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
тема: «Маршруты»

Выполнил: ст. группы ПВ-212
Степанов Степан Николаевич

Проверили:
Рязанов Юрий Дмитриевич
Бондаренко Татьяна Владимировна

Белгород 2022 г.

Вариант №8

Цель работы: изучить основные понятия теории графов, способы задания графов, научиться программно реализовывать алгоритмы получения и анализа маршрутов в графах.

1. Представить графы G_1 и G_2 (см.Варианты заданий, п.а) матрицей смежности, матрицей инцидентности, диаграммой.

G_1:

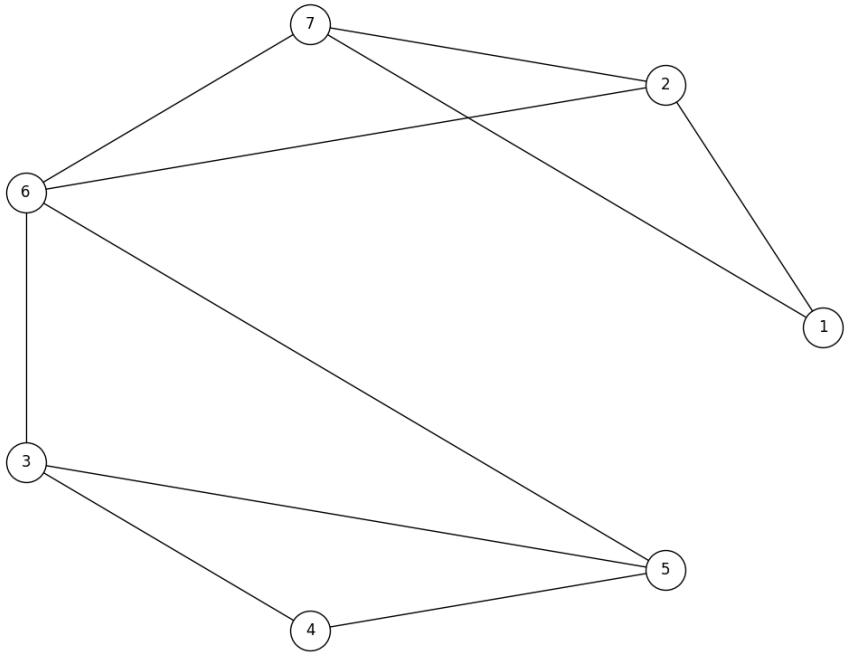
Матрица смежности

Ø	1	Ø	Ø	Ø	Ø	1
1	Ø	Ø	Ø	Ø	1	1
Ø	Ø	Ø	1	1	1	Ø
Ø	Ø	1	Ø	1	Ø	Ø
Ø	Ø	1	1	Ø	1	Ø
Ø	1	1	Ø	1	Ø	1
1	1	Ø	Ø	Ø	1	Ø

Матрица инцидентности

1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1	Ø	1	1	Ø	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	1	1	1	Ø	Ø	Ø
Ø	Ø	Ø	Ø	1	Ø	Ø	1	Ø	Ø
Ø	Ø	Ø	Ø	Ø	1	Ø	1	1	Ø
Ø	Ø	1	Ø	Ø	Ø	1	Ø	1	1
Ø	1	Ø	1	Ø	Ø	Ø	Ø	Ø	1

Диаграмма



G_2:

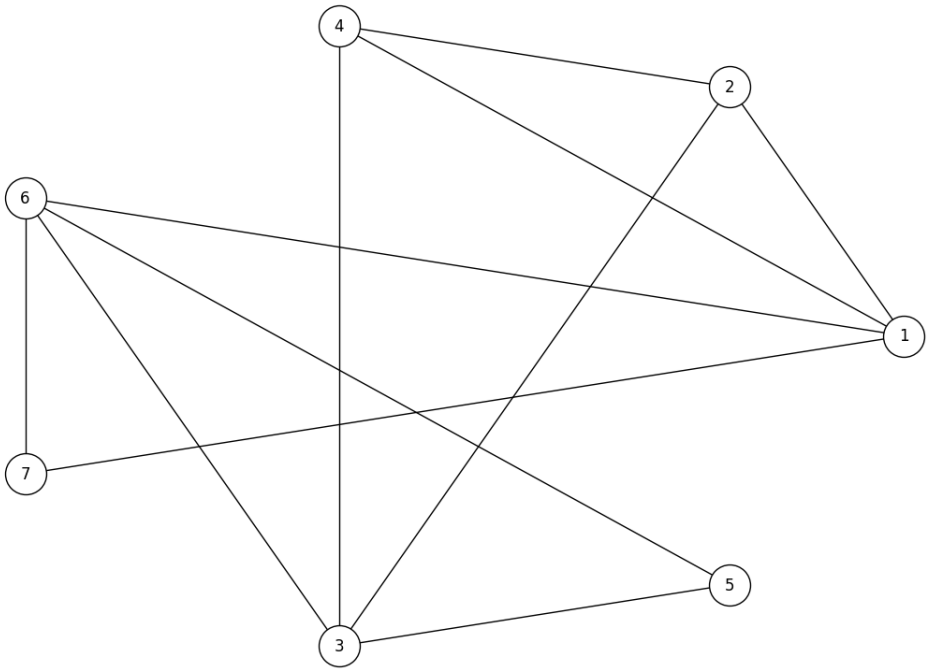
Матрица смежности

Ø	1	Ø	1	Ø	1	1
1	Ø	1	1	Ø	Ø	Ø
Ø	1	Ø	1	1	1	Ø
1	1	1	Ø	Ø	Ø	Ø
Ø	Ø	1	Ø	Ø	1	Ø
1	Ø	1	Ø	1	Ø	1
1	Ø	Ø	Ø	Ø	1	Ø

Матрица инцидентности

1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1	Ø	Ø	Ø	1	1	Ø	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	1	Ø	1	1	1	Ø	Ø
Ø	1	Ø	Ø	Ø	1	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	1	Ø
Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	1	1	1
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	1

Диаграмма



2. 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).

G_1:

	Маршрут	Цепь	Простая цепь	Цикл	Простой цикл
(3, 5, 6, 7, 1, 2)	+	+	+	-	-
(3, 6, 7, 1, 2, 6, 5)	+	+	-	-	-
(4, 3. 6. 5, 4)	+	+	-	+	+
(5, 5, 7, 6, 3, 5)	-	-	-	-	-
(5, 3, 6, 7. 1, 6, 5)	-	-	-	-	-

G_2:

	Маршрут	Цепь	Простая цепь	Цикл	Простой цикл
(3, 5, 6, 7, 1, 2)	+	+	+	-	-
(3, 6, 7, 1, 2, 6, 5)	-	-	-	-	-
(4, 3. 6. 5, 4)	-	-	-	-	-
(5, 5, 7, 6, 3, 5)	-	-	-	-	-
(5, 3, 6, 7. 1, 6, 5)	+	+	-	+	-

3. Написать программу, определяющую, является ли заданная последовательность 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).

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <map>

using adjacencyMatrixRow = std::vector<bool>;
using adjacencyMatrix = std::vector<adjacencyMatrixRow>;

bool graph_isRoute(const adjacencyMatrix &m,
                   const std::vector<int> &verticesSequence) {
    for (int i = 1; i < verticesSequence.size(); ++i)
        if (!m[verticesSequence[i - 1] - 1][verticesSequence[i] - 1])
            return false;

    return true;
}

bool graph_isChain(const adjacencyMatrix &m,
                   const std::vector<int> &verticesSequence) {
    if (!graph_isRoute(m, verticesSequence))
        return false;

    std::map<int, int> edges;

    for (int i = 1; i < verticesSequence.size(); ++i) {
        if (edges[verticesSequence[i]] == verticesSequence[i - 1])
            return false;

        edges[verticesSequence[i - 1]] = verticesSequence[i];
    }

    return true;
}

bool graph_isSimpleChain(const adjacencyMatrix &m,
                         const std::vector<int> &verticesSequence) {
    if (!graph_isChain(m, verticesSequence))
        return false;

    std::set<int> uniqueVertices;

    for (const auto &vertex: verticesSequence)
        uniqueVertices.insert(vertex);

    return uniqueVertices.size() == verticesSequence.size();
}

bool graph_isCycle(const adjacencyMatrix &m,
                   const std::vector<int> &verticesSequence) {
    if (!graph_isChain(m, verticesSequence))
        return false;

    return verticesSequence.front() == verticesSequence.back();
}
```

```
bool graph_isSimpleCycle(const adjacencyMatrix &m,
                         const std::vector<int> &verticesSequence) {
    if (!graph_isCycle(m, verticesSequence))
        return false;

    std::set<int> uniqueVertices;

    for (const auto &vertex: verticesSequence)
        uniqueVertices.insert(vertex);

    return uniqueVertices.size() + 1 == verticesSequence.size();
}
1)
int main() {
    std::vector<std::vector<int>> verticesSequences = {
                                                       {3, 5, 6, 7, 1, 2},
                                                       {3, 6, 7, 1, 2, 6, 5},
                                                       {4, 3, 6, 5, 4},
                                                       {5, 5, 7, 6, 3, 5},
                                                       {5, 3, 6, 7, 1, 6, 5}};

    adjacencyMatrix m1 = {{0, 1, 0, 0, 0, 0, 1},
                          {1, 0, 0, 0, 0, 1, 1},
                          {0, 0, 0, 1, 1, 1, 0},
                          {0, 0, 1, 0, 1, 0, 0},
                          {0, 0, 1, 1, 0, 1, 0},
                          {0, 1, 1, 0, 1, 0, 1},
                          {1, 1, 0, 0, 0, 1, 0}};

    adjacencyMatrix m2 = {{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}};


    bool (*functions[])(const adjacencyMatrix &,
                        const std::vector<int> &) = {graph_isRoute,
                                                     graph_isChain,
                                                     graph_isSimpleChain,
                                                     graph_isCycle,
                                                     graph_isSimpleCycle};
    std::vector<std::string> names = {"Route",
                                      "Chain",
                                      "Simple Chain",
                                      "Cycle",
                                      "Simple Cycle"};

    std::vector<adjacencyMatrix> matrices = {m1, m2};
```

```
for (const auto &matrix: matrices) {
        for (auto & name : names)
            std::cout << ';' << name;
        std::cout << ";\n";

        for (const auto &seq: verticesSequences) {
            std::cout << "( ";
            for (const auto &vertex: seq) {
                std::cout << vertex << ' ';
            }
            std::cout << ");";

            for (int i = 0; i < names.size(); ++i) {
                std::cout << functions[i](matrix, seq) << ';';
            }

            std::cout << '\n';
        }

        std::cout << '\n';
    }

    return 0;

}
```

```
;Route;Chain;Simple Chain;Cycle;Simple Cycle;
( 3 5 6 7 1 2 );1;1;1;0;0;
( 3 6 7 1 2 6 5 );1;1;0;0;0;
( 4 3 6 5 4 );1;1;0;1;1;
( 5 5 7 6 3 5 );0;0;0;0;0;
( 5 3 6 7 1 6 5 );0;0;0;0;0;

;Route;Chain;Simple Chain;Cycle;Simple Cycle;
( 3 5 6 7 1 2 );1;1;1;0;0;
( 3 6 7 1 2 6 5 );0;0;0;0;0;
( 4 3 6 5 4 );0;0;0;0;0;
( 5 5 7 6 3 5 );0;0;0;0;0;
( 5 3 6 7 1 6 5 );1;1;0;1;0;
```

Получим таблицы, которые совпадают с полученными ранее:

A	B	C	D	E	F
	Route	Chain	Simple Chain	Cycle	Simple Cycle
(356712)	1	1	1	0	0
(3671265)	1	1	0	0	0
(43654)	1	1	0	1	1
(557635)	0	0	0	0	0
(5367165)	0	0	0	0	0
	Route	Chain	Simple Chain	Cycle	Simple Cycle
(356712)	1	1	1	0	0
(3671265)	0	0	0	0	0
(43654)	0	0	0	0	0
(557635)	0	0	0	0	0
(5367165)	1	1	0	1	0

4. 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).

```
std::set<int> graph_getAdjacentVertices(const adjacencyMatrix &m,
                                        const int vertex) {
    std::set<int> res;

    for (int i = 0; i < m.size(); ++i)
        if (m[vertex - 1][i])
            res.insert(i + 1);


    return res;
}

std::set<std::set<int>>
graph_getSetsOfAdjacentVertices(const adjacencyMatrix &m,
                                const std::set<int> &vertices) {
    std::set<std::set<int>> res;

    for (const auto &vertex: vertices)
        res.insert(graph_getAdjacentVertices(m, vertex));

    return res;
}

void graph__getRoutes(const size_t l,
                      std::vector<int> &currRoute,
                      std::set<std::vector<int>> &routes,
                      const adjacencyMatrix &m) {
    auto adjacentVertices = graph_getAdjacentVertices(m, currRoute.back());

    for (const auto &vertex: adjacentVertices) {
        currRoute.push_back(vertex);

        if (currRoute.size() == l + 1)
            routes.insert(currRoute);
        else
            graph__getRoutes(l, currRoute, routes, m);

        currRoute.pop_back();
    }
}

std::set<std::vector<int>> graph_getRoutes(const adjacencyMatrix &m,
                                           const int vertex,
                                           const size_t length) {
    if (0 >= vertex && vertex >= m.size())
        throw std::runtime_error("There is no such vertex in the graph");

    std::set<std::vector<int>> routes;
    std::vector<int> W1 = {vertex};

    graph__getRoutes(length, W1, routes, m);

    return routes;
}
```

```cpp
int main() {
    adjacencyMatrix m1 = {{0, 1, 0, 0, 0, 0, 1},
                          {1, 0, 0, 0, 0, 1, 1},
                          {0, 0, 0, 1, 1, 1, 0},
                          {0, 0, 1, 0, 1, 0, 0},
                          {0, 0, 1, 1, 0, 1, 0},
                          {0, 1, 1, 0, 1, 0, 1},
                          {1, 1, 0, 0, 0, 1, 0}};

    adjacencyMatrix m2 = {{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}};

    std::vector<adjacencyMatrix> matrices = {m1, m2};

    int length;
    std::cin >> length;

    for (const auto &m: matrices) {
        for (int i = 1; i <= m.size(); ++i) {
            auto res = graph_getRoutes(m, i, length);

            for (auto &set: res) {
                std::cout << "{ ";
                for (auto &elem: set) {
                    std::cout << elem << ' ';
                }
                std::cout << "}\n";
            }
        }

        std::cout << "\n";
    }

    return 0;

}
```

```
1
{ 1 2 }
{ 1 7 }
{ 2 1 }
{ 2 6 }
{ 2 7 }
{ 3 4 }
{ 3 5 }
{ 3 6 }
{ 4 3 }
{ 4 5 }
{ 5 3 }
{ 5 4 }
{ 5 6 }
{ 6 2 }
{ 6 3 }
{ 6 5 }
{ 6 7 }
{ 7 1 }
{ 7 2 }
{ 7 6 }
```

```
{ 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 }
```

5. 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).

```
 void graph__getRoutesAmount(const int v,
                             const size_t l,
                             std::vector<int> &currRoute,
                             std::vector<std::vector<int>> &R,
                             const adjacencyMatrix &m) {
    auto adjacentVertices = graph_getAdjacentVertices(m, currRoute.back());

    for (const auto &vertex: adjacentVertices) {
        currRoute.push_back(vertex);

        if (currRoute.size() == l + 1)
            R[v][currRoute.back() - 1]++;
        else
            graph__getRoutesAmount(v, l, currRoute, R, m);

        currRoute.pop_back();
    }
}

std::vector<std::vector<int>> graph_getRoutesAmount(const adjacencyMatrix &m,
                                                    const size_t length) {
    auto size = m.size();
    std::vector<std::vector<int>> R(size,
                                    std::vector<int>(size, 0));

    for (int i = 0; i < size; ++i) {
        std::vector<int> W1 = {i + 1};
        graph__getRoutesAmount(i, length, W1, R, m);
    }

    return R;
}
```

Реализация через умножение матриц:

```
template<typename T>
std::vector<std::vector<T>>
multiplyMatrices(const std::vector<std::vector<T>> &m1,
                 const std::vector<std::vector<T>> &m2) {
    std::vector<std::vector<T>> res(m1.size(),
                                    std::vector<T>(m2[0].size(), 0));

    for (int i = 0; i < m1.size(); ++i)
        for (int j = 0; j < m2[0].size(); ++j)
            for (int k = 0; k < m1[0].size(); ++k)
                res[i][j] += m1[i][k] * m2[k][j];

    return res;
}
```

```cpp
template<typename T>
std::vector<std::vector<T>>
getIdenticalMatrix(const size_t size) {
    std::vector<std::vector<T>> res(size,
                                    std::vector<T>(size, 0));

    for (int i = 0; i < size; ++i)
        res[i][i] = 1;

    return res;
}

template<typename T>
std::vector<std::vector<T>>
powMatrix(std::vector<std::vector<T>> matrix,
          size_t power) {

    std::vector<std::vector<T>> res =
            getIdenticalMatrix<T>(matrix.size());
    std::vector<std::vector<T>> currPow0f2 = matrix;

    while (power) {
        if (power & 1)
            res = multiplyMatrices(res, currPow0f2);

        currPow0f2 = multiplyMatrices(currPow0f2, currPow0f2);
        power >>= 1;
    }

    return res;
}

std::vector<std::vector<int>>
graph_getRoutesAmountByAdjacencyMatrix(const adjacencyMatrix &m,
                                       const size_t length) {
    std::vector<std::vector<int>> R(m.size(),
                                    std::vector<int>(m.size()));
    for (int i = 0; i < m.size(); ++i)
        for (int j = 0; j < m.size(); ++j)
            R[i][j] = m[i][j];

    return powMatrix(R, length);
}

int main() {
    adjacencyMatrix m1 = {{0, 1, 0, 0, 0, 0, 1},
                          {1, 0, 0, 0, 0, 1, 1},
                          {0, 0, 0, 1, 1, 1, 0},
                          {0, 0, 1, 0, 1, 0, 0},
                          {0, 0, 1, 1, 0, 1, 0},
                          {0, 1, 1, 0, 1, 0, 1},
                          {1, 1, 0, 0, 0, 1, 0}};

    adjacencyMatrix m2 = {{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}};
```

```
    std::vector<adjacencyMatrix> matrices = {m1, m2};

    int length;
    std::cin >> length;

    for (const auto &m: matrices) {

        auto res = graph_getRoutesAmount(m, length);

        for (auto &set: res) {
            for (auto &elem: set) {
                std::cout << elem << ' ';
            }
            std::cout << "\n";
        }

        std::cout << "\n";
    }

    return 0;

}
```

```
2
2 1 0 0 0 2 1
1 3 1 0 1 1 2
0 1 3 1 2 1 1
0 0 1 2 1 2 0
0 1 2 1 3 1 1
2 1 1 2 1 4 1
1 2 1 0 1 1 3

4 1 3 1 1 1 1
1 3 1 2 1 2 1
3 1 4 1 1 1 1
1 2 1 3 1 2 1
1 1 1 1 2 1 1
1 2 1 2 1 4 1
1 1 1 1 1 1 2
```

6. 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).

```
void graph__getRoutesBetweenVertices(const size_t l,
                                     const int vertexEnd,
                                     std::vector<int> &currRoute,
                                     std::set<std::vector<int>> &routes,
                                     const adjacencyMatrix &m) {
    auto adjacentVertices = graph_getAdjacentVertices(m, currRoute.back());
```

```
    for (const auto &vertex: adjacentVertices) {
        currRoute.push_back(vertex);

        if (currRoute.size() == l + 1) {
            if (currRoute.back() == vertexEnd)
                routes.insert(currRoute);
        } else
            graph__getRoutesBetweenVertices(l, vertexEnd,
                                            currRoute, routes, m);

        currRoute.pop_back();
    }
}

std::set<std::vector<int>>
graph_getRoutesBetweenVertices(const adjacencyMatrix &m,
                               const int vertex1,
                               const int vertex2,
                               const size_t length) {
    if (0 >= vertex1 && vertex1 >= m.size() ||
        0 >= vertex2 && vertex2 >= m.size())
        throw std::runtime_error("There is no such vertex in the graph");

    std::set<std::vector<int>> routes;
    std::vector<int> W1 = {vertex1};

    graph__getRoutesBetweenVertices(length, vertex2, W1, routes, m);

    return routes;
}

int main() {
    adjacencyMatrix m1 = {{0, 1, 0, 0, 0, 0, 1},
                          {1, 0, 0, 0, 0, 1, 1},
                          {0, 0, 0, 1, 1, 1, 0},
                          {0, 0, 1, 0, 1, 0, 0},
                          {0, 0, 1, 1, 0, 1, 0},
                          {0, 1, 1, 0, 1, 0, 1},
                          {1, 1, 0, 0, 0, 1, 0}};

    adjacencyMatrix m2 = {{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}};

    std::vector<adjacencyMatrix> matrices = {m1, m2};

    int length, vertex1, vertex2;
    std::cin >> length >> vertex1 >> vertex2;
```

```
for (const auto &m: matrices) {

        auto res =
                graph_getRoutesBetweenVertices(m, vertex1, vertex2, length);

        for (auto &set: res) {
            std::cout << "{ ";
            for (auto &elem: set) {
                std::cout << elem << ' ';
            }
            std::cout << "}\n";
        }
        std::cout << "\n";
    }

    return 0;
}
```

```
2
1 1
{ 1 2 1 }
{ 1 7 1 }

{ 1 2 1 }
{ 1 4 1 }
{ 1 6 1 }
{ 1 7 1 }
```

```
3
1 3
{ 1 2 6 3 }
{ 1 7 6 3 }

{ 1 2 4 3 }
{ 1 4 2 3 }
{ 1 6 5 3 }
{ 1 7 6 3 }
```

7. Написать программу, получающую все простые максимальные цепи, выходящие из заданной вершины графа. Использовать программу для получения всех простых максимальных цепей, выходящих из заданной вершины в графах G_1 и G_2 (см.Варианты заданий, п.а).

```
void graph__getMaxSimpleChain(std::vector<int> &currRoute,
                              std::set<int> &V,
                              std::set<std::vector<int>> &routes,
                              const adjacencyMatrix &m) {
    auto adjacentVertices = graph_getAdjacentVertices(m, currRoute.back());

    std::set<int> remainingVertices;
    std::set_difference(adjacentVertices.begin(), adjacentVertices.end(),
                        V.begin(), V.end(),
                        std::inserter(remainingVertices,
                                      remainingVertices.begin()));
```

```
    for (const auto &vertex: remainingVertices) {
        currRoute.push_back(vertex);

        auto newAdjacentVertices = graph_getAdjacentVertices(m,
                                                             currRoute.back());

        if (std::includes(V.begin(), V.end(),
                          newAdjacentVertices.begin(),
                          newAdjacentVertices.end()))
            routes.insert(currRoute);
        else {
            V.insert(vertex);
            graph__getMaxSimpleChain(currRoute, V, routes, m);
            V.erase(vertex);
        }

        currRoute.pop_back();
    }
}


std::set<std::vector<int>> graph_getMaxSimpleChain(const adjacencyMatrix &m,
                                                   const int vertex) {
    if (0 >= vertex && vertex >= m.size())
        throw std::runtime_error("There is no such vertex in the graph");

    std::set<std::vector<int>> routes;
    std::vector<int> W1 = {vertex};
    std::set<int> V = {vertex};

    graph__getMaxSimpleChain(W1, V, routes, m);

    return routes;
}

int main() {
    adjacencyMatrix m1 = {{0, 1, 0, 0, 0, 0, 1},
                          {1, 0, 0, 0, 0, 1, 1},
                          {0, 0, 0, 1, 1, 1, 0},
                          {0, 0, 1, 0, 1, 0, 0},
                          {0, 0, 1, 1, 0, 1, 0},
                          {0, 1, 1, 0, 1, 0, 1},
                          {1, 1, 0, 0, 0, 1, 0}};

    adjacencyMatrix m2 = {{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}};

    std::vector<adjacencyMatrix> matrices = {m1, m2};
```

```
int vertex;
    std::cin >> vertex;

    for (const auto &m: matrices) {

        auto res =
                graph_getMaxSimpleChain(m, vertex);

        for (auto &set: res) {
            std::cout << "{ ";
            for (auto &elem: set) {
                std::cout << elem << ' ';
            }
            std::cout << "}\n";
        }
        std::cout << "\n";
    }

    return 0;
}
```

```
1
{ 1 2 6 3 4 5 }
{ 1 2 6 3 5 4 }
{ 1 2 6 5 3 4 }
{ 1 2 6 5 4 3 }
{ 1 2 6 7 }
{ 1 2 7 6 3 4 5 }
{ 1 2 7 6 3 5 4 }
{ 1 2 7 6 5 3 4 }
{ 1 2 7 6 5 4 3 }
{ 1 7 2 6 3 4 5 }
{ 1 7 2 6 3 5 4 }
{ 1 7 2 6 5 3 4 }
{ 1 7 2 6 5 4 3 }
{ 1 7 6 2 }
{ 1 7 6 3 4 5 }
{ 1 7 6 3 5 4 }
{ 1 7 6 5 3 4 }
{ 1 7 6 5 4 3 }
```

```
{ 1 2 3 4 }
{ 1 2 3 5 6 7 }
{ 1 2 3 6 5 }
{ 1 2 3 6 7 }
{ 1 2 4 3 5 6 7 }
{ 1 2 4 3 6 5 }
{ 1 2 4 3 6 7 }
{ 1 4 2 3 5 6 7 }
{ 1 4 2 3 6 5 }
{ 1 4 2 3 6 7 }
{ 1 4 3 2 }
{ 1 4 3 5 6 7 }
{ 1 4 3 6 5 }
{ 1 4 3 6 7 }
{ 1 6 3 2 4 }
{ 1 6 3 4 2 }
{ 1 6 3 5 }
{ 1 6 5 3 2 4 }
{ 1 6 5 3 4 2 }
{ 1 6 7 }
{ 1 7 6 3 2 4 }
{ 1 7 6 3 4 2 }
{ 1 7 6 3 5 }
{ 1 7 6 5 3 2 4 }
{ 1 7 6 5 3 4 2 }
```

Вывод: в ходе работы были изучены основные понятия теории графов, способы задания графов, были программно реализовованы алгоритмы получения и анализа маршрутов в графах.