Лабораторна робота №8.1

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

**Завдання:**

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

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

n m

v1u1w1

.......

Vmumwm

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

2.Визначити максимальний потік в мережі. За допомогою алгоритму Форда-Фалкерсона знайти максимальний потік в мережі. Програма повинна виводити об’єм (розмір) потоку та кількість потоку через кожну ділянку мережі (ребро графу). Програма має самостійно визначати джерело та стік мережі на основі степенів вершин.

Код:

**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** DS\_IP92\_LR81\_MedvedievM {  
  
 **public static void** main(String[] args) **throws** IOException {  
 DirectedGraph graph = **new** DirectedGraph(**new** File("inputs/input.txt"));  
 graph.algorithmByFordFulcerson();  
  
 }  
  
}  
  
**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];  
 }  
  
  
 **protected** 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** DirectedGraph **extends** Graph {  
  
 **protected** DirectedGraph(File file) **throws** FileNotFoundException {  
 **super**(file);  
 }  
  
 **public void** algorithmByFordFulcerson() {  
 **int** [][] startAdjacencyMatrix = getCopyOfMatrix(**this**.adjacencyMatrix);  
 **int** source = findSource();  
 **int** stock = findStock();  
 **int** maxStream = 0;  
 **boolean** flag = **true**;  
 **while** (flag) {  
 ArrayList<**int**[]> notes = **new** ArrayList<>();  
 **int** currentNode = source;  
 notes.add(**new int**[]{Integer.*MAX\_VALUE*, -1});  
 **boolean**[] doneNodes = **new boolean**[numberOfNodes];  
  
 **while** (**true**) {  
 **if** (currentNode == stock) {  
 **int** fp = findMinimalStreamOnWay(notes);  
 **int** node2 = stock;  
 **for** (**int** i = notes.size() - 1; i >= 1; i--) {  
 **int** node1 = notes.get(i)[1];  
 adjacencyMatrix[node1][node2] -= fp;  
 adjacencyMatrix[node2][node1] += fp;  
 node2 = node1;  
 }  
 maxStream += fp;  
 **break**;  
 }  
  
 ArrayList<Integer> availableNodes = getAvailableNodes(currentNode, doneNodes);  
  
 **if** (availableNodes.size() == 0) {  
 **if** (currentNode != source) {  
 **int**[] lastNote = notes.remove(notes.size() - 1);  
 doneNodes[currentNode] = **true**;  
 currentNode = lastNote[1];  
 **continue**;  
 } **else** {  
 flag = **false**;  
 **break**;  
 }  
 }  
 **int** maxStreamIndex = findMaxLastStream(availableNodes, currentNode);  
 notes.add(**new int**[]{adjacencyMatrix[currentNode][availableNodes.get(maxStreamIndex)], currentNode});  
 doneNodes[currentNode] = **true**;  
 currentNode = availableNodes.get(maxStreamIndex);  
 }  
 }  
  
  
 System.*out*.println("Maximum stream: " + maxStream);  
 System.*out*.println("Streams on verges: ");  
 **for** (**int**[] verge : verges) {  
 **int** streamOnVerge = Math.*abs*(startAdjacencyMatrix[verge[0] - 1][verge[1] - 1] - adjacencyMatrix[verge[0] - 1][verge[1] - 1]);  
 System.*out*.println(verge[0] + "->" + verge[1] + " = " + streamOnVerge);  
 }  
 **this**.adjacencyMatrix = startAdjacencyMatrix;  
 }  
  
 **private int** findMinimalStreamOnWay(ArrayList<**int**[]> notes) {  
 **int** minimalStream = notes.get(0)[0];  
 **for** (**int** i = 1; i < notes.size(); i++) {  
 **if** (minimalStream > notes.get(i)[0])  
 minimalStream = notes.get(i)[0];  
 }  
 **return** minimalStream;  
 }  
  
 **private int** findMaxLastStream(ArrayList<Integer> list, **int** node) {  
 **int** stream = adjacencyMatrix[node][list.get(0)];  
 **int** index = 0;  
 **for** (**int** i = 1; i < list.size(); i++) {  
 **if** (stream < adjacencyMatrix[node][list.get(i)]) {  
 stream = adjacencyMatrix[node][list.get(i)];  
 index = i;  
 }  
 }  
  
 **return** index;  
 }  
  
 **private** ArrayList<Integer> getAvailableNodes(**int** node, **boolean**[] doneNodes) {  
 ArrayList<Integer> nodes = **new** ArrayList<>();  
 **for** (**int** i = 0; i < adjacencyMatrix[0].length; i++) {  
 **if** (adjacencyMatrix[node][i] > 0 && !doneNodes[i])  
 nodes.add(i);  
 }  
 **return** nodes;  
 }  
  
  
 **private int** findSource() {  
 **for** (**int** i = 0; i < adjacencyMatrix[0].length; i++) {  
 **int** counter = 0;  
 **for** (**int** j = 0; j < adjacencyMatrix.length; j++) {  
 counter += adjacencyMatrix[j][i];  
 }  
 **if** (counter == 0)  
 **return** i;  
 }  
 **return** -1;  
 }  
  
 **private int** findStock() {  
 **for** (**int** i = 0; i < adjacencyMatrix.length; i++) {  
 **int** counter = 0;  
 **for** (**int** j = 0; j < adjacencyMatrix[0].length; j++) {  
 counter += adjacencyMatrix[i][j];  
 }  
 **if** (counter == 0)  
 **return** i;  
 }  
 **return** -1;  
 }  
  
 **int**[][] getCopyOfMatrix(**int**[][] matrix) {  
 **int**[][] output = **new int**[matrix.length][matrix[0].length];  
 **for** (**int** i = 0; i < matrix.length; i++) {  
 **for** (**int** j = 0; j < matrix[0].length; j++) {  
 output[i][j] = matrix[i][j];  
 }  
 }  
 **return** output;  
 }  
  
 @Override  
 **protected void** preSetAdjacencyMatrix() {  
 **super**.preSetAdjacencyMatrix();  
  
 **for** (**int** i = 0; i < **this**.numberOfVerges; i++) {  
 **this**.adjacencyMatrix[**this**.verges[i][0] - 1][**this**.verges[i][1] - 1] = verges[i][2];  
 }  
  
// System.out.println(matrixToString(adjacencyMatrix, ""));  
  
 }  
}

Результати роботи програми:

