Лабораторна робота №7.2

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

**Завдання:**

Реалізувати програмне застосування (програму), яке виконує наступні функції. Причому на вхід програми подається вхідний файл з описом графу, зі структурою, яка вказана у лабораторній роботіNo1 «Представлення графів».

Розв’язати задачу побудови досконалих паросполучень. Програма визначає наявність досконалих паросполученьдля заданого двочасткового графута виводить їх на екрану вигляді матриць, що мають розмір матриці суміжності, де \* - на позиції, де немає ребрав початковому графі, 1 – на позиції, що підходить для певного досконалого паросполучення,.0 - на позиції, що не підходить для певного досконалого паросполучення(зразок на початку відео лекції за 2020.04.28).

Код:

**package** com.company;  
  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.Scanner;  
  
**public class** DS\_IP92\_LR72\_MedvedievM {  
  
 **public static void** main(String[] args) **throws** IOException {  
 DicotyledonousGraph graph = **new** DicotyledonousGraph(**new** File("inputs/input.txt"));  
 graph.printPairs();  
// System.out.println(Graph.matrixToString(graph.adjacencyMatrix, ""));  
// graph.printPairs();  
 }  
  
}  
  
**abstract class** Graph {  
 **protected int**[][] verges;  
 **protected int** numberOfNodes, numberOfVerges;// n вершин, m ребер  
 **protected int**[][] incidenceMatrix, adjacencyMatrix;  
 **protected** ArrayList<Integer> leftNodes = **new** ArrayList<>(), rightNodes = **new** ArrayList<>();  
  
 **protected** Graph(File file) **throws** FileNotFoundException {  
 parseFile(file);  
 preSetAdjacencyMatrix();  
 }  
  
 **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][2];  
 **for** (**int** i = 0; i < **this**.numberOfVerges; i++) {  
 verges[i][0] = fileScanner.nextInt();  
 **if** (!leftNodes.contains(verges[i][0]))  
 leftNodes.add(verges[i][0]);  
 verges[i][1] = fileScanner.nextInt();  
 **if** (!rightNodes.contains(verges[i][1]))  
 rightNodes.add(verges[i][1]);  
 }  
 }  
  
  
  
 **protected void** preSetAdjacencyMatrix() {  
 **this**.adjacencyMatrix = **new int**[**this**.numberOfNodes][**this**.numberOfNodes];  
 }  
  
  
  
 **protected static** String matrixToString(**int**[][] matrix, String extraText) {  
 StringBuilder outputText = **new** StringBuilder(extraText + "\n");  
  
 **for** (**int**[] line : matrix) {  
 **for** (**int** j = 0; j < matrix[0].length; j++) {  
 String text = (line[j] != -1) ? Integer.*toString*(line[j]) : "\*";  
 outputText.append(text).append(" ");  
  
 }  
 outputText.append("\n");  
 }  
 **return** outputText.toString();  
 }  
  
}  
  
**class** DicotyledonousGraph **extends** Graph {  
 **protected** DicotyledonousGraph(File file) **throws** FileNotFoundException {  
 **super**(file);  
// System.out.println(getTypeOfConnectedness());  
// System.out.println(matrixToString(this.adjacencyMatrix,""));  
 }  
  
 **protected void** preSetAdjacencyMatrix() {  
 **this**.adjacencyMatrix = **new int**[leftNodes.size()][rightNodes.size()];  
 **for** (**int** i = 0; i < **this**.numberOfVerges; i++)  
 **this**.adjacencyMatrix[**this**.verges[i][0] - 1][**this**.verges[i][1] - 1 - leftNodes.size()] = 1;  
 }  
  
 **public void** printPairs() {  
 System.*out*.println("Possible Matrixs with perfect pairs: ");  
 **int**[][] newAdjacencyMatrix = getCopyOfMatrix(**this**.adjacencyMatrix);  
 **for** (**int** i = 0; i < newAdjacencyMatrix.length; i++)  
 **for** (**int** j = 0; j < newAdjacencyMatrix[0].length; j++)  
 newAdjacencyMatrix[i][j]--;  
// System.out.println(matrixToString(newAdjacencyMatrix, ""));  
 printPairsRecurs(newAdjacencyMatrix, 0);  
 }  
 **private void** printPairsRecurs(**int**[][] adjMatrix, **int** level) {  
 **if** (level >= adjMatrix.length) {  
 printAdjacencyMatrix(adjMatrix);  
 **return**;  
 }  
 **boolean** flag = **false**;  
 **for** (**int** i = 0; i < adjMatrix[0].length; i++) {  
 **if** (adjMatrix[level][i] != -1) {  
 **boolean** hasOnes = **false**;  
 **for** (**int** j = 0; j < level; j++) {  
 **if** (adjMatrix[j][i] == 1) {  
 hasOnes = **true**;  
 **break**;  
 }  
 }  
  
 **if** (!hasOnes) {  
 flag = **true**;  
 **int**[][] newAdjMatrix = getCopyOfMatrix(adjMatrix);  
 newAdjMatrix[level][i] = 1;  
// System.out.println(matrixToString(adjMatrix, "Adj"));  
// System.out.println(matrixToString(newAdjMatrix, "New Adj"));  
 printPairsRecurs(newAdjMatrix, level + 1);  
 }  
 }  
 }  
 }  
 **private int**[][] getCopyOfMatrix(**int**[][] matrix) {  
 **int**[][] output = **new int**[matrix.length][matrix[0].length];  
 **for** (**int** i = 0; i < matrix.length; i++) {  
 System.*arraycopy*(matrix[i], 0, output[i], 0, matrix.length);  
 }  
 **return** output;  
 }  
 **private void** printAdjacencyMatrix(**int** [][]adjMatrix) {  
// System.out.println(Arrays.toString(rightNodes.toArray()));  
 StringBuilder text = **new** StringBuilder(" ");  
 **for** (Integer rightNode : rightNodes) {  
 text.append(rightNode).append(" ");  
 }  
 text.append("\n");  
 **for**(**int** i=0;i<leftNodes.size();i++){  
 text.append(leftNodes.get(i)).append(" ");  
 **for**(**int** j=0;j<rightNodes.size();j++){  
 text.append((adjMatrix[i][j] != -1) ? adjMatrix[i][j] : "\*").append(" ");  
 }  
 text.append("\n");  
 }  
 System.*out*.println(text);  
 }  
}

Результати роботи програми:

