Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«Национальный исследовательский

Томский политехнический Университет»



Инженерная школа ядерных технологий

Направление 01.04.02 «Прикладная математика и информатика»

**ОТЧЕТ**

по ИНДИВИДУАЛЬНОМУ ЗАДАНИЮ №7

Построение кратчайшего остова графа

раскраска графа

нахождение максимального потока в сети

Вариант 1

по дисциплине:

**Дискретная математика и теория графов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Исполнитель:** |  | Е. В. Петрович | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| студент группы 0ВМ92 |  | Дата сдачи: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |  |
|  |  |  |  |
| **Руководитель:** |  | М. Л. Шинкеев | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| доцент, |  | Дата проверки: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| кандидат физико-математических наук |  |  |  |
|  |  |  |  |

Томск - 2019

Оглавление

[Задание варианта 1 2](#_Toc30666347)

[**Цель работы** 3](#_Toc30666348)

[**Основная часть** 3](#_Toc30666349)

[Задание 1. Построить граф 3](#_Toc30666350)

[Задание 2. Раскрасить граф, используя минимальное количество цветов 4](#_Toc30666351)

[Задание 3. Найти кратчайший остов графа, полагая длину ребра, равной весу ребра 5](#_Toc30666352)

[Задание 4. Считая, что множество E есть множество ориентированных ребер с пропускной способностью, равной весу ребра, а G - сетевой граф с источником в вершине V1 и стоком в вершине V45, найти максимальный поток в сети и указать минимальный (минимальные) разрез(ы) сети. 6](#_Toc30666353)

[**Приложение 1** 8](#_Toc30666354)

[Задание 1 8](#_Toc30666355)

[Задание 2 9](#_Toc30666356)

[Задание 3 12](#_Toc30666357)

[Задание 4 15](#_Toc30666358)

[**Приложение 2** 19](#_Toc30666359)

[Ссылка на проект 19](#_Toc30666360)

**Задание**

Задание варианта 1

Граф G задан списком ребер E с указанием веса ребра. Требуется, в соответствии с вариантом задания:

1. Построить граф;

2. Раскрасить граф, используя минимальное количество цветов;

2. Найти кратчайший остов графа, полагая длину ребра, равной весу ребра;

3. Считая, что множество E есть множество ориентированных ребер с пропускной способностью, равной весу ребра, а G - сетевой граф с источником в вершине V1 и стоком в вершине V45, найти максимальный поток в сети и указать минимальный (минимальные) разрез(ы) сети.

E={{1,2,5},{1,3,11},{1,4,11},{1,5,4},{1,9,5},{1,10,11},{1,12,10},{1,14,12},{1,15,6},{1,16,6},{2,6,4},{2,10,11},{3,5,4},{3,6,11},{4,15,5},{4,16,7},{5,22,10},{6,10,10},{9,20,7},{9,23,10},{10,14,4},{10,16,11},{10,22,9},{10,23,4},{12,24,10},{12,27,5},{14,16,12},{14,20,6},{15,22,9},{15,26,5},{16,18,10},{16,25,11},{16,27,4},{16,28,9},{16,29,7},{18,31,10},{18,38,6},{20,29,4},{22,26,10},{22,33,8},{23,25,8},{23,33,9},{24,28,6},{24,32,9},{25,30,5},{25,36,12},{26,31,6},{27,32,9},{27,36,6},{27,39,6},{28,37,12},{29,35,4},{29,40,7},{30,35,11},{30,40,6},{31,37,5},{31,39,6},{31,41,8},{32,33,9},{32,45,6},{33,36,5},{33,39,10},{33,45,10},{35,41,6},{35,45,5},{36,38,9},{36,45,10},{37,39,4},{38,45,11},{39,41,12},{39,45,10},{40,45,12},{41,45,4}}

**Цель работы**

Работа с графами.

**Основная часть**

Задание 1. Построить граф

Работа программы в пакете Wolfram Mathematicа по построению графа представлена на Рис. 1. Код и вывод представлен в Приложении 1, Задание 1.

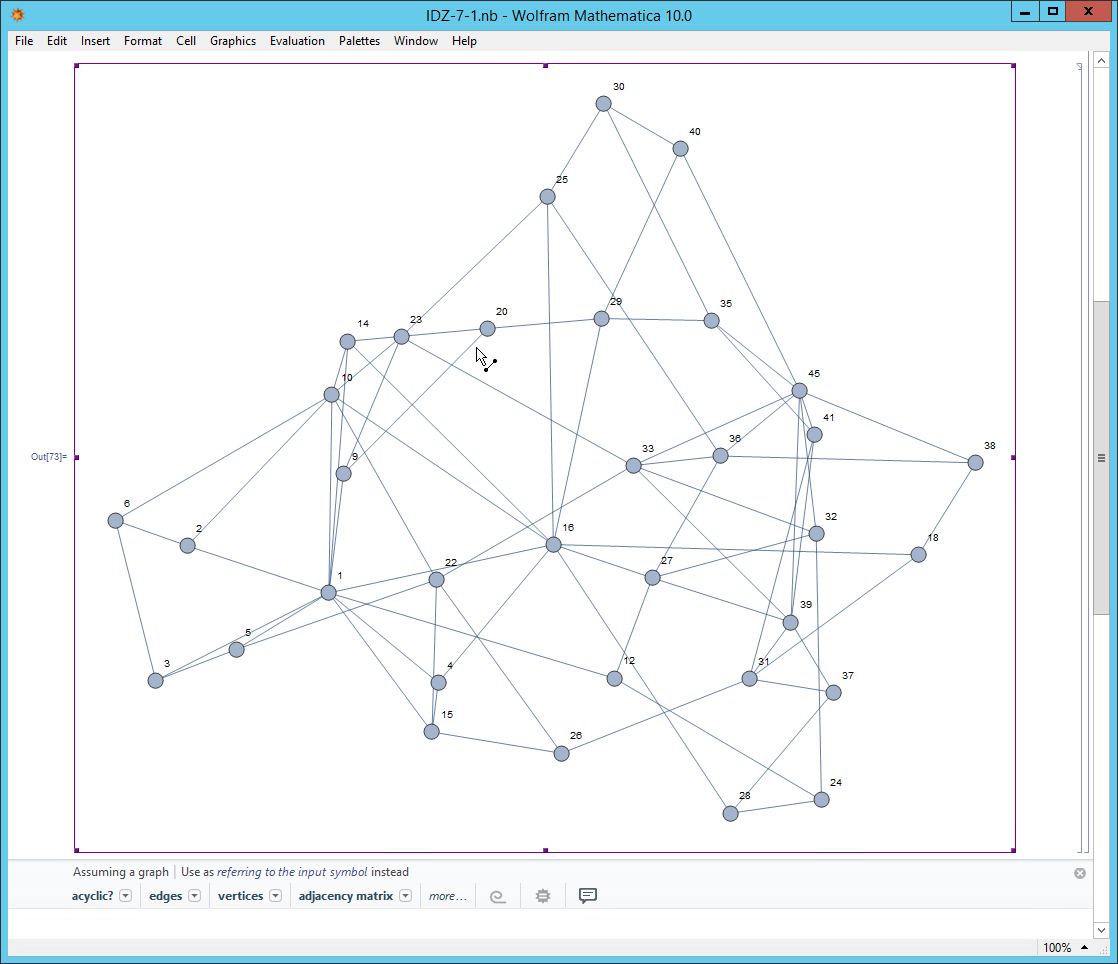


Рис. 1

Задание 2. Раскрасить граф, используя минимальное количество цветов

Для раскраски графа отсортируем все вершины по их степеням. Вершине с самой большой степенью присваиваем некоторый цвет. Далее, при проходе по всему списку, каждую последующую вершину проверяем на инцидентность с вершинами, уже имеющими текущий цвет. Если вершина инцидентна, то переходим к следующей. Если вершина не инцидентна, то она получает текущий цвет. Дойдя до конца списка, выбираем новый цвет и выполняем новый проход, пропуская окрашенные вершины. В задании первого варианта для окраски графа потребовалось 4 цвета.

Работа программы в пакете Wolfram Mathematicа по построению графа представлена на Рис.2. Код и вывод представлен в Приложении 1, Задание 2.

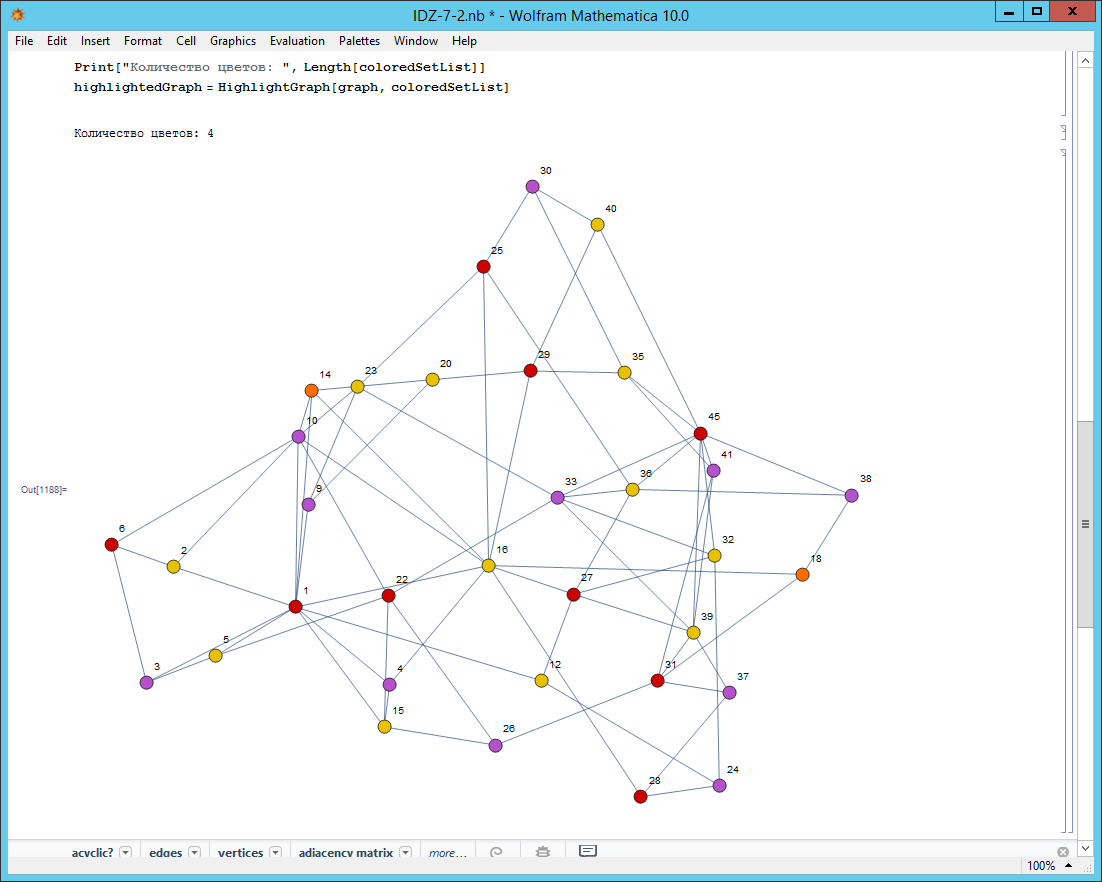


Рис. 2

Задание 3. Найти кратчайший остов графа, полагая длину ребра, равной весу ребра

Для поиска кратчайшего остова применяем алгоритм Крускала. Отсортируем все пары вершин, образующие ребра по их весу. Добавляем в остовной подграф ребра, пока количество вершин в остовном подграфе не сравняются с количеством вершин в исходном графе. Если добавляемое ребро образует в остовном подграфе петлю, оно пропускается.

Работа программы в пакете Wolfram Mathematicа представлена на Рис.3. Код и вывод представлен в Приложении 1, Задание 3.

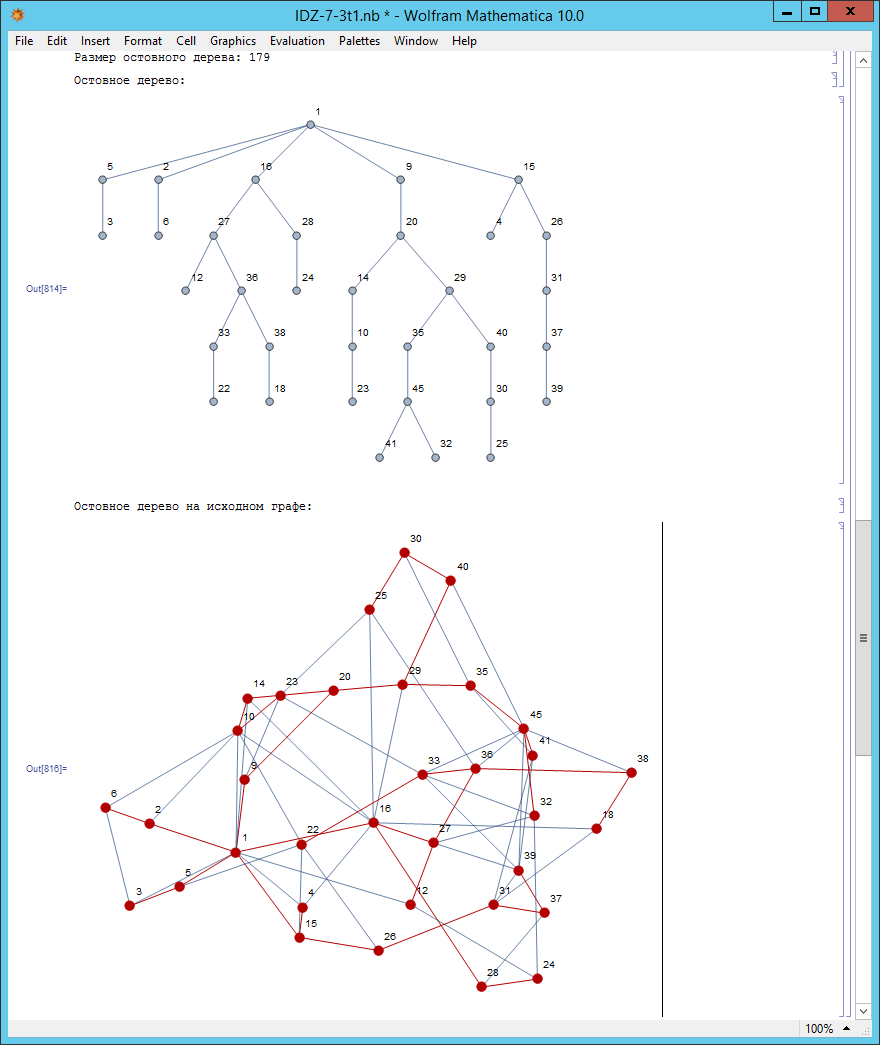


Рис. 3

Задание 4. Считая, что множество E есть множество ориентированных ребер с пропускной способностью, равной весу ребра, а G - сетевой граф с источником в вершине V1 и стоком в вершине V45, найти максимальный поток в сети и указать минимальный (минимальные) разрез(ы) сети.

Для расчета максимального потока в сети применяем алгоритм Форда — Фалкерсона.

Первый этап алгоритма заключается в поиске всех маршрутов от истока к стоку и последовательное заполнение маршрутов их максимально возможными потоками. Величина максимально возможного потока для маршрута определяется минимальной остаточной пропускной способности ребер, входящих в маршрут. После того, как все такие маршруты найдены, выполняется проход с поиском маршрутов, в которых можно перераспределить поток, уменьшив потоки в некоторых ребрах, направленных против найденного маршрута. Для определения минимального разреза в сети последовательно включаем в множество вершин, начиная с истока, все вершины исходящие ребра которых – не насыщены.

Работа программы в пакете Wolfram Mathematicа представлена на Рис.4. Код и вывод представлен в Приложении 1, Задание 4.

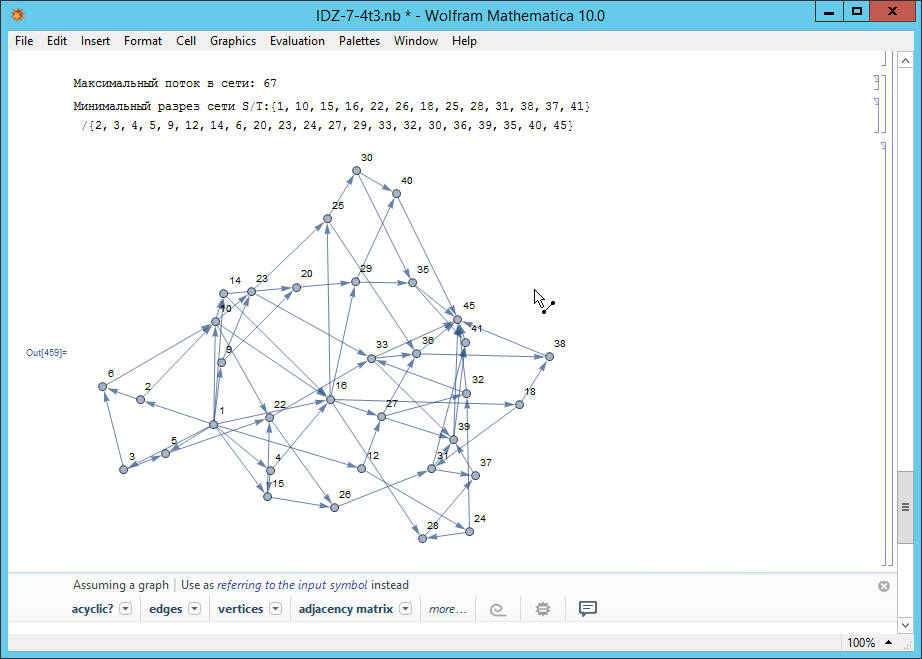


Рис.

**Заключение**

В работе проведены операции над графами в пакете Wolfram Mathematica. Представлены скриншоты работы программы, исходные коды и ответы.

**Приложение 1**

Задание 1

Unprotect[E]

E = {{1, 2, 5}, {1, 3, 11}, {1, 4, 11}, {1, 5, 4}, {1, 9, 5}, {1, 10,

11}, {1, 12, 10}, {1, 14, 12}, {1, 15, 6}, {1, 16, 6}, {2, 6,

4}, {2, 10, 11}, {3, 5, 4}, {3, 6, 11}, {4, 15, 5}, {4, 16,

7}, {5, 22, 10}, {6, 10, 10}, {9, 20, 7}, {9, 23, 10}, {10, 14,

4}, {10, 16, 11}, {10, 22, 9}, {10, 23, 4}, {12, 24, 10}, {12, 27,

5}, {14, 16, 12}, {14, 20, 6}, {15, 22, 9}, {15, 26, 5}, {16, 18,

10}, {16, 25, 11}, {16, 27, 4}, {16, 28, 9}, {16, 29, 7}, {18,

31, 10}, {18, 38, 6}, {20, 29, 4}, {22, 26, 10}, {22, 33, 8}, {23,

25, 8}, {23, 33, 9}, {24, 28, 6}, {24, 32, 9}, {25, 30, 5}, {25,

36, 12}, {26, 31, 6}, {27, 32, 9}, {27, 36, 6}, {27, 39, 6}, {28,

37, 12}, {29, 35, 4}, {29, 40, 7}, {30, 35, 11}, {30, 40, 6}, {31,

37, 5}, {31, 39, 6}, {31, 41, 8}, {32, 33, 9}, {32, 45, 6}, {33,

36, 5}, {33, 39, 10}, {33, 45, 10}, {35, 41, 6}, {35, 45, 5}, {36,

38, 9}, {36, 45, 10}, {37, 39, 4}, {38, 45, 11}, {39, 41,

12}, {39, 45, 10}, {40, 45, 12}, {41, 45, 4}};

(\*Edge lists\*)

edgeList = Map[Function[x, UndirectedEdge[Part[x, 1], Part[x, 2]]], E];

(\*subGraphs\*)

graph = Graph[edgeList];

(\*вес\*)

For[i = 1, i <= Length[E], i++,

graph = SetProperty[{graph, edgeList[[i]]},

EdgeWeight -> E[[i]][[3]]];

]

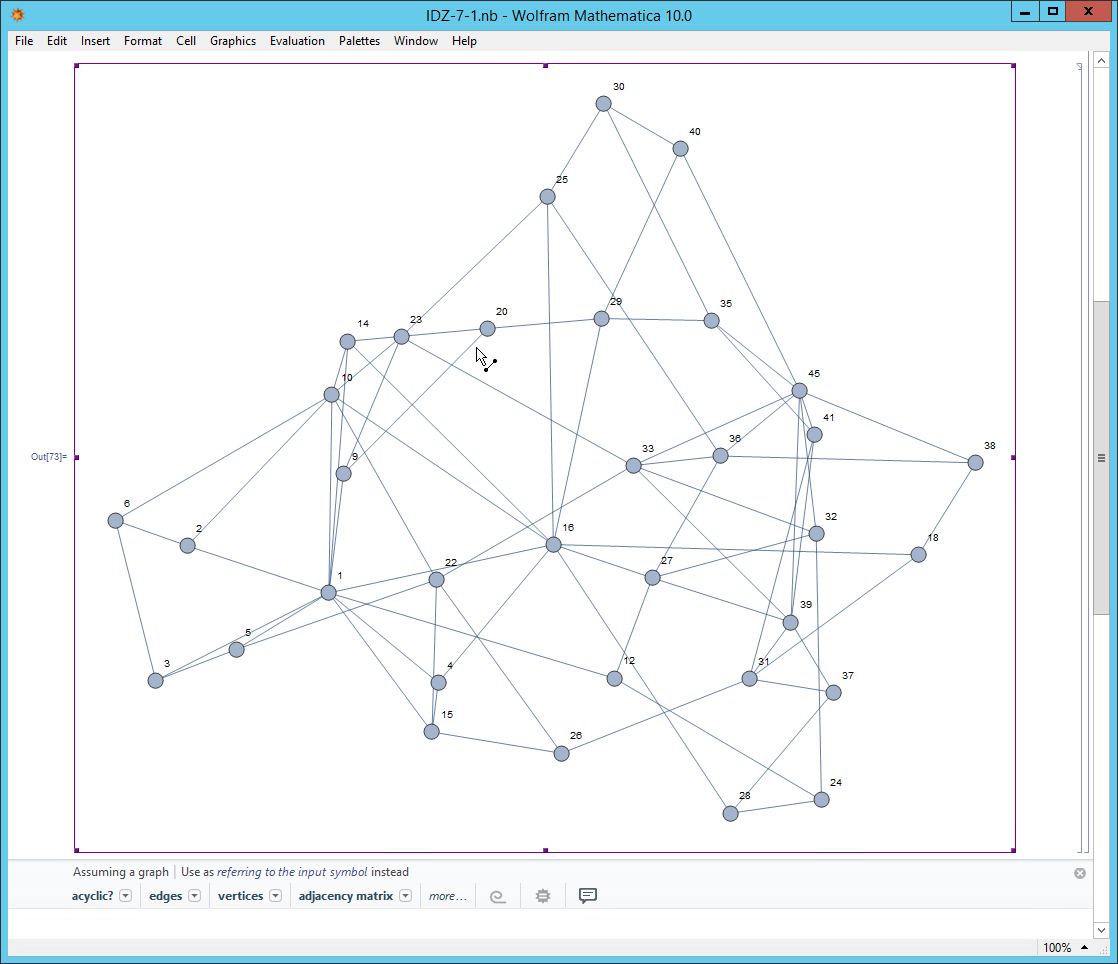
graph = SetProperty[graph, VertexLabels -> "Name"];

graph = SetProperty[graph,

GraphLayout -> {VertexLayout -> {"SpringElectricalEmbedding",

"EdgeWeighted" -> True}}];

graph



Задание 2

(\*Раскраска графа\*)

Unprotect[E];

E = {{1, 2, 5}, {1, 3, 11}, {1, 4, 11}, {1, 5, 4}, {1, 9, 5}, {1, 10,

11}, {1, 12, 10}, {1, 14, 12}, {1, 15, 6}, {1, 16, 6}, {2, 6,

4}, {2, 10, 11}, {3, 5, 4}, {3, 6, 11}, {4, 15, 5}, {4, 16,

7}, {5, 22, 10}, {6, 10, 10}, {9, 20, 7}, {9, 23, 10}, {10, 14,

4}, {10, 16, 11}, {10, 22, 9}, {10, 23, 4}, {12, 24, 10}, {12, 27,

5}, {14, 16, 12}, {14, 20, 6}, {15, 22, 9}, {15, 26, 5}, {16, 18,

10}, {16, 25, 11}, {16, 27, 4}, {16, 28, 9}, {16, 29, 7}, {18,

31, 10}, {18, 38, 6}, {20, 29, 4}, {22, 26, 10}, {22, 33, 8}, {23,

25, 8}, {23, 33, 9}, {24, 28, 6}, {24, 32, 9}, {25, 30, 5}, {25,

36, 12}, {26, 31, 6}, {27, 32, 9}, {27, 36, 6}, {27, 39, 6}, {28,

37, 12}, {29, 35, 4}, {29, 40, 7}, {30, 35, 11}, {30, 40, 6}, {31,

37, 5}, {31, 39, 6}, {31, 41, 8}, {32, 33, 9}, {32, 45, 6}, {33,

36, 5}, {33, 39, 10}, {33, 45, 10}, {35, 41, 6}, {35, 45, 5}, {36,

38, 9}, {36, 45, 10}, {37, 39, 4}, {38, 45, 11}, {39, 41,

12}, {39, 45, 10}, {40, 45, 12}, {41, 45, 4}};

(\*Edge lists\*)

edgeList = Map[Function[x, UndirectedEdge[Part[x, 1], Part[x, 2]]], E];

weights = Map[Function[x, Part[x, 3]], E];

graph = Graph[edgeList, EdgeWeight -> weights];

graph = SetProperty[graph, VertexLabels -> "Name"];

graph = SetProperty[graph,

GraphLayout -> {VertexLayout -> {"SpringElectricalEmbedding",

"EdgeWeighted" -> True}}];

graph;

(\*verticess degree table\*)

(\*1-vertex, 2-color mark, 3-degree\*)

vertexDegree = VertexDegree[graph];

vertexTab = Partition[VertexList[graph], 1];

For[i = 1, i <= Length[vertexTab], i++,

AppendTo[vertexTab[[i]], 0];

AppendTo[vertexTab[[i]], vertexDegree[[i]]];

]

vertexTab = SortBy[vertexTab, Last];

coloredSetList = {};

For[c = 1, c <= Length[vertexDegreeTab], c++,

coloredSet = {};

For[i = Length[vertexTab], i > 0, i--,

If[vertexTab[[i]][[2]] == 0,

(\*check vertex\*)

res = True;

If[Length[coloredSet] != 0,

For[j = 1, j <= Length[coloredSet], j++,

If[

MemberQ[edgeList,

UndirectedEdge[vertexTab[[i]][[1]], coloredSet[[j]]]]

||

MemberQ[edgeList,

UndirectedEdge[coloredSet[[j]], vertexTab[[i]][[1]]]],

res = False;

Break[];]

];

];

If[res == True, AppendTo[coloredSet, vertexTab[[i]][[1]]];

vertexTab[[i]][[2]] = c];

]

];

If[Length[coloredSet] == 0, Break[]];

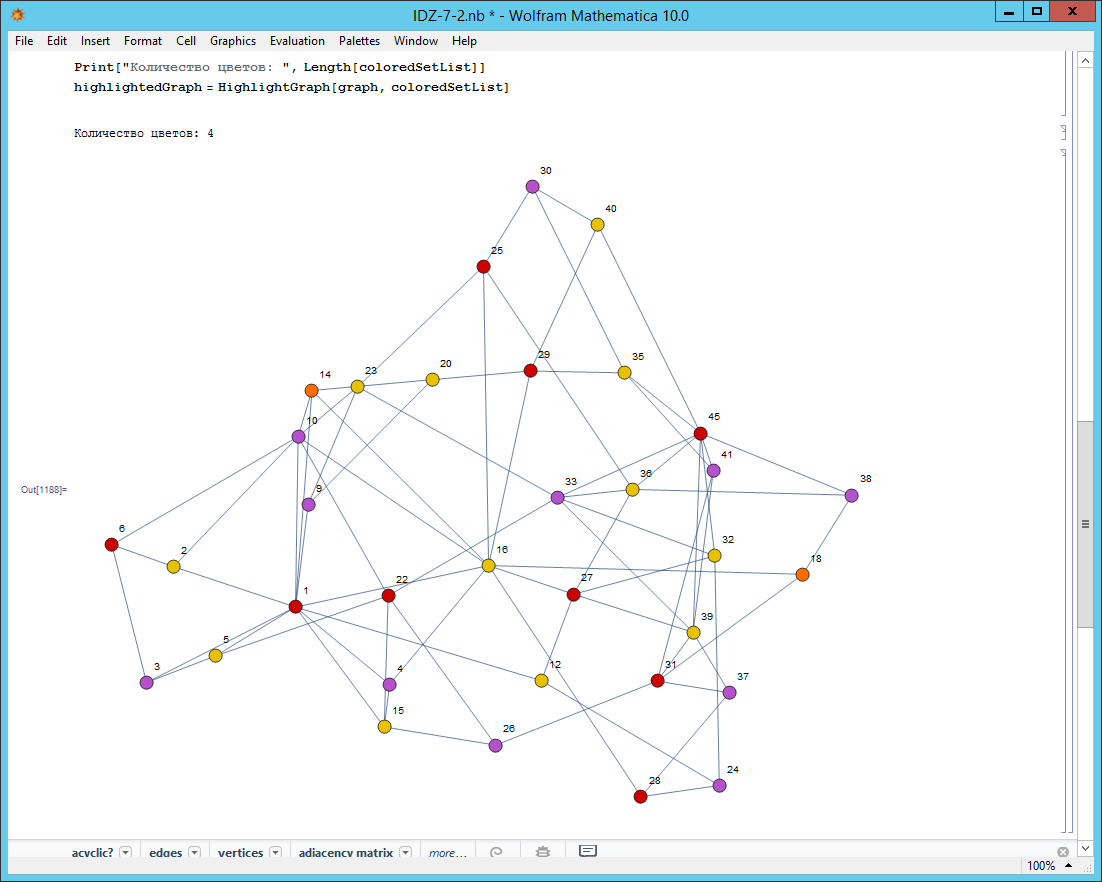
AppendTo[coloredSetList, coloredSet];

ClearAll[coloredSet];

];

Print["Количество цветов: ", Length[coloredSetList]]

highlightedGraph = HighlightGraph[graph, coloredSetList]



Задание 3

Unprotect[E];

E = {{1, 2, 5}, {1, 3, 11}, {1, 4, 11}, {1, 5, 4}, {1, 9, 5}, {1, 10,

11}, {1, 12, 10}, {1, 14, 12}, {1, 15, 6}, {1, 16, 6}, {2, 6,

4}, {2, 10, 11}, {3, 5, 4}, {3, 6, 11}, {4, 15, 5}, {4, 16,

7}, {5, 22, 10}, {6, 10, 10}, {9, 20, 7}, {9, 23, 10}, {10, 14,

4}, {10, 16, 11}, {10, 22, 9}, {10, 23, 4}, {12, 24, 10}, {12, 27,

5}, {14, 16, 12}, {14, 20, 6}, {15, 22, 9}, {15, 26, 5}, {16, 18,

10}, {16, 25, 11}, {16, 27, 4}, {16, 28, 9}, {16, 29, 7}, {18,

31, 10}, {18, 38, 6}, {20, 29, 4}, {22, 26, 10}, {22, 33, 8}, {23,

25, 8}, {23, 33, 9}, {24, 28, 6}, {24, 32, 9}, {25, 30, 5}, {25,

36, 12}, {26, 31, 6}, {27, 32, 9}, {27, 36, 6}, {27, 39, 6}, {28,

37, 12}, {29, 35, 4}, {29, 40, 7}, {30, 35, 11}, {30, 40, 6}, {31,

37, 5}, {31, 39, 6}, {31, 41, 8}, {32, 33, 9}, {32, 45, 6}, {33,

36, 5}, {33, 39, 10}, {33, 45, 10}, {35, 41, 6}, {35, 45, 5}, {36,

38, 9}, {36, 45, 10}, {37, 39, 4}, {38, 45, 11}, {39, 41,

12}, {39, 45, 10}, {40, 45, 12}, {41, 45, 4}};

(\*Edge lists\*)

edgeList = Map[Function[x, UndirectedEdge[Part[x, 1], Part[x, 2]]], E];

weights = Map[Function[x, Part[x, 3]], E];

graph = Graph[edgeList, EdgeWeight -> weights];

graph = SetProperty[graph, VertexLabels -> "Name"];

graph = SetProperty[graph,

GraphLayout -> {VertexLayout -> {"SpringElectricalEmbedding",

"EdgeWeighted" -> True}}];

graph;

vertexCount = VertexCount[graph];

sortedE = SortBy[E, Last];

FindSet[vertices\_List, vertex\_] := Module[{i},

For[i = 1, i <= Length[vertices], i++,

If[MemberQ[vertices[[i]], vertex], Return[i];];

];

Return[0];

]

vertices = {};

edges = {};

edgePaths = {};

AppendTo[vertices, {sortedE[[1]][[1]], sortedE[[1]][[2]]}];

AppendTo[edges, UndirectedEdge[sortedE[[1]][[1]], sortedE[[1]][[2]]]];

insCount = 2;

treeSize = sortedE[[1]][[3]];

For[i = 2, i <= Length[sortedE], i++,

If[insCount >= vertexCount && Length[vertices] == 1, Break[]];

v1 = sortedE[[i]][[1]];

v2 = sortedE[[i]][[2]];

set1 = FindSet[vertices, v1];

set2 = FindSet[vertices, v2];

If[set1 == set2 != 0, Continue[]];

(\*append edge in all other cases\*)

AppendTo[edges,

UndirectedEdge[sortedE[[i]][[1]], sortedE[[i]][[2]]]];

treeSize = treeSize + sortedE[[i]][[3]];

(\*new set\*)

If[set1 == set2 == 0,

AppendTo[vertices, {sortedE[[i]][[1]], sortedE[[i]][[2]]}];

insCount = insCount + 2;

Continue[];];

If[set1 != set2 && set1 > 0 && set2 > 0,

(\*join sets\*)

uSet = Join[vertices[[set1]], vertices[[set2]]];

delList = Sort[{{set1}, {set2}}];

vertices = Delete[vertices, delList[[2]]];

vertices = Delete[vertices, delList[[1]]];

AppendTo[vertices, uSet];

Continue[];

]

(\*set1\[NotEqual]set2 && set1\[Equal]0 || set2==0\*)

AppendTo[vertices[[set1 + set2]],

sortedE[[i]][[1]]\*Boole[set1 == 0] +

sortedE[[i]][[2]]\*Boole[set2 == 0]];

insCount++;

];

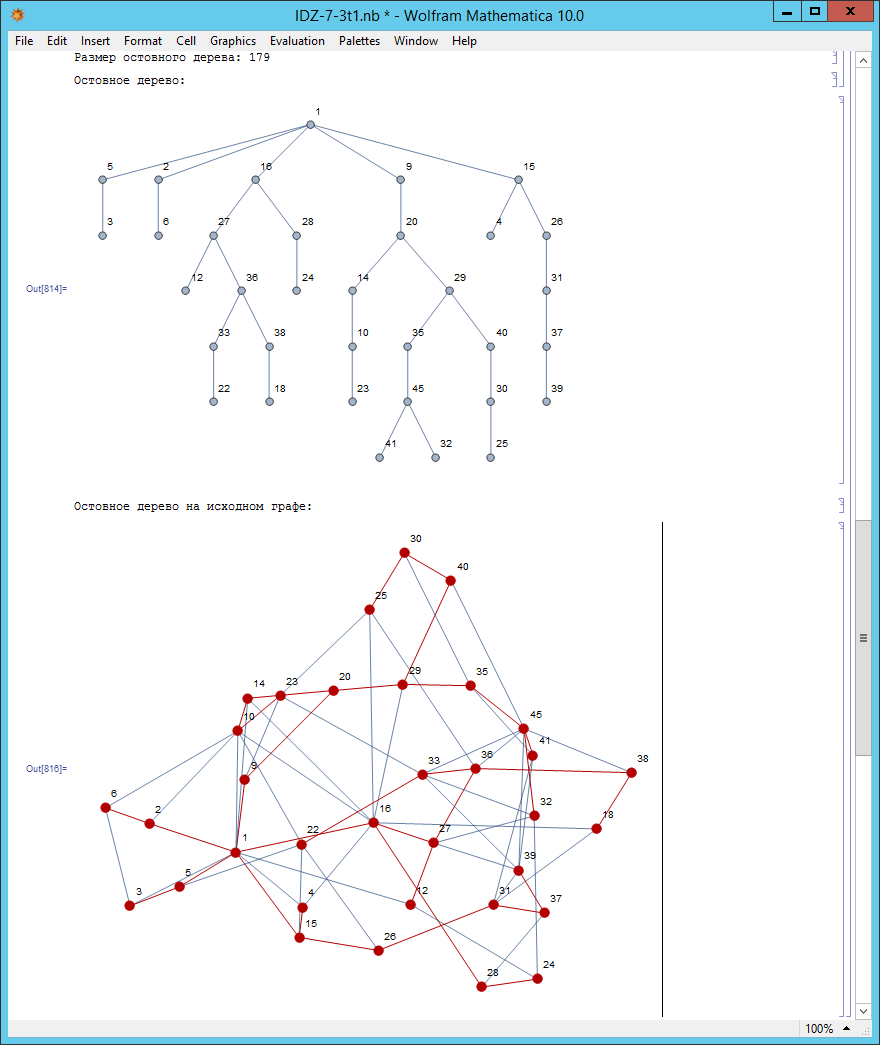
Print["Размер остовного дерева: ", treeSize];

Print["Остовное дерево:"];

treeGraph = Graph[edges, VertexLabels -> "Name"]

Print["Остовное дерево на исходном графе:"];

highlightedGraph = HighlightGraph[graph, treeGraph]



Задание 4

Unprotect[E];

E = {{1, 2, 5}, {1, 3, 11}, {1, 4, 11}, {1, 5, 4}, {1, 9, 5}, {1, 10,

11}, {1, 12, 10}, {1, 14, 12}, {1, 15, 6}, {1, 16, 6}, {2, 6,

4}, {2, 10, 11}, {3, 5, 4}, {3, 6, 11}, {4, 15, 5}, {4, 16,

7}, {5, 22, 10}, {6, 10, 10}, {9, 20, 7}, {9, 23, 10}, {10, 14,

4}, {10, 16, 11}, {10, 22, 9}, {10, 23, 4}, {12, 24, 10}, {12, 27,

5}, {14, 16, 12}, {14, 20, 6}, {15, 22, 9}, {15, 26, 5}, {16, 18,

10}, {16, 25, 11}, {16, 27, 4}, {16, 28, 9}, {16, 29, 7}, {18,

31, 10}, {18, 38, 6}, {20, 29, 4}, {22, 26, 10}, {22, 33, 8}, {23,

25, 8}, {23, 33, 9}, {24, 28, 6}, {24, 32, 9}, {25, 30, 5}, {25,

36, 12}, {26, 31, 6}, {27, 32, 9}, {27, 36, 6}, {27, 39, 6}, {28,

37, 12}, {29, 35, 4}, {29, 40, 7}, {30, 35, 11}, {30, 40, 6}, {31,

37, 5}, {31, 39, 6}, {31, 41, 8}, {32, 33, 9}, {32, 45, 6}, {33,

36, 5}, {33, 39, 10}, {33, 45, 10}, {35, 41, 6}, {35, 45, 5}, {36,

38, 9}, {36, 45, 10}, {37, 39, 4}, {38, 45, 11}, {39, 41,

12}, {39, 45, 10}, {40, 45, 12}, {41, 45, 4}};

(\*Edge lists\*)

edgeList = Map[Function[x, DirectedEdge[Part[x, 1], Part[x, 2]]], E];

weights = Map[Function[x, Part[x, 3]], E];

graph = Graph[edgeList, EdgeWeight -> weights];

graph = SetProperty[graph, VertexLabels -> "Name"];

graph = SetProperty[graph,

GraphLayout -> {VertexLayout -> {"SpringElectricalEmbedding",

"EdgeWeighted" -> True}}];

graph;

(\*rested bandwidth matrix\*)

R = E;(\*

\*)edgeCount = Length[R];

GetAdjacency[v\_, findBackward\_: False] := Module[{},

adjacencyList = {};

For[i = 1, i <= edgeCount, i++,

If[R[[i]][[1]] == v && R[[i]][[3]] != 0,

AppendTo[adjacencyList, R[[i]]]];

If[findBackward,

If[R[[i]][[2]] == v && E[[i]][[3]] - R[[i]][[3]] > 0,

bkw = {R[[i]][[2]], R[[i]][[1]], E[[i]][[3]] - R[[i]][[3]]};

AppendTo[adjacencyList, bkw];];

];

];

Return[adjacencyList];

]

SearchPath[vertex\_, trg\_, findBackward\_, minFlow\_: Infinity,

visited\_List: {}, vPath\_List: {}] :=

Module[{i, resList, adjacencyList, locMinFlow, locVPath, locVisited,

locFindBackward},

locFindBackward = findBackward;

locVPath = vPath;

locMinFlow = minFlow;

locVisited = visited;

resList = {};

AppendTo[locVPath, vertex];

AppendTo[locVisited, vertex];

AppendTo[resList, locVisited];

(\*if trg\*)

If[vertex == trg,

(\*return minFlow, path\*)

AppendTo[resList, {True, locVPath, minFlow}];

,

(\*go forward\*)

AppendTo[resList, {False, locVPath, minFlow}];

adjacencyList = GetAdjacency[vertex, locFindBackward];

If[Length[adjacencyList] == 0,

Return[resList]];

For[i = 1, i <= Length[adjacencyList], i++,

If[MemberQ[locVPath, adjacencyList[[i]][[2]]] ||

MemberQ[locVisited, adjacencyList[[i]][[2]]], Continue[]];

locMinFlow = Min[minFlow, adjacencyList[[i]][[3]]];

resList =

SearchPath[adjacencyList[[i]][[2]], trg, locFindBackward,

locMinFlow, locVisited, locVPath];

If[Length[resList] > 1 && resList[[2]][[1]] == True, Break[]];

locVisited = resList[[1]];

];

];

Return[resList];

];(\*END SEARCH\*)

RecalcMatrix[R\_List, vPath\_List, minFlow\_] := Module[{i, j, locR},

locR = R;

For[i = 1, i < Length[vPath], i++,

For[j = 1, j <= Length[locR], j++,

If[vPath[[i]] == locR[[j]][[1]] &&

vPath[[i + 1]] == locR[[j]][[2]],

locR[[j]][[3]] = locR[[j]][[3]] - minFlow;];

(\*slow down in backward edges\*)

If[vPath[[i]] == locR[[j]][[2]] &&

vPath[[i + 1]] == locR[[j]][[1]],

locR[[j]][[3]] = locR[[j]][[3]] + minFlow;];

];

];

Return[locR];

]

pathList = {};

src = 1;

trg = 45;

totalFlow = 0;

backward = False;

(\*Clear[GetAdjacency,RecalcMatrix,SearchPath];\*)

Do[

maxIt = 100;

While[maxIt > 0,

res = SearchPath[src, trg, backward];

If[res[[2]][[1]],

AppendTo[pathList, {res[[2]][[2]], res[[2]][[3]]}];

totalFlow = totalFlow + res[[2]][[3]];

R = RecalcMatrix[R, res[[2]][[2]], res[[2]][[3]]];,

Break[];

];

maxIt--;

];

backward = ! backward;

, {2}];

(\*Min cut\*)

S = {1};

T = Delete[VertexList[graph], 1];

For[i = 1, i <= Length[S], i++,

For[j = 1, j <= Length[R], j++,

If[R[[j]][[1]] == S[[i]] &&

R[[j]][[3]] > 0 && ! MemberQ[S, R[[j]][[2]]],

AppendTo[S, R[[j]][[2]]];

For[k = 1, k <= Length[T], k++,

If[T[[k]] == R[[j]][[2]],

T = Delete[T, k]; Break[]];

];

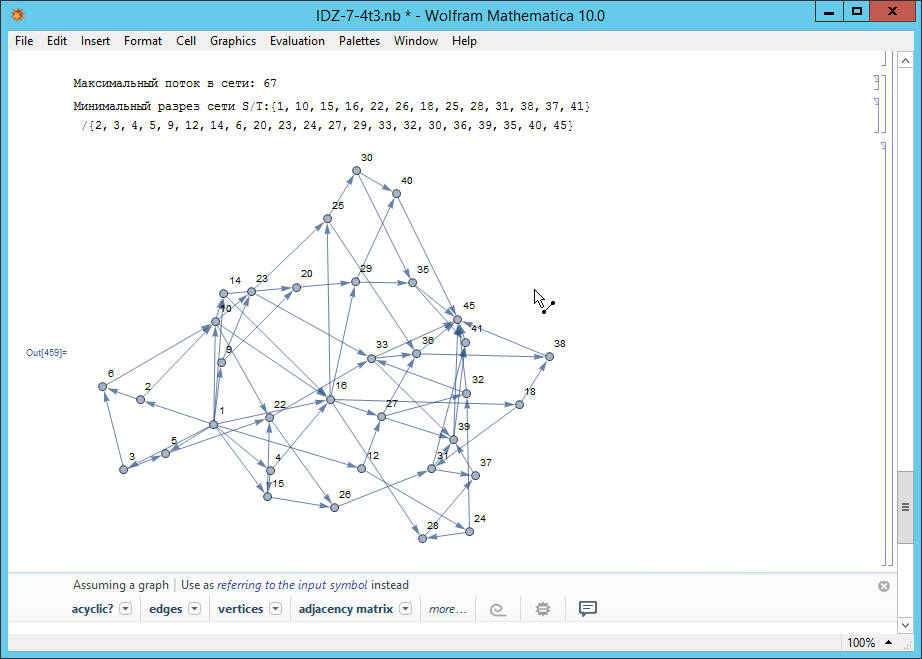
];

];

];

Print["Максимальный поток в сети: ", totalFlow];

Print["Минимальный разрез сети S/T:", S, "/", T];



**Приложение 2**

Ссылка на проект

[https://](https://github.com/petrovicheugene/MobAppLabs.git)github.com/[petrovicheugene](https://github.com/petrovicheugene/DiscreteMathAndGraphs.git)/[DiscreteMathAndGraphs](https://github.com/petrovicheugene/DiscreteMathAndGraphs.git).git