Лабораторна робота №5.1

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

**Завдання:**

Реалізувати програмне застосування (програму), яке виконуєнаступні функції.

При реалізації вважати, що заданий граф є орієнтованим.

**Частина перша:**

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.

2.Визначити найкоротший маршрут між двома вершинами та його довжину.За допомогою алгоритму Дейкстри визначити найкоротшу відстань між двома заданими вершинами (які вводяться користувачем), а також вивести сам знайдений найкоротший маршрут. Програма повинна коректно опрацьовувати факт наявності ребер у графі з від’ємною вагою.

3.Визначити найкоротшу відстань від заданої вершини до всіх інших вершин.За допомогою алгоритму Дейкстри визначити найкоротшу відстань від заданої вершини (вводиться користувачем) до всіх інших вершин графу. Програма виводить на екран список вершин із відповідними значеннями найкоротших відстаней.

**Частина друга:**

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

n m

v1u1w1

v2u2w2

...

....

Vmumwm

1.Визначити найкоротші відстані між усіма парами вершин в графіза допомогою алгоритму Флойда-Уоршела. За допомогою алгоритму Флойда-Уоршола визначити найкоротшу відстань між усіма парами вершин. Програма повинна виводити на екран або у файл знайдену матрицю відстанейта матрицю історії. За запитом користувача програма повинна виводити знайдений найкоротший маршрут для

початкової та кінцевої вершини (вводяться користувачем). Програма повинна коректно опрацьовувати факт наявності в графі циклів з від’ємною вагою.

Код:

**package** com.company;  
  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.Scanner;  
  
**public class** DS\_IP92\_LR51\_MedvedievM {  
 **public static void** main(String[] args) **throws** FileNotFoundException {  
 Graph graph = **new** Graph(**new** File("inputs/input5.txt"));  
 doMenu(graph);  
 }  
  
 **static void** doMenu(Graph graph) {  
 Scanner consoleScanner = **new** Scanner(System.in);  
 System.out.print("Dijkstra(1) or Floyd(2): ");  
 **int** choice = Integer.parseInt(consoleScanner.nextLine());  
 **if** (choice == 1) {  
 **if** (graph.isHasMinusVertex()) {  
 System.out.print("Graph has minus vertex!");  
 **return**;  
 }  
 System.out.print("All distances(1) or only one way(2): ");  
 **int** choice2 = Integer.parseInt(consoleScanner.nextLine());  
 **if** (choice2 == 1) {  
 System.out.print("Input node: ");  
 **int** node = consoleScanner.nextInt();  
 graph.findShortestWaysDijkstra(node);  
 } **else if** (choice2 == 2) {  
 System.out.print("Input two nodes: ");  
 String[] nodes = consoleScanner.nextLine().split(" ");  
 graph.findShortestWayDijkstra(Integer.parseInt(nodes[0]), Integer.parseInt(nodes[1]));  
  
 }  
 } **else if** (choice == 2) {  
 graph.findShortestWaysFloyd();  
 }  
 }  
  
}  
  
  
**class** Graph {  
  
 **private int**[][] vertex;  
 **private int** numberOfNodes, numberOfVertex;  
 **private int**[][] adjacencyMatrix;  
 **private boolean** hasMinusVertex = **false**;  
 **private final int** MAX\_VALUE = Integer.MAX\_VALUE / 3;  
  
 Graph(File file) **throws** FileNotFoundException {  
 readFile(file);  
 **this**.adjacencyMatrix = getAdjacencyMatrix();  
 }  
  
 **public boolean** isHasMinusVertex() {  
 **return** hasMinusVertex;  
 }  
  
 **private void** readFile(File file) **throws** FileNotFoundException {  
 Scanner scanner = **new** Scanner(file);  
 **this**.numberOfNodes = scanner.nextInt();  
 **this**.numberOfVertex = scanner.nextInt();  
 **this**.vertex = **new int**[numberOfVertex][3];  
 **for** (**int** i = 0; i < numberOfVertex; i++) {  
 **for** (**int** j = 0; j < 3; j++) {  
 vertex[i][j] = scanner.nextInt();  
 }  
 **if** (!hasMinusVertex && vertex[i][2] < 0)  
 hasMinusVertex = **true**;  
 }  
 scanner.close();  
 }  
  
// public void findShortestWays(int node) {  
// node--;  
// findShortestWaysFloyd();  
// int[] distances = findArrayOfShortestWaysDijkstra(node);  
// System.out.println("Minimal distances: ");  
// for (int i = 0; i < distances.length; i++) {  
// if (i != node)  
// System.out.println((node + 1) + "->" + (i + 1) + ": " + distances[i]);  
// }  
// }  
  
  
 **public void** findShortestWaysFloyd() {  
 **int**[][] matrixA = **new int**[numberOfNodes][numberOfNodes];  
 **int**[][] matrixP = **new int**[numberOfNodes][numberOfNodes];  
  
 **for** (**int** i = 0; i < numberOfNodes; i++) {  
 **for** (**int** j = 0; j < numberOfNodes; j++) {  
 **if** (i != j)  
 matrixP[i][j] = i + 1;  
 }  
 }  
  
 **for** (**int** i = 0; i < numberOfNodes; i++)  
 matrixA[i] = Arrays.copyOf(**this**.adjacencyMatrix[i], numberOfNodes);  
  
 **for** (**int** k = 0; k < numberOfNodes; k++) {  
 **for** (**int** i = 0; i < numberOfNodes; i++) {  
 **for** (**int** j = 0; j < numberOfNodes; j++) {  
 **if** (matrixA[i][j] > matrixA[i][k] + matrixA[k][j]) {  
 matrixA[i][j] = matrixA[i][k] + matrixA[k][j];  
 matrixP[i][j] = matrixP[k][j];  
// for (int l = 0; l < numberOfNodes; l++)  
// System.out.println(Arrays.toString(matrixP[l]));  
// System.out.println();  
 }  
 }  
 }  
// printMatrix(matrixP);  
 }  
 **boolean** hasMinusCycles = **false**;  
 **for** (**int** i = 0; i < numberOfNodes; i++) {  
 **if** (matrixA[i][i] < 0) {  
 hasMinusCycles = **true**;  
 **break**;  
 }  
  
 }  
 **if** (hasMinusCycles) {  
 System.out.println("Graph has minus cycles!");  
 **return**;  
 }  
  
 System.out.println("Distance matrix: ");  
 printMatrix(matrixA);  
  
 System.out.println("History matrix: ");  
 printMatrix(matrixP);  
  
 Scanner consoleScanner = **new** Scanner(System.in);  
 System.out.print("Do you want to find the way? (yes/no): ");  
 String choice = consoleScanner.nextLine();  
 **if** (choice.equals("yes")) {  
 System.out.print("Input two nodes: ");  
 String[] nodes = consoleScanner.nextLine().split(" ");  
 **int** minimalDistance = matrixA[Integer.parseInt(nodes[0]) - 1][Integer.parseInt(nodes[1]) - 1];  
 **if** (minimalDistance == MAX\_VALUE) {  
 System.out.println("There\'s no way between these nodes!");  
 **return**;  
 }  
  
 System.out.println("Minimal distance: " + minimalDistance);  
 System.out.println("Way: " + Arrays.toString(findWay(matrixP, Integer.parseInt(nodes[0]), Integer.parseInt(nodes[1]))));  
 }  
 }  
  
 **private void** printMatrix(**int**[][] matrix) {  
 **for** (**int** i = 0; i < matrix.length; i++) {  
 System.out.print("[ ");  
 **for** (**int** j = 0; j < matrix[0].length; j++) {  
 System.out.print(((matrix[i][j] == MAX\_VALUE) ? " ∞" : ((matrix[i][j] > 9) ? (matrix[i][j]) : (" " + matrix[i][j]))) + " ");  
 }  
 System.out.print("]\n");  
 }  
 System.out.println();  
 }  
  
  
 **private int**[] findWay(**int**[][] matrixP, **int** node1, **int** node2) {  
  
 ArrayList<Integer> nodesOnWay = **new** ArrayList<>();  
 nodesOnWay.add(node2);  
 node1--;  
 node2--;  
  
 **while** (node2 != node1) {  
  
 **int** currentNode = matrixP[node1][node2];  
// System.out.println("NOde: " + currentNode + " " + node1 + " " + node2);  
 nodesOnWay.add(currentNode);  
// System.out.println(nodesOnWay.toString());  
 **if** (currentNode == node1)  
 **break**;  
 node2 = currentNode - 1;  
 }  
// System.out.println(nodesOnWay.toString());  
 **return** getInvertedArrayFromList(nodesOnWay);  
  
 }  
  
 **private int**[] getInvertedArrayFromList(ArrayList<Integer> list) {  
 **int**[] arr = **new int**[list.size()];  
// System.out.println(list.toString());  
 **for** (**int** i = list.size() - 1; i >= 0; i--) {  
 arr[list.size() - i - 1] = list.get(i);  
 }  
// System.out.println(Arrays.toString(arr));  
 **return** arr;  
 }  
  
 **public void** findShortestWayDijkstra(**int** node1, **int** node2) {  
 node1--;  
 node2--;  
 **int**[][] res = findArrayOfShortestWaysDijkstra(**this**.adjacencyMatrix, node1);  
 **int**[] distances = res[0];  
 **int**[] arrayP = res[1];  
 **if** (distances[node2] == MAX\_VALUE) {  
 System.out.println("There\'s no way between these nodes!");  
 **return**;  
 }  
// System.out.println(Arrays.toString(arrayP));  
 **int**[] way = findWay(arrayP, node2 + 1);  
// for (int i = 0; i < way.length; i++)  
// way[i]++;  
 System.out.println("Minimal distance: " + distances[node2]);  
 System.out.println("Way: " + Arrays.toString(way));  
 }  
  
 **public void** findShortestWaysDijkstra(**int** node) {  
 node--;  
 **int**[] distances = findArrayOfShortestWaysDijkstra(**this**.adjacencyMatrix, node)[0];  
 System.out.println("Minimal distances: ");  
 **for** (**int** i = 0; i < distances.length; i++) {  
 **if** (i != node)  
 System.out.println((node + 1) + "->" + (i + 1) + ": " + ((distances[i] == MAX\_VALUE) ? "∞" : distances[i]));  
 }  
 }  
  
  
 **int**[] findWay(**int**[] arrayP, **int** node2) {  
 ArrayList<Integer> list = **new** ArrayList<>();  
 **while** (node2 != -1) {  
 list.add(node2);  
 node2 = arrayP[node2 - 1];  
 }  
 **return** getInvertedArrayFromList(list);  
 }  
  
 **private boolean** hasFalses(**boolean**[] arr) {  
 **for** (**int** i = 0; i < arr.length; i++)  
 **if** (!arr[i])  
 **return true**;  
  
 **return false**;  
 }  
  
  
 **private int**[][] findArrayOfShortestWaysDijkstra(**int**[][] adjacencyMatrix, **int** node) {  
  
 **int**[] arrayP = **new int**[numberOfNodes];  
 Arrays.fill(arrayP, -1);  
 **for** (**int** i = 0; i < numberOfNodes; i++) {  
 **if** (i != node && adjacencyMatrix[node][i] != MAX\_VALUE) {  
 arrayP[i] = node + 1;  
 }  
 }  
  
 **boolean**[] doneNodes = **new boolean**[adjacencyMatrix.length];  
 Arrays.fill(doneNodes, **false**);  
  
 **int**[] distanceArray = **new int**[numberOfNodes];  
 Arrays.fill(distanceArray, MAX\_VALUE);  
  
 distanceArray[node] = 0;  
 **while** (hasFalses(doneNodes)) {  
 doneNodes[node] = **true**;  
// System.out.println(Arrays.toString(distanceArray) + " " + node + " " + Arrays.toString(doneNodes));  
 **for** (**int** i = 0; i < distanceArray.length; i++) {  
 **if** (i != node && adjacencyMatrix[node][i] != MAX\_VALUE) {  
 **if** (distanceArray[i] > (distanceArray[node] + adjacencyMatrix[node][i])) {  
 distanceArray[i] = distanceArray[node] + adjacencyMatrix[node][i];  
 arrayP[i] = node + 1;  
 }  
  
 }  
 }  
  
 **int** closestNode = -1;  
 **for** (**int** i = 0; i < distanceArray.length; i++) {  
 **if** (i != node && !doneNodes[i]) {  
 **if** (closestNode == -1 || distanceArray[i] < distanceArray[closestNode])  
 closestNode = i;  
 }  
 }  
  
 **if** (closestNode == -1)  
 **break**;  
  
 node = closestNode;  
// System.out.println(Arrays.toString(distanceArray));  
  
 }  
 **return new int**[][]{distanceArray, arrayP};  
 }  
  
  
 **private int**[][] getAdjacencyMatrix() {  
 **int**[][] adjacencyMatrix = **new int**[**this**.numberOfNodes][**this**.numberOfNodes];  
 **for** (**int** i = 0; i < adjacencyMatrix.length; i++) {  
 Arrays.fill(adjacencyMatrix[i], MAX\_VALUE);  
 }  
 **for** (**int**[] ints : **this**.vertex) {  
 adjacencyMatrix[ints[0] - 1][ints[1] - 1] = ints[2];  
 }  
 **for** (**int** i = 0; i < numberOfNodes; i++)  
 adjacencyMatrix[i][i] = 0;  
 **return** adjacencyMatrix;  
 }  
  
  
}

Результати роботи програми:

