Лабораторна робота №7.1

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 побудови кода Прюфера. Програма визначає код Прюферадля заданого деревата виводить його на екран.
2. Розв’язати задачу побудовидерева з коду Прюфера. Програма визначає матрицю суміжності для заданого коду Прюфера (який задається на вхіду у вигляді рядка

1 3 3 1 4 7 4) та виводить матрицю суміжності на екран.

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\_LR71\_MedvedievM {  
  
 **public static void** main(String[] args) **throws** IOException {  
 UndirectedGraph graph = **new** UndirectedGraph(**new** File("inputs/input2.txt"));  
 Scanner scanner = **new** Scanner(System.*in*);  
 System.*out*.print("Encode(1) or decode(2): ");  
 **int** choice = Integer.*parseInt*(scanner.nextLine());  
 **if**(choice==1)  
 graph.encodePrufer();  
 **else if**(choice == 2) {  
 System.*out*.print("Input code: ");  
 **int** [] code = *mapStringArray*(scanner.nextLine().split(" "));  
 System.*out*.println(UndirectedGraph.*matrixToString*(UndirectedGraph.*decodePrufer*(code).adjacencyMatrix,"Adjacency matrix: "));  
 }  
 }  
 **static int** [] mapStringArray(String [] array){  
 **int** [] output = **new int**[array.length];  
 **for** (**int** i=0;i<array.length;i++) {  
 output[i] = Integer.*parseInt*(array[i]);  
 }  
 **return** output;  
 }  
}  
  
**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 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();  
 }  
  
 **protected** UndirectedGraph(**int**[][] verges, **int** numberOfNodes) {  
 **this**.verges = verges;  
 **this**.numberOfNodes = numberOfNodes;  
 preSetAdjacencyMatrix();  
 }  
  
 **public static** UndirectedGraph decodePrufer(**int**[] code) {  
 **int** numberOfNodes = code.length + 2;  
 **boolean**[] doneNodes = **new boolean**[numberOfNodes];  
 **int**[][] verges = **new int**[code.length + 1][2];  
  
 **for** (**int** i = 0; i < code.length; i++) {  
 **int** currentNode = code[i];  
 **int** node = -1;  
 **for** (**int** j = 0; j < numberOfNodes; j++) {  
 **if** (doneNodes[j])  
 **continue**;  
 **boolean** flag = **true**;  
 **for** (**int** k = i; k < code.length; k++) {  
 **if** (code[k] == j + 1) {  
 flag = **false**;  
 **break**;  
 }  
 }  
 **if** (flag) {  
 node = j + 1;  
 doneNodes[j] = **true**;  
 **break**;  
 }  
  
 }  
  
 verges[i][0] = currentNode;  
 verges[i][1] = node;  
 }  
 **boolean** first = **true**;  
 **for** (**int** i = 0; i < doneNodes.length; i++) {  
 **if** (!doneNodes[i]) {  
 **if** (first) {  
 first = **false**;  
 verges[verges.length - 1][0] = i + 1;  
 } **else** verges[verges.length - 1][1] = i + 1;  
 }  
 }  
 **return new** UndirectedGraph(verges,numberOfNodes);  
 }  
  
 **public void** encodePrufer() {  
 ArrayList<Integer> code = encodeRecurs(**this**, **new** ArrayList<>());  
 System.*out*.print("Code: ");  
 **for** (**int** i = 0; i < code.size(); i++) {  
 System.*out*.print(code.get(i) + " ");  
 }  
 }  
  
 **private** ArrayList<Integer> encodeRecurs(UndirectedGraph currentGraph, ArrayList<Integer> code) {  
 **final int**[][] currentAdjacencyMatrix = currentGraph.adjacencyMatrix;  
 **int** indexX = -1, indexY = -1;  
 **for** (**int** i = 0; i < currentGraph.numberOfNodes; i++) {  
 **int** counter = 0;  
 **int** x = -1;  
 **for** (**int** j = 0; j < currentGraph.numberOfNodes; j++) {  
 **if** (currentAdjacencyMatrix[i][j] == 1 && i != j) {  
 counter++;  
 x = j;  
 }  
 }  
 **if** (counter == 1) {  
 indexY = i;  
 indexX = x;  
 **break**;  
 }  
 }  
// System.out.println(indexY + " " + indexX);  
// System.out.println(matrixToString(currentAdjacencyMatrix,"Current: "));  
  
 code.add(indexX + 1);  
  
 **final int**[][] currentVerges = currentGraph.verges;  
 **int**[][] newVerges = **new int**[currentVerges.length - 1][2];  
 **int** temp = 0;  
 **for** (**int** i = 0; i < newVerges.length; i++) {  
 **if** (currentVerges[i][0] == (indexY + 1) || currentVerges[i][1] == (indexY + 1)) {  
 temp++;  
 }  
 newVerges[i] = currentVerges[i + temp].clone();  
 }  
  
  
 **if** (newVerges.length == 1)  
 **return** code;  
  
 **return** encodeRecurs(**new** UndirectedGraph(newVerges, currentGraph.numberOfNodes), code);  
  
 }  
 @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] = 1;  
 **this**.adjacencyMatrix[**this**.verges[i][1] - 1][**this**.verges[i][0] - 1] = 1;  
 }  
 **for** (**int** i = 0; i < adjacencyMatrix.length; i++)  
 **this**.adjacencyMatrix[i][i] = 1;  
 }  
}

Результати роботи програми:



