Лабораторна робота №6.2

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».

1.Розв’язати задачу розфарбування графів. Програма визначає мінімальну кількість кольорів, в який можна розфарбувати граф, та виводить на екран список вершин із відповідними призначеними кольорами.

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\_LR62\_MedvedievM {  
  
 **public static void** main(String[] args) **throws** IOException {  
 UndirectedGraph graph = **new** UndirectedGraph(**new** File("inputs/input.txt"));  
 graph.colourGraph();  
  
 }  
  
}  
  
**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();  
 }  
  
 **protected** Graph() {  
 }  
  
 **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();  
 verges[i][1] = 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** 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();  
 }  
  
  
 **protected** UndirectedGraph(**int**[][] adjacencyMatrix) {  
 **this**.adjacencyMatrix = adjacencyMatrix;  
 }  
  
 **public void** colourGraph() {  
  
 **int**[] colours = hromaticRecurs(**this**);  
 **int** maxx = -1;  
 **for**(**int** i=0;i<colours.length;i++){  
 **if**(colours[i]>maxx)  
 maxx = colours[i];  
 }  
 System.*out*.println("Minimal number of colours: " + maxx);  
 **for**(**int** i=0;i<colours.length;i++){  
 System.*out*.println((i+1) + " -> " + colours[i]);  
 }  
  
 }  
  
  
  
 **int** getDegreeOfNode(**int** node, **int** [][] adj){  
 **int** output = 0;  
 **for**(**int** i=0;i<adj.length;i++)  
 output+=adj[node][i];  
 **return** output;  
 }  
  
  
 **int**[] hromaticRecurs(UndirectedGraph currentGraph) {  
 **int**[][] currentAdjacencyMatrix = currentGraph.getAdjacencyMatrix();  
 **int** indexX=-1, indexY=-1;  
 **int** sumOfDegree = 0;  
 **boolean** flag = **false**;  
// System.out.println(matrixToString(currentAdjacencyMatrix, "Current: "));  
 **for** (**int** i = 0; i < currentAdjacencyMatrix.length; i++) {  
 **for** (**int** j = 0; j < currentAdjacencyMatrix[0].length; j++) {  
 **if** (currentAdjacencyMatrix[i][j] == 0) {  
 **int** sum = getDegreeOfNode(i,currentAdjacencyMatrix) + getDegreeOfNode(j,currentAdjacencyMatrix);  
 **if**(sum>sumOfDegree){  
 indexX = j;  
 indexY = i;  
 sumOfDegree = sum;  
 }  
 flag = **true**;  
 }  
 }  
 }  
  
  
 **if** (!flag) {  
 **int**[] colours = **new int**[currentAdjacencyMatrix.length];  
 **for** (**int** k = 0; k < colours.length; k++)  
 colours[k] = k + 1;  
 **return** colours;  
 }  
  
  
// UndirectedGraph leftGraph = getConnectedGraph(currentGraph, indexY, indexX);  
 UndirectedGraph rightGraph = getMergedGraph(currentGraph, indexY, indexX);  
// System.out.println(matrixToString(leftGraph.getAdjacencyMatrix(),"Left"));  
// hromaticRecurs(leftGraph,list);  
// System.out.println(matrixToString(rightGraph.getAdjacencyMatrix(),"Right"));  
  
 **int**[] getColours = hromaticRecurs(rightGraph);  
// System.out.println(Arrays.toString(getColours));  
// System.out.println("i: " + indexY + ", j: " + indexX);  
 **int**[] newColours = **new int**[getColours.length + 1];  
 **int** d = 0;  
 **for** (**int** k = 0; k < newColours.length; k++) {  
 **if** (k == indexX) {  
 newColours[k] = newColours[indexY];  
 d = 1;  
 **continue**;  
 }  
 newColours[k] = getColours[k - d];  
 }  
  
 **return** newColours;  
 }  
  
 **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;  
 }  
  
 UndirectedGraph getConnectedGraph(UndirectedGraph currentGraph, **int** i, **int** j) {  
 **int**[][] newAdjacencyMatrix = getCopyOfMatrix(currentGraph.getAdjacencyMatrix());  
 newAdjacencyMatrix[i][j] = 1;  
 newAdjacencyMatrix[j][i] = 1;  
 **return new** UndirectedGraph(newAdjacencyMatrix);  
 }  
  
 UndirectedGraph getMergedGraph(UndirectedGraph currentGraph, **int** i, **int** j) {  
 **int**[][] currentAdjacencyMatrix = getCopyOfMatrix(currentGraph.getAdjacencyMatrix());  
// System.out.println(matrixToString(currentAdjacencyMatrix, "Current: "));  
 **for** (**int** k = 0; k < currentAdjacencyMatrix[0].length; k++) {  
 **if** (currentAdjacencyMatrix[j][k] == 1)  
 currentAdjacencyMatrix[i][k] = 1;  
 **if** (currentAdjacencyMatrix[k][j] == 1)  
 currentAdjacencyMatrix[k][i] = 1;  
 }  
// System.out.println(matrixToString(currentAdjacencyMatrix, "Current2: "));  
// System.out.println("i: " + i + ", j: " + j);  
  
 **int**[][] newAdjacencyMatrix = getMatrixWithoutRowAndColumn(currentAdjacencyMatrix, j);  
// System.out.println(matrixToString(newAdjacencyMatrix, "New: "));  
 **return new** UndirectedGraph(newAdjacencyMatrix);  
  
 }  
  
 **private int**[][] getMatrixWithoutRowAndColumn(**int**[][] matrix, **int** n) {  
 **int**[][] output = **new int**[matrix.length - 1][matrix[0].length - 1];  
 **int** deltaX = 0, deltaY = 0;  
 **for** (**int** i = 0; i < output.length; i++) {  
 **if** (i == n)  
 deltaY = 1;  
  
 **for** (**int** j = 0; j < output[0].length; j++) {  
 **if** (j == n)  
 deltaX = 1;  
  
 output[i][j] = matrix[i + deltaY][j + deltaX];  
 }  
 deltaX = 0;  
 }  
  
 **return** output;  
 }  
  
 @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**.numberOfVerges; i++) {  
 **this**.adjacencyMatrix[**this**.verges[i][0] - 1][**this**.verges[i][1] - 1] = 1;  
 **this**.adjacencyMatrix[**this**.verges[i][1] - 1][**this**.verges[i][0] - 1] = 1;  
 }  
 **for** (**int** i = 0; i < adjacencyMatrix.length; i++)  
 adjacencyMatrix[i][i] = 1;  
 }  
}

Результати роботи програми:

