Лабораторна робота №6.1

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

**Завдання:**

1.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». При реалізації вважати, що заданий граф є неорієнтованим.

1)Знайти ейлерів цикл в графі. Програма виводить на екран знайдений ейлерів цикл або сповіщає про його відсутність у графі.

2)Знайти ейлерів маршрут в графі. У випадку, якщо ейлеревого циклув графі не існує, то спробувати знайти ейлерів маршрут. Програма виводить на екран знайдений ейлерів маршрут, якщо він є, або сповіщає про його відсутність.

2.Реалізувати програмне застосування (програму), яке виконує наступні функції. Причому на вхід програми подається вхідний файл з описом графу, зі структурою, яка вказана у практичному завданні No1 «Представлення графів». При реалізації вважати, що заданий граф є неорієнтованим.

1.Знайти гамільтонів цикл в графі. Програма виводить на екран знайдений гамільтонів цикл або сповіщає про його відсутність у графі.

2.Знайти гамільтонів маршрут в графі. У випадку, якщо гамільтонового циклу в графі не існує, то спробувати знайти гамільтонів маршрут. Програма виводить на екран знайдений гамільтонів маршрут, якщо він є, або сповіщає про його відсутність.

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\_LR61\_MedvedievM {  
  
 **public static void** main(String[] args) **throws** IOException {  
 UndirectedGraph graph = **new** UndirectedGraph(**new** File("inputs/input.txt"));  
 Scanner scanner = **new** Scanner(System.in);  
 System.out.print("Euler(1) or Gamilton(2): ");  
 **int** choice = scanner.nextInt();  
 **if**(choice == 1)  
 graph.findEulerPath();  
 **else if**(choice == 2)  
 graph.findGamiltonPath();  
 }  
  
  
}  
  
**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][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**[][] getIncidenceMatrix() {  
 **return** incidenceMatrix;  
 }  
  
 **public int**[][] getAdjacencyMatrix() {  
 **return** adjacencyMatrix;  
 }  
  
  
 **private** String matrixToString(**int**[][] matrix, String extraText) {  
 String outputText = extraText + "\n";  
  
 **for** (**int** i = 0; i < matrix.length; i++) {  
 **for** (**int** j = 0; j < matrix[0].length; j++)  
 outputText += ((matrix[i][j] >= 0) ? " " : "") + matrix[i][j] + " ";  
  
 outputText += "\n";  
 }  
 **return** outputText;  
 }  
  
}  
  
**class** UndirectedGraph **extends** Graph {  
  
 **protected** UndirectedGraph(File file) **throws** FileNotFoundException {  
 **super**(file);  
// findEulerPath();  
// findGamiltonPath();  
 }  
  
  
 **public void** findEulerPath() {  
  
 **int** odd = 0;  
 **int** even = 0;  
 **int** v = 0;  
  
 **for** (**int** i = 0; i < numberOfNodes; i++) {  
 **int** numberOfIncidentVertex = 0;  
 **for** (**int** j = 0; j < numberOfVerges; j++) {  
 numberOfIncidentVertex += incidenceMatrix[i][j];  
 }  
 **if** (numberOfIncidentVertex % 2 == 0)  
 even++;  
 **else** {  
 odd++;  
 v = i;  
 }  
 }  
  
 **if** (odd != 0) {  
 System.out.println("No Euler Cicles!");  
 **if** (odd != 2) {  
 System.out.println("No Euler Ways!");  
 **return**;  
 }  
 }  
  
  
 MyStack output = **new** MyStack(2 \* numberOfNodes);  
 MyStack stack = **new** MyStack(2 \* numberOfNodes);  
  
 stack.put(v);  
 **while** (stack.getCurrentNode() != -1) {  
 v = stack.getCurrentNode();  
// System.out.println(stack.getString() + " " + output.getString());  
  
 **int** nextNode = -1;  
 **for** (**int** i = 0; i < numberOfVerges; i++) {  
  
 **if** (incidenceMatrix[v][i] == 1) {  
 **int**[] vertex = verges[i];  
 **if** (vertex[0] == v + 1)  
 nextNode = vertex[1] - 1;  
 **else** nextNode = vertex[0] - 1;  
  
 **for** (**int** j = 0; j < numberOfNodes; j++)  
 incidenceMatrix[j][i] = 0;  
  
 **break**;  
 }  
 }  
  
 **if** (nextNode == -1) {  
 output.put(v);  
 stack.removeLast();  
 **continue**;  
 }  
  
 stack.put(nextNode);  
 }  
 System.out.println("Way: [ " + output.getString() + "]");  
 }  
  
 ArrayList<String> paths = **new** ArrayList<>();  
 ArrayList<String> cicles = **new** ArrayList<>();  
  
 **public void** findGamiltonPath(){  
 MyStack stack = **new** MyStack(numberOfNodes+1);  
 **boolean** [] doneNodes = **new boolean**[numberOfNodes];  
 gamilton(0,doneNodes,stack);  
  
 **if**(cicles.size() == 0){  
 System.out.println("There\'s no cicles!");  
 **if**(paths.size() == 0){  
 System.out.println("There\'s no paths!");  
 }  
 **else** System.out.println("Variant of path: [ " + paths.get(0) + "]");  
 }  
 **else** System.out.println("Variant of cicle: [ "+cicles.get(0) + "]");  
 }  
  
 **void** gamilton(**int** v, **boolean** [] doneNodes, MyStack stack){  
  
 stack.put(v);  
 doneNodes[v] = **true**;  
 **for**(**int** i=0;i<numberOfNodes;i++){  
 **if**(i!=v && adjacencyMatrix[v][i] == 1 && !doneNodes[i]){  
 **boolean** [] newDoneNodes = Arrays.copyOf(doneNodes,doneNodes.length);  
 gamilton(i,newDoneNodes,stack.getCopy());  
 }  
 }  
  
 **if**(!hasFalses(doneNodes) && stack.lastIndex == doneNodes.length-1) {  
 **if**(adjacencyMatrix[stack.getCurrentNode()][stack.getFirstNode()]==1) {  
 stack.put(stack.getFirstNode());  
 cicles.add(stack.getString());  
 }  
 **else** paths.add(stack.getString());  
 }  
  
 }  
  
 **boolean** hasFalses(**boolean** [] array){  
 **boolean** has = **false**;  
 **for**(**int** i=0;i<array.length;i++){  
 **if**(array[i] = **false**){  
 has = **true**;  
 **break**;  
 }  
 }  
 **return** has;  
 }  
  
  
 @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;  
 }  
 }  
}  
  
**class** MyStack {  
 **int**[] mStack;  
 **int** lastIndex = -1;  
  
 MyStack(**int** length) {  
 mStack = **new int**[length];  
 //mStack[0] = first;  
 }  
  
 **int** getLength() {  
 **return** mStack.length;  
 }  
  
 **int** getCurrentNode() {  
 **if** (lastIndex >= 0)  
 **return** mStack[lastIndex];  
 **else return** -1;  
 }  
 **int** getFirstNode(){  
 **if**(lastIndex>=0)  
 **return** mStack[0];  
 **return** -1;  
 }  
  
  
 **void** put(**int** node) {  
 lastIndex++;  
 mStack[lastIndex] = node;  
 }  
  
 **boolean** removeLast() {  
 **if** (lastIndex != -1) {  
 mStack[lastIndex] = 0;  
 lastIndex--;  
 **return false**;  
 }  
 **return true**;  
  
 }  
  
 MyStack getCopy(){  
 MyStack output = **new** MyStack(mStack.length);  
 **for**(**int** i=0;i<=lastIndex;i++){  
 output.put(mStack[i]);  
 }  
 **return** output;  
 }  
  
 String getString() {  
 StringBuilder output = **new** StringBuilder();  
 **for** (**int** i = 0; i <= lastIndex; i++) {  
 output.append(mStack[i] + 1).append(" ");  
 }  
 **return** output.toString();

Результати роботи програми:



