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-211
Чувилко Илья Романович

Проверили:
Рязанов Юрий Дмитриевич
Бондаренко Татьяна Владимировна

Белгород 2022 г.

Вариант №9

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

№1. Представить графы G1 и G2 (см.Варианты заданий, п.а) матрицей смежности, матрицей инцидентности, диаграммой

G1:

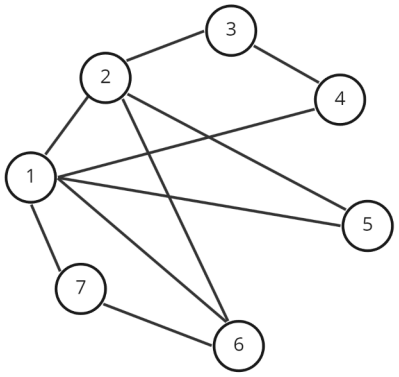
Матрица смежности

	1	2	3	4	5	6	7
1	0	1	0	1	0	1	1
2	1	0	0	0	1	1	0
3	0	0	0	1	0	0	0
4	1	0	1	0	0	0	0
5	1	1	0	0	0	0	0
6	1	1	0	0	0	0	1
7	1	0	0	0	0	1	0

Матрица инцидентности

	1	2	3	4	5	6	7
1	1	1	1	1	1		
2	1						1
3							
4		1					
5			1				
6				1		1	1
7					1	1	

Диаграмма



G2:

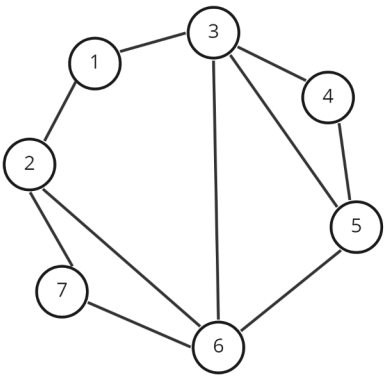
Матрица смежности

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	1	0	0	0	0	1	1
3	1	0	0	1	1	1	0
4	0	0	1	0	1	1	0
5	0	0	1	1	0	1	0
6	0	1	1	0	1	0	1
7	0	1	0	0	0	1	0

Матрица инцидентности

	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0
3	0	0	0	0	1	1	1	0	0	0
4	0	0	0	0	0	0	1	1	0	0
5	0	0	0	0	0	1	0	1	0	1
6	0	0	0	1	1	0	0	0	1	1
7	0	1	1	0	0	0	0	0	1	0

Диаграмма



№2. Определить, являются ли последовательности вершин (см. Варианты заданий, п.б) маршрутом, цепью, простой цепью, циклом, простым циклом в графах G1 и G2 (см.Варианты заданий, п.а).

G1:

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

G2:

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

№3. 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).

Код программы:

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

using namespace std;

using GraphRow = vector<bool>;
using Graph = vector<GraphRow>;

// Возвращает true, если последовательности вершин являются маршрутом
bool graphIsRoute(const Graph &g, const vector<int> &vertices) {
  for (int i = 1; i < vertices.size(); ++i)
    if (!g[vertices[i - 1] - 1][vertices[i] - 1])
      return false;
  return true;
```

```
}

// Возвращает true, если последовательности вершин являются цепью
bool graphIsChain(const Graph &m, const vector<int> &vertices) {
  if (!graphIsRoute(m, vertices))
    return false;
  map<int, int> edges;
  for (int i = 1; i < vertices.size(); ++i) {
    if (edges[vertices[i]] == vertices[i - 1])
      return false;
    edges[vertices[i - 1]] = vertices[i];
  }
  return true;
}

// Возвращает true, если последовательности вершин являются простой цепью
bool graphIsSimpleChain(const Graph &m,
                        const vector<int> &verticesSequence) {
  if (!graphIsChain(m, verticesSequence))
    return false;
  set<int> uniqueVertices;
  for (const auto &vertex: verticesSequence)
    uniqueVertices.insert(vertex);
  return uniqueVertices.size() == verticesSequence.size();
}

// Возвращает true, если последовательности вершин являются циклом
bool graphIsCycle(const Graph &m,
                  const vector<int> &verticesSequence) {
  if (!graphIsChain(m, verticesSequence))
    return false;
  return verticesSequence.front() == verticesSequence.back();
}

// Возвращает true, если последовательности вершин являются простым циклом
bool graphIsSimpleCycle(const Graph &m,
                        const vector<int> &verticesSequence) {
  if (!graphIsCycle(m, verticesSequence))
    return false;
  set<int> uniqueVertices;
  for (const auto &vertex: verticesSequence)
    uniqueVertices.insert(vertex);
  return uniqueVertices.size() + 1 == verticesSequence.size();
}



int main() {
  vector<vector<int>> verticesSequences = {
                        {1, 2, 6, 7},
          {2, 6, 1, 2, 3, 4},
          {7, 6, 2, 1, 7},
          {6, 2, 1, 7, 2, 6},
          {7, 6, 2, 1, 6, 2}};
  Graph m1 = {{0, 1, 0, 1, 0, 1, 1},
              {1, 0, 0, 0, 1, 1, 0},
              {0, 0, 0, 1, 0, 0, 0},
              {1, 0, 1, 0, 0, 0, 0},
              {1, 1, 0, 0, 0, 0, 0},
              {1, 1, 0, 0, 0, 0, 1},
              {1, 0, 0, 0, 0, 1, 0}};
```

```
 Graph m2 = {{0, 1, 1, 0, 0, 0, 0},
             {1, 0, 0, 0, 0, 1, 1},
             {1, 0, 0, 1, 1, 1, 0},
             {0, 0, 1, 0, 1, 1, 0},
             {0, 0, 1, 1, 0, 1, 0},
             {0, 1, 1, 0, 1, 0, 1},
             {0, 1, 0, 0, 0, 1, 0}};

 bool (*functions[])(const Graph &, const vector<int> &) ={
         graphIsRoute,
         graphIsChain,
         graphIsSimpleChain,
         graphIsCycle,
         graphIsSimpleCycle
 };
 vector<string> names = {"Route",
                         "Chain",
                         "Simple Chain",
                         "Cycle",
                         "Simple Cycle"};

 vector<Graph> matrices = {m1, m2};
 for (const auto &matrix: matrices) {
   for (auto &name: names)
     cout << name << "; ";
   cout << "\n";
   for (const auto &seq: verticesSequences) {
     cout << "( ";
     for (const auto &vertex: seq) {
       cout << vertex << ' ';
     }
     cout << ")\t\t";
     for (int i = 0; i < names.size(); ++i) {
       cout << functions[i](matrix, seq) << "; ";
     }
     cout << '\n';
   }
   cout << '\n';
 }
 return 0;
}
```

Результат работы программы:

```
C:\BGTU\BGTU\DisMat\lab_4_1\Code\cmake-build-debug\Code.exe
Route; Chain; Simple Chain; Cycle; Simple Cycle;
( 1 2 6 7 )             1; 1; 1; 0; 0;
( 2 6 1 2 3 4 )         0; 0; 0; 0; 0;
( 7 6 2 1 7 )           1; 1; 0; 1; 1;
( 6 2 1 7 2 6 )         0; 0; 0; 0; 0;
( 7 6 2 1 6 2 )         1; 1; 0; 0; 0;

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


Process finished with exit code 0
```

Результат работы программы совпал с полученными выше результатами

№4. Написать программу, получающую все маршруты заданной длины, выходящие из заданной вершины. Использовать программу для получения всех маршрутов заданной длины в графах G1 и G2 (см.Варианты заданий, п.а).

Код программы:

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


// Возвращает смежный граф
set<int> graphGetAdjacent(const Graph &m, const int vertex) {
  set<int> res;
  for (int i = 0; i < m.size(); ++i)
    if (m[vertex - 1][i])
      res.insert(i + 1);
  return res;
}

set<set<int>> graphGetSetsOfAdjacentVertices(const Graph &m, const set<int> &vertices) {
  set<set<int>> res;
  for (const auto &vertex: vertices)
    res.insert(graphGetAdjacent(m, vertex));
  return res;
}

void graphGetRoutes_(const size_t l, vector<int> &currRoute,
                     set<vector<int>> &routes, const Graph &m) {
  auto adjacentVertices = graphGetAdjacent(m, currRoute.back());
  for (const auto &vertex: adjacentVertices) {
    currRoute.push_back(vertex);
    if (currRoute.size() == l + 1)
      routes.insert(currRoute);
    else
      graphGetRoutes_(l, currRoute, routes, m);
    currRoute.pop_back();
  }
}

set<vector<int>> graphGetRoutes(const Graph &m, const int vertex, const size_t length) {
  if (0 >= vertex && vertex >= m.size())
    throw runtime_error("There is no such vertex in the graph");
  set<vector<int>> routes;
  vector<int> W1 = {vertex};
  graphGetRoutes_(length, W1, routes, m);
  return routes;
}

void outputRoutes(Graph &m, int length) {
  cout << "{";
  for (int i = 1; i <= m.size(); ++i) {
    auto res = graphGetRoutes(m, i, length);
    for (auto &set: res) {
      cout << "{";
      for (auto &elem: set) {
        cout << elem << ", ";
      }
      cout << "\b\b}, ";
    }
```

```
    }
    cout << "}\n\n";
}


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

    Graph m2 = {{0, 1, 1, 0, 0, 0, 0},
                {1, 0, 0, 0, 0, 1, 1},
                {1, 0, 0, 1, 1, 1, 0},
                {0, 0, 1, 0, 1, 1, 0},
                {0, 0, 1, 1, 0, 1, 0},
                {0, 1, 1, 0, 1, 0, 1},
                {0, 1, 0, 0, 0, 1, 0}};

    int length;
    cin >> length;
    cout << "G1 = ";
    outputRoutes(m1, length);

    cout << "G2 = ";
    outputRoutes(m2, length);

    return 0;
}
```

Результат работы программы:

```
C:\BGTU\BGTU\DisMat\lab_4_1\Code\cmake-build-debug\Code.exe
1
G1 = {{1, 2}, {1, 4}, {1, 6}, {1, 7}, {2, 1}, {2, 5}, {2, 6}, {3, 4}, {4, 1}, {4, 3}, {5, 1}, {5, 2}, {6, 1}, {6, 2}, {6
, 7}, {7, 1}, {7, 6}, }

G2 = {{1, 2}, {1, 3}, {2, 1}, {2, 6}, {2, 7}, {3, 1}, {3, 4}, {3, 5}, {3, 6}, {4, 3}, {4, 5}, {4, 6}, {5, 3}, {5, 4}, {5
, 6}, {6, 2}, {6, 3}, {6, 5}, {6, 7}, {7, 2}, {7, 6}, }


Process finished with exit code 0
```

№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 G1 и G2 (см.Варианты заданий, п.а).

Код программы:

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

void graphGetRoutesAmount_(const int v, const size_t l, vector<int> &currRoute,
                           vector<vector<int>> &R, const Graph &m) {
  auto adjacentVertices = graphGetAdjacent(m, currRoute.back());
  for (const auto &vertex: adjacentVertices) {
    currRoute.push_back(vertex);
    if (currRoute.size() == l + 1)
      R[v][currRoute.back() - 1]++;
    else
      graphGetRoutesAmount_(v, l, currRoute, R, m);
    currRoute.pop_back();
  }
}

vector<vector<int>> graphGetRoutesAmount(const Graph &m, const size_t length) {
  auto size = m.size();
  vector<vector<int>> R(size, vector<int>(size, 0));
  for (int i = 0; i < size; ++i) {
    vector<int> W1 = {i + 1};
    graphGetRoutesAmount_(i, length, W1, R, m);
  }
  return R;
}

template<typename T>
vector<vector<T>> multiplyMatrices(const vector<vector<T>> &m1, const vector<vector<T>> &m2) {
  vector<vector<T>> res(m1.size(), 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;
}

template<typename T>
vector<vector<T>> getIdenticalMatrix(const size_t size) {
  vector<vector<T>> res(size, vector<T>(size, 0));
  for (int i = 0; i < size; ++i)
    res[i][i] = 1;
  return res;
}

template<typename T>
vector<vector<T>> powMatrix(vector<vector<T>> matrix, size_t power) {
  vector<vector<T>> res = getIdenticalMatrix<T>(matrix.size());
  vector<vector<T>> currPowOf2 = matrix;
  while (power) {
    if (power & 1)
      res = multiplyMatrices(res, currPowOf2);
    currPowOf2 = multiplyMatrices(currPowOf2, currPowOf2);
```

```
    power >>= 1;
  }
  return res;
}

vector<vector<int>> graphGetRoutesAmountByAdjacencyMatrix
        (const Graph &m, const size_t length) {
  vector<vector<int>> R(m.size(), 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() {
  Graph m1 = {{0, 1, 0, 1, 0, 1, 1},
              {1, 0, 0, 0, 1, 1, 0},
              {0, 0, 0, 1, 0, 0, 0},
              {1, 0, 1, 0, 0, 0, 0},
              {1, 1, 0, 0, 0, 0, 0},
              {1, 1, 0, 0, 0, 0, 1},
              {1, 0, 0, 0, 0, 1, 0}};

  Graph m2 = {{0, 1, 1, 0, 0, 0, 0},
              {1, 0, 0, 0, 0, 1, 1},
              {1, 0, 0, 1, 1, 1, 0},
              {0, 0, 1, 0, 1, 1, 0},
              {0, 0, 1, 1, 0, 1, 0},
              {0, 1, 1, 0, 1, 0, 1},
              {0, 1, 0, 0, 0, 1, 0}};

  vector<Graph> matrices = {m1, m2};
  int length;
  cin >> length;
  for (const auto &m: matrices) {
    auto res = graphGetRoutesAmount(m, length);
    for (auto &set: res) {
      for (auto &elem: set) {
        cout << elem << ' ';
      }
      cout << "\n";
    }
    cout << "\n";
  }

  return 0;
}
```

Результат работы программы:

```
C:\BGTU\BGTU\DisMat\lab_4_1\Code\cmake-build-debug\Code.exe
2
4 1 1 0 1 2 1
2 3 0 1 0 1 2
1 0 1 0 0 0 0
0 1 0 2 0 1 1
1 1 0 1 1 2 1
2 1 0 1 1 3 1
1 2 0 1 0 1 2

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


Process finished with exit code 0
```

№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 G1 и G2 (см.Варианты заданий, п.а).

Код программы:

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

void graphGetRoutesBetweenVertices_(const size_t l,
                                    const int vertexEnd,
                                    vector<int> &currRoute,
                                    set<vector<int>> &routes,
                                    const Graph &m) {
  auto adjacentVertices = graphGetAdjacent(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
      graphGetRoutesBetweenVertices_(l, vertexEnd, currRoute, routes, m);
    currRoute.pop_back();
  }
}

set<vector<int>> graphGetRoutesBetweenVertices(const Graph &m,
                                               const int vertex1,
                                               const int vertex2,
                                               const size_t length) {
  if (0 >= vertex1 && vertex1 >= m.size() ||
```

```
    0 >= vertex2 && vertex2 >= m.size())
   throw runtime_error("There is no such vertex in the graph");
  set<vector<int>> routes;
  vector<int> W1 = {vertex1};
  graphGetRoutesBetweenVertices_(length, vertex2, W1, routes, m);
  return routes;
}

void outputRoutesBetweenVertices(Graph &m, int length, int from, int to) {
  auto res = graphGetRoutesBetweenVertices(m, from, to, length);
  for (auto &set: res) {
    cout << "{ ";
    for (auto &elem: set) {
      cout << elem << ' ';
    }
    cout << "}\n";
  }
  cout << "\n";
}


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

  Graph m2 = {{0, 1, 1, 0, 0, 0, 0},
              {1, 0, 0, 0, 0, 1, 1},
              {1, 0, 0, 1, 1, 1, 0},
              {0, 0, 1, 0, 1, 1, 0},
              {0, 0, 1, 1, 0, 1, 0},
              {0, 1, 1, 0, 1, 0, 1},
              {0, 1, 0, 0, 0, 1, 0}};

  int length, from, to;
  cin >> length >> from >> to;

  cout << "G1:\n";
  outputRoutesBetweenVertices(m1, length, from, to);

  cout << "G2:\n";
  outputRoutesBetweenVertices(m2, length, from, to);

  return 0;
}
```

Результат работы программы:

```
C:\BGTU\BGTU\DisMat\lab_4_1\Code\cmake-build-debug\Code.exe
2
1 1
G1:
{ 1 2 1 }
{ 1 4 1 }
{ 1 6 1 }
{ 1 7 1 }

G2:
{ 1 2 1 }
{ 1 3 1 }


Process finished with exit code 0
```

№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 G1 и G2 (см.Варианты заданий, п.а).

Код программы:

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

void graphGetMaxSimpleChain_(vector<int> &currRoute, set<int> &V,
                             set<vector<int>> &routes, const Graph &m) {
  auto adjacentVertices = graphGetAdjacent(m, currRoute.back());
  set<int> remainingVertices;
  set_difference(adjacentVertices.begin(), adjacentVertices.end(),
                 V.begin(), V.end(),
                 inserter(remainingVertices, remainingVertices.begin()));

  for (const auto &vertex: remainingVertices) {
    currRoute.push_back(vertex);
    auto newAdjacentVertices = graphGetAdjacent(m,
                                                currRoute.back());
    if (includes(V.begin(), V.end(),
                 newAdjacentVertices.begin(),
                 newAdjacentVertices.end()))
      routes.insert(currRoute);
    else {
      V.insert(vertex);
      graphGetMaxSimpleChain_(currRoute, V, routes, m);
      V.erase(vertex);
    }
    currRoute.pop_back();
  }
}

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

```
  set<vector<int>> routes;
  vector<int> W1 = {vertex};
  set<int> V = {vertex};
  graphGetMaxSimpleChain_(W1, V, routes, m);
  return routes;
}

void outputMaxSimpleChain(Graph &m, int vertex) {
  auto res = graph_getMaxSimpleChain(m, vertex);
  for (auto &set: res) {
    cout << "{ ";
    for (auto &elem: set) {
      cout << elem << ' ';
    }
    cout << "}\n";
  }
  cout << "\n";
}


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

  Graph m2 = {{0, 1, 1, 0, 0, 0, 0},
              {1, 0, 0, 0, 0, 1, 1},
              {1, 0, 0, 1, 1, 1, 0},
              {0, 0, 1, 0, 1, 1, 0},
              {0, 0, 1, 1, 0, 1, 0},
              {0, 1, 1, 0, 1, 0, 1},
              {0, 1, 0, 0, 0, 1, 0}};

  vector<Graph> matrices = {m1, m2};
  int vertex;
  cin >> vertex;

  cout << "G1:\n";
  outputMaxSimpleChain(m1, vertex);

  cout << "G2:\n";
  outputMaxSimpleChain(m1, vertex);

  return 0;
}
```

Результат работы программы:

```
C:\BGTU\BGTU\DisMat\lab_4_1\Code\cmake-build-debug\Code.exe
2
G1:
{ 2 1 4 3 }
{ 2 1 6 7 }
{ 2 1 7 6 }
{ 2 5 1 4 3 }
{ 2 5 1 6 7 }
{ 2 5 1 7 6 }
{ 2 6 1 4 3 }
{ 2 6 1 7 }
{ 2 6 7 1 4 3 }

G2:
{ 2 1 4 3 }
{ 2 1 6 7 }
{ 2 1 7 6 }
{ 2 5 1 4 3 }
{ 2 5 1 6 7 }
{ 2 5 1 7 6 }
{ 2 6 1 4 3 }
{ 2 6 1 7 }
{ 2 6 7 1 4 3 }


Process finished with exit code 0
```

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