**Лабораторная работа №4**

**«Взвешенные графы»**

**Задание №1.**

Измените программу path.java (см. листинг 14.2), чтобы она выводила таблицу минимальной стоимости перемещения от любой вершины до всех остальных вершин. Для этого придется внести изменения в методы, предполагающие, что начальной вершиной всегда является вершина A.

Листинг 14.2. Программа path.java

*// path.java*

*// Выбор кратчайшего пути в направленном взвешенном графе*

*// Запуск программы: C>java PathApp*

*////////////////////////////////////////////////////////////////*

**class** DistPar *// Расстояние и родительская вершина*

{ *// Объекты сохраняются в массиве sPath*

**public** **int** distance; *// Расстояние от начальной вершины до текущей*

**public** **int** parentVert; *// Текущий родитель вершины*

*// -------------------------------------------------------------*

**public** DistPar(**int** pv, **int** d) *// Конструктор*

{

distance = d;

parentVert = pv;

}

} *// Конец класса DistPar*

*///////////////////////////////////////////////////////////////*

**class** Vertex

{

**public** **char** label; *// Метка (например, 'A')*

**public** **boolean** isInTree;

*// -------------------------------------------------------------*

**public** Vertex(**char** lab) *// Конструктор*

{

label = lab;

isInTree = **false**;

}

*// -------------------------------------------------------------*

} *// Конец класса Vertex*

*////////////////////////////////////////////////////////////////*

**class** Graph

{

**private** **final** **int** MAX\_VERTS = 20;

**private** **final** **int** INFINITY = 1000000;

**private** Vertex vertexList[]; *// Список вершин*

**private** **int** adjMat[][]; *// Матрица смежности*

**private** **int** nVerts; *// Текущее количество вершин*

**private** **int** nTree; *// Количество вершин в дереве*

**private** DistPar sPath[]; *// Массив данных кратчайших путей*

**private** **int** currentVert; *// Текущая вершина*

**private** **int** startToCurrent; *// Расстояние до currentVert*

*// -------------------------------------------------------------*

**public** Graph() *// Конструктор*

{

vertexList = **new** Vertex[MAX\_VERTS];

*// Матрица смежности*

adjMat = **new** **int**[MAX\_VERTS][MAX\_VERTS];

nVerts = 0;

nTree = 0;

**for**(**int** j=0; j<MAX\_VERTS; j++) *// Матрица смежности*

**for**(**int** k=0; k<MAX\_VERTS; k++) *// заполняется*

adjMat[j][k] = INFINITY; *// бесконечными расстояниями*

sPath = **new** DistPar[MAX\_VERTS]; *// shortest paths*

}

*// -------------------------------------------------------------*

**public** **void** addVertex(**char** lab)

{

vertexList[nVerts++] = **new** Vertex(lab);

}

*// -------------------------------------------------------------*

**public** **void** addEdge(**int** start, **int** end, **int** weight)

{

adjMat[start][end] = weight; *// (направленный граф)*

}

*// -------------------------------------------------------------*

**public** **void** path() *// Выбор кратчайших путей*

{

**int** startTree = 0; *// Начиная с вершины 0*

vertexList[startTree].isInTree = **true**;

nTree = 1; *// Включение в дерево*

*// Перемещение строки расстояний из adjMat в sPath*

**for**(**int** j=0; j<nVerts; j++)

{

**int** tempDist = adjMat[startTree][j];

sPath[j] = **new** DistPar(startTree, tempDist);

}

*// Пока все вершины не окажутся в дереве*

**while**(nTree < nVerts)

{

**int** indexMin = getMin(); *// Получение минимума из sPath*

**int** minDist = sPath[indexMin].distance;

**if**(minDist == INFINITY) *// Если все расстояния бесконечны*

{ *// или уже находятся в дереве,*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("There are unreachable vertices");

**break**; *// построение sPath завершено*

}

**else**

{ *// Возврат currentVert*

currentVert = indexMin; *// к ближайшей вершине*

startToCurrent = sPath[indexMin].distance;

*// Минимальное расстояние от startTree*

*// до currentVert равно startToCurrent*

}

*// Включение текущей вершины в дерево*

vertexList[currentVert].isInTree = **true**;

nTree++;

adjust\_sPath(); *// Обновление массива sPath[]*

}

displayPaths(); *// Вывод содержимого sPath[]*

nTree = 0; *// Очистка дерева*

**for**(**int** j=0; j<nVerts; j++)

vertexList[j].isInTree = **false**;

}

*// -------------------------------------------------------------*

**public** **int** getMin() *// Поиск в sPath элемента*

{ *// с наименьшим расстоянием*

**int** minDist = INFINITY; *// Исходный высокий "минимум"*

**int** indexMin = 0;

**for**(**int** j=1; j<nVerts; j++) *// Для каждой вершины*

{ *// Если она не включена в дерево*

**if**( !vertexList[j].isInTree && *// и ее расстояние меньше*

sPath[j].distance < minDist ) *// старого минимума*

{

minDist = sPath[j].distance;

indexMin = j; *// Обновление минимума*

}

}

**return** indexMin; *// Метод возвращает индекс*

} *// элемента с наименьшим расстоянием*

*// -------------------------------------------------------------*

**public** **void** adjust\_sPath()

{

*// Обновление данных в массиве кратчайших путей sPath*

**int** column = 1; *// Начальная вершина пропускается*

**while**(column < nVerts) *// Перебор столбцов*

{

*// Если вершина column уже включена в дерево, она пропускается*

**if**( vertexList[column].isInTree )

{

column++;

**continue**;

}

*// Вычисление расстояния для одного элемента sPath*

*// Получение ребра от currentVert к column*

**int** currentToFringe = adjMat[currentVert][column];

*// Суммирование расстояний*

**int** startToFringe = startToCurrent + currentToFringe;

*// Определение расстояния текущего элемента sPath*

**int** sPathDist = sPath[column].distance;

*// Сравнение расстояния от начальной вершины с элементом sPath*

**if**(startToFringe < sPathDist) *// Если меньше,*

{ *// данные sPath обновляются*

sPath[column].parentVert = currentVert;

sPath[column].distance = startToFringe;

}

column++;

}

}

*// -------------------------------------------------------------*

**public** **void** displayPaths()

{

**for**(**int** j=0; j<nVerts; j++) *// display contents of sPath[]*

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(vertexList[j].label + "="); *// B=*

**if**(sPath[j].distance == INFINITY)

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("inf"); *// inf*

**else**

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(sPath[j].distance); *// 50*

**char** parent = vertexList[ sPath[j].parentVert ].label;

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("(" + parent + ") "); *// (A)*

}

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("");

}

*// -------------------------------------------------------------*

} *// Конец класса Graph*

*////////////////////////////////////////////////////////////////*

**class** PathApp

{

**public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args)

{

Graph theGraph = **new** Graph();

theGraph.addVertex('A'); *// 0 (исходная вершина)*

theGraph.addVertex('B'); *// 1*

theGraph.addVertex('C'); *// 2*

theGraph.addVertex('D'); *// 3*

theGraph.addVertex('E'); *// 4*

theGraph.addEdge(0, 1, 50); *// AB 50*

theGraph.addEdge(0, 3, 80); *// AD 80*

theGraph.addEdge(1, 2, 60); *// BC 60*

theGraph.addEdge(1, 3, 90); *// BD 90*

theGraph.addEdge(2, 4, 40); *// CE 40*

theGraph.addEdge(3, 2, 20); *// DC 20*

theGraph.addEdge(3, 4, 70); *// DE 70*

theGraph.addEdge(4, 1, 50); *// EB 50*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Shortest paths");

theGraph.path(); *// Кратчайшие пути*

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println();

}

} *// Конец класса PathApp*

*////////////////////////////////////////////////////////////////*