Лабораторна робота №8.2

Виконав: студент 1-го курсу ФІОТ групи ІП-92 Медведєв Михайло Євгенович

**Завдання:**

Реалізувати програмне застосування (програму), яке виконує наступні функції.

1.Зчитування графу з вхідного файлу. На вхід подається текстовий файл наступного вигляду:

n m

v1u1w1

v2u2w2

.......

Vmumwm

Тут n–кількість вершин графу (ціле число, більше нуля), m–кількість ребер графу (ціле число, більше нуля), viта ui–початкова та кінцева вершина ребра i(1≤vi≤n, 1≤ui≤n, цілі числа), wi–вага ребра (vi,ui). Індексація вершин у файлі ведеться з 1. Вважається, що граф є неорієнтованим.

2.Побудувати мінімальне кістякове дерево. За допомогою або алгоритму Прима, або алгоритму Краскала (на вибір) розв’язати задачу знаходження мінімального кістякового дерева в неорієнтованому графі.

Код:

**package** com.company;  
  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** DS\_IP92\_LR82\_MedvedievM {  
  
 **public static void** main(String[] args) **throws** IOException {  
 UndirectedGraph graph = **new** UndirectedGraph(**new** File("inputs/input2.txt"));  
 graph.algorithmByPrim();  
 }  
  
}  
  
**abstract class** Graph {  
 **protected int**[][] verges;  
 **protected int** numberOfNodes, numberOfVerges;// n вершин, m ребер  
 **protected int**[][] incidenceMatrix, adjacencyMatrix;  
  
 **protected** Graph(File file) **throws** FileNotFoundException {  
 parseFile(file);  
 preSetAdjacencyMatrix();  
 preSetIncidenceMatrix();  
 }  
  
  
  
 **private void** parseFile(File file) **throws** FileNotFoundException {  
 Scanner fileScanner = **new** Scanner(file);  
 **this**.numberOfNodes = fileScanner.nextInt();  
 **this**.numberOfVerges = fileScanner.nextInt();  
 **this**.verges = **new int**[**this**.numberOfVerges][3];  
 **for** (**int** i = 0; i < **this**.numberOfVerges; i++) {  
 verges[i][0] = fileScanner.nextInt();  
 verges[i][1] = fileScanner.nextInt();  
 verges[i][2] = fileScanner.nextInt();  
 }  
 }  
  
 **protected void** preSetIncidenceMatrix() {  
 **this**.incidenceMatrix = **new int**[**this**.numberOfNodes][**this**.numberOfVerges];  
 }  
  
 **protected void** preSetAdjacencyMatrix() {  
 **this**.adjacencyMatrix = **new int**[**this**.numberOfNodes][**this**.numberOfNodes];  
 }  
  
  
 **public int**[][] getAdjacencyMatrix() {  
 **return** adjacencyMatrix;  
 }  
  
  
 **protected static** String matrixToString(**int**[][] matrix, String extraText) {  
 StringBuilder outputText = **new** StringBuilder(extraText + "\n");  
  
 **for** (**int** i = 0; i < matrix.length; i++) {  
 **for** (**int** j = 0; j < matrix[0].length; j++)  
 outputText.append((matrix[i][j] >= 0) ? " " : "").append(matrix[i][j]).append(" ");  
  
 outputText.append("\n");  
 }  
 **return** outputText.toString();  
 }  
  
}  
  
**class** UndirectedGraph **extends** Graph {  
  
 **protected** UndirectedGraph(File file) **throws** FileNotFoundException {  
 **super**(file);  
// findEulerPath();  
// findGamiltonPath();  
  
 }  
  
 **public void** algorithmByPrim() {  
 ArrayList<Integer> doneNodes = **new** ArrayList<>();  
 doneNodes.add(0);  
 ArrayList<**int**[]> doneVerges = **new** ArrayList<>();  
 **while** (doneNodes.size() < numberOfNodes) {  
 **int** minimalWay = Integer.*MAX\_VALUE*, minIndexX = -1, minIndexY = -1;  
 **for** (**int** i = 0; i < adjacencyMatrix.length; i++) {  
 **if** (!doneNodes.contains(i))  
 **continue**;  
 **for** (**int** j = 0; j < adjacencyMatrix[0].length; j++) {  
 **if** (!doneNodes.contains(j)) {  
 **if** (adjacencyMatrix[i][j] != 0 && adjacencyMatrix[i][j] < minimalWay) {  
 minimalWay = adjacencyMatrix[i][j];  
 minIndexX = j;  
 minIndexY = i;  
 }  
 }  
 }  
 }  
  
 doneNodes.add(minIndexX);  
 doneVerges.add(**new int**[]{minIndexY, minIndexX,minimalWay});  
 }  
  
 System.*out*.println("Necessary verges: ");  
 **int** way = 0;  
 **for**(**int**[] arr : doneVerges){  
 System.*out*.println((arr[0]+1) + "->" + (arr[1]+1) + " weight: " + arr[2]);  
 way+=arr[2];  
 }  
 System.*out*.println("Summary weight: " + way);  
 }  
  
  
 @Override  
 **protected void** preSetIncidenceMatrix() {  
 **super**.preSetIncidenceMatrix();  
 **for** (**int** i = 0; i < **this**.numberOfNodes; i++) {  
 **for** (**int** j = 0; j < **this**.numberOfVerges; j++) {  
 **if** (**this**.verges[j][0] == i + 1 || **this**.verges[j][1] == i + 1)  
 **this**.incidenceMatrix[i][j] = 1;  
  
 **else this**.incidenceMatrix[i][j] = 0;  
 }  
 }  
 }  
  
 @Override  
 **protected void** preSetAdjacencyMatrix() {  
 **super**.preSetAdjacencyMatrix();  
 **for** (**int** i = 0; i < **this**.verges.length; i++) {  
 **this**.adjacencyMatrix[**this**.verges[i][0] - 1][**this**.verges[i][1] - 1] = verges[i][2];  
 **this**.adjacencyMatrix[**this**.verges[i][1] - 1][**this**.verges[i][0] - 1] = verges[i][2];  
 }  
  
 }  
}

Результати роботи програми:



