6, 5)   (2, 3, 5, 6, 1, 4)      (2, 3, 5, 6, 1, 7)      (2, 3, 5, 6, 7, 1, 4)
(2, 3, 6, 1, 4) (2, 3, 6, 1, 7) (2, 3, 6, 5)    (2, 3, 6, 7, 1, 4)      (2, 4, 1, 6, 3, 5)      (2, 4, 1, 6, 5, 3)
(2, 4, 1, 6, 7) (2, 4, 1, 7, 6, 3, 5)   (2, 4, 1, 7, 6, 5, 3)   (2, 4, 3, 5, 6, 1, 7)   (2, 4, 3, 5, 6, 7, 1)   (2, 4, 3
, 6, 1, 7)      (2, 4, 3, 6, 5) (2, 4, 3, 6, 7, 1)
All ax simple chains from vertex 2, for G2:
(2, 1, 7, 6, 3, 4, 5)   (2, 1, 7, 6, 5, 4, 3)   (2, 6, 3, 4, 5) (2, 6, 5, 4, 3) (2, 6, 7, 1)
Process finished with exit code 0
```

Для графа $G_1$ количество максимальных простых цепей будет слишком велико, чтобы проверить их вручную, но на примере графа $G_2$ мы можем видеть, что они составлены верно.

Примечание: данный алгоритм не рассматривает замкнутые простые цепи.

# Вывод

Граф — пара из множества его вершин и множества связей между ними, то есть двухэлементных подмножеств множества вершин. В ходе данной работы мы анализированы неориентированные графы, не касаясь всего многообразия родственных им объектов: ориентированных графов, мультиграфов, псевдографов, гиперграфов. Граф можно однозначно представить в виде матрицы смежности (элементы матрицы дают информацию о том, смежна ли некоторая пара вершин), матрицы индицентности (элементы матрицы дают информацию о том, инцидентна ли некоторая пара вершины и ребра) и диаграммы (связи между вершинами и ребрами представлены графически). Последовательность попарно перемежающихся вершин и ребер, где соседние элементы инцидентны, задает маршрут в некотором графе. Маршрут можно представить и как последовательность только вершин (или только ребер). Дополнительные свойства маршрутов позволяют выделять среди них цепи, простые цепи, циклы и простые циклы.

В ходе лабораторной работы представили граф тремя разными способами, научились вручную определять, является ли некая последовательность вершин маршрутом, цепью, простой цепью, циклом или простым циклом в конкретном графе, а также написали ряд функций, которые: 1) определяют свойства последовательности вершин относительно конкретного графа, 2) находят все маршруты заданной длины из конкретной вершины, 3) находят все маршруты заданной длины, 4) находят все маршруты заданной длины из конкретной вершины в другую, 5) находят количество маршрутов заданной длины между любой парой вершин, 6) находят максимальные простые цепи из конкретной вершины.