public class Edge {  
  
  
  
 private int vertexOne;  
 private int vertexTwo;  
 private int weight;  
  
 public int getVertexOne() {  
 return vertexOne;  
 }  
  
 public void setVertexOne(int vertexOne) {  
 this.vertexOne = vertexOne;  
 }  
  
 public int getVertexTwo() {  
 return vertexTwo;  
 }  
  
 public void setVertexTwo(int vertexTwo) {  
 this.vertexTwo = vertexTwo;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public void setWeight(int weight) {  
 this.weight = weight;  
 }  
  
 public Edge( int weight,int vertexOne, int vertexTwo) {  
 this.vertexOne = vertexOne;  
 this.vertexTwo = vertexTwo;  
 this.weight = weight;  
 }  
  
  
 @Override  
 public boolean equals(Object obj) {  
 Edge e=(Edge) obj;  
 return ((e.getVertexOne()==this.getVertexOne()||e.getVertexOne()==this.getVertexTwo()) &&  
 (e.getVertexTwo()==this.getVertexOne()||e.getVertexTwo()==this.getVertexTwo()));  
 }  
  
  
}

import java.util.ArrayList;  
  
public class Graph {  
  
  
 private ArrayList<Edge> edges = new ArrayList<>();  
 private ArrayList<Integer> vertices = new ArrayList<>();  
  
  
 public ArrayList<Edge> getEdges() {  
 return edges;  
 }  
  
 public void setEdges(ArrayList<Edge> edges) {  
 this.edges = edges;  
 }  
  
 public ArrayList<Integer> getVertices() {  
 return vertices;  
 }  
  
 public void setVertices(ArrayList<Integer> vertices) {  
 this.vertices = vertices;  
 }  
  
public Graph(){}  
 public Graph(ArrayList<Edge> edges, int noOfVertices) {  
  
// System.out.println(" 4) Edges and number of vertices are adding from main method " +noOfVertices);  
  
 this.edges = edges;  
  
 for (int i = 1; i <= noOfVertices; i++) {  
  
 vertices.add(i);  
  
 }  
  
  
 }  
  
  
 public int getVertexWeight(int vertex) {  
  
// System.out.println("%%entered getvertexweight method with value " + vertex );  
  
 int cpt = 0;  
 for (Edge edge : this.getEdges()) {  
  
  
 if (edge.getVertexOne() == vertex || edge.getVertexTwo() == vertex) {  
  
 cpt++;  
 }  
  
 }  
// System.out.println("%%cpt return value " + cpt);  
 return (cpt);  
  
 }  
  
  
 public int getHeaviestEdge(int vertex) {  
  
 int k = -1;  
  
 for (int i = 0; i < this.getEdges().size(); i++) {  
  
 if (this.getEdges().get(i).getVertexOne() == vertex || this.getEdges().get(i).getVertexTwo() == vertex) {  
  
  
 if (k == -1) {  
  
 k = i;  
 } else {  
  
 if (this.getEdges().get(i).getWeight() >= this.getEdges().get(k).getWeight()) {  
  
 k = i;  
  
 }  
 }  
  
 }  
  
 }  
 return k;  
  
 }  
  
  
 public int weight(){  
  
// System.out.println("entered weight method ");  
 int cpt=0;  
  
 for (Edge edge: this.getEdges()){  
// System.out.println("Entered weight's for loop ");  
  
 cpt+=edge.getWeight();  
  
 }  
// System.out.println("value of cpt " + cpt);  
 return cpt;  
  
  
 }  
  
  
 public void remove(int vertex){  
// System.out.println(" 15) entered remove method " + "and vertex equals " + vertex);  
  
 for (int i = 0; i < this.getEdges().size(); i++){  
  
// System.out.println(" 16) entered for loop ");  
  
 if((edges.get(i).getVertexOne() == vertex || edges.get(i).getVertexTwo() == vertex)){  
// System.out.println("entered if");  
  
 this.getEdges().remove(this.getEdges().get(i));  
// System.out.println("removed edge ");  
 i--;  
  
// System.out.println("i decremented");  
 }  
  
 }  
// System.out.println("trying to remove vertices ");  
  
 this.getVertices().remove(this.getVertices().indexOf(vertex));  
// System.out.println("vertices removed");  
  
 }  
  
  
  
  
}

import java.util.ArrayList;  
import java.util.Random;  
  
public class Main {  
  
  
  
  
  
 public static int pathGrowingAlgorithm(Graph g){  
  
 Graph m1 = new Graph();  
 Graph m2 = new Graph();  
  
 int i = 1;  
  
  
 while (g.getEdges().size()>0){  
  
// System.out.println(" 6) entered while loop g.getEdges().size() " + g.getEdges().size());  
  
 Integer random = g.getVertices().get(new Random().nextInt(g.getVertices().size()));  
 System.*out*.println("Random vertex " + random);  
  
 int kVertex = random;  
  
 while (g.getVertexWeight(kVertex)>0){  
  
  
 int secondVertex;  
  
 int f = g.getHeaviestEdge(kVertex);  
  
  
  
 if (g.getEdges().get(f).getVertexOne() == kVertex){  
  
 secondVertex = g.getEdges().get(f).getVertexTwo();  
 }  
  
 else {  
  
 secondVertex = g.getEdges().get(f).getVertexOne();  
 }  
  
 System.*out*.println("second vertex " + secondVertex);  
  
 if (i==1) {  
 m1.getEdges().add(g.getEdges().get(f));  
 }  
  
 else {  
  
 m2.getEdges().add(g.getEdges().get(f));  
 }  
  
 i = 3 - i;  
  
 g.remove(kVertex);  
  
 kVertex = secondVertex;  
  
 }  
  
  
 }  
  
  
 System.*out*.println(" M1 has total of weight of " + m1.weight() + " with number of edges " + m1.getEdges().size());  
 System.*out*.println(" M2 has total of weight of " + m2.weight() + " with number of edges " + m2.getEdges().size());  
  
  
 return Math.*max*(m1.weight(),m2.weight());  
  
  
 }  
  
  
 public static void main(String[] args) {  
 Graph graph;  
// System.out.println(" 1) graph of type Graph");  
 ArrayList<Edge> edges = new ArrayList<>();  
  
// System.out.println(" 2) edge arraylist created");  
  
  
  
  
  
 edges.add(new Edge(5,1,2));  
 edges.add(new Edge(2,2,3));  
 edges.add(new Edge(1,2,4));  
 edges.add(new Edge(6,3,4));  
  
 edges.add(new Edge(3,4,5));  
 edges.add(new Edge(4,5,1));  
  
  
  
  
  
  
// System.out.println(" 3) edges are added to arraylist of edge ");  
 graph=new Graph(edges,5);  
// System.out.println(" 5) added successfully ");  
  
 System.*out*.println("The maximum weighted matching using the Path growing algorithm is "+*pathGrowingAlgorithm*(graph));  
 }  
  
}