PROGRAMANDO EN JAVA

```
/*
 * EL SIGUIENTE EJEMPLO MUESTRA 4 ASPECTOS FUNDAMENTALES PARA EL ENVÍO CON
JAVA
 *    LECTURA RÁPIDA(NO CON EL SCANNER)
 *    ORDENAMIENTO
 *    REDONDEO
 *    Y ES UNA SOLUCIÓN EN EL FORMATO NECESARIO PARA EL COJ Y EL JURADO
XTREME *
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigDecimal;
import java.util.Arrays;


public class Main {

      static class nodo implements Comparable<nodo>{
            int fila, col;
            double costo;

            public nodo(int fila, int col, double costo) {
              this.fila = fila;
              this.col = col;
              this.costo = costo;
            }

            @Override
            public int compareTo(nodo o)
            {
               if( costo < o.costo ) return -1;
               if( costo > o.costo ) return 1;
               return 0;
            }

            @Override
            public String toString(){
              BigDecimal bdec = BigDecimal.valueOf(costo).setScale(3,
BigDecimal.ROUND_HALF_UP);
              return "["+fila+","+col+"] : "+bdec;
            }

          }
      public static void main(String[] args) throws IOException {
```

```
             BufferedReader bf = new BufferedReader( new
InputStreamReader(System.in) );

            int n = Integer.valueOf( bf.readLine() ), cont = 0;
            nodo[] conjunto_nodos = new nodo[n];
            while( n--> 0 ) {
              String[] part = bf.readLine().split(" ");
              conjunto_nodos[cont++] = new nodo( Integer.valueOf(part[0]),
Integer.valueOf(part[1]), Double.valueOf(part[2]) );
            }

            // aqui ordenamos el arreglo
            Arrays.sort(conjunto_nodos);

            for (int i = 0, size = conjunto_nodos.length; i < size; i++)
              System.out.println( conjunto_nodos[i] );

      }

}
////////

        /**
         * ****Como usar BigInteger en Java*****
         */
//de esta forma lo inicializamos
        BigInteger x = new BigInteger("123456789"); //donde el parametro
puede ser un String muy grande
        //        y seria el valor de inicializacion //tambien puede
inicializarse asi

        BigInteger a = BigInteger.ZERO;     //lo inicializa con valor 0
        BigInteger b = BigInteger.ONE;       //lo inicializa con valor 1
        BigInteger c = BigInteger.TEN;       //lo inicializa con valor 10

//estas son algunas de las operaciones que podemos realizar con este tipo de
variables
        a = a.add(b);           //suma
        a = a.divide(b);       //division
        a = a.mod(b);          //resto de division
        a = a.multiply(b);   //multiplicacion
        a = a.pow(3);          //potencia
        a = a.subtract(b);    //resta
//en todos estos casos el parametro b es otro BigInteger

////////////////////
/**
         * ****Como usar Tabla Hash en Java*****
         */
//esta estructura permite busquedas de elementos en O(1)
        /*
         * asi creamos la Tabla Hash, donde los parametros entre angulares
puede
         * ser cualquier tipo de datos primitivo digase Integer , String ,
etc
         * ... El parametro 1 sera la clave unica que los identifica y el
         * parametro 2 el valor asociado a esa clave
```

```
         */
        Hashtable<String, String> ht = new Hashtable<String, String>();

//asi colocamos un elemento en la tabla, donde los parametros tienen que ser del mismo tipo que definimos cuando la creamos, ejemplo: String, String, donde "a" seria la clave y "ejemplo" el valor
        ht.put("a", "ejemplo");

//asi obtenemos el valor asociado a la clave "a" o null si no existe en la tabla
        ht.get("a");
/////
//EN JAVA CUANDO LOS MÉTODOS NECESITAN COMPARAR OBJETOS Y ESTOS NO SON
//DE TIPOS PRIMITIVOS ES NECESARIO IMPLEMENTAR UN COMPARADOR
//EJEMPLO QUE ORDENA UNA LISTA DE PERSONAS DE ACUERDO A SU EDAD
    Collections.sort(personas, new Comparator<Persona>() {
            @Override
            public int compare(Persona o1, Persona o2) {
                  if (o1.edad < o2.edad) return -1;
                  if (o1.edad > o2.edad) return 1;
                  return 0;
            }
      });
///////
    Collections.binarySearch(collections, value);
    Collections.fill(collection, value);
    Collections.replaceAll(collection, oldValue, newValue);
    Collections.frequency(collection, value);
    Collections.max(collection);
    Collections.min(collection);
    Collections.reverse(collection);
    Collections.sort(collection);
```

//NÚMEROS PRIMOS

```
      boolean isPrime(int n) {
            if (n == 1)
                  return false;
            if (n == 2)
                  return true;
            if (n % 2 == 0)
                  return false;
            for (int i = 3; i * i <= n; i += 2)
                  if (n % i == 0)
                        return false;
            return true;
      }
///////
public static List<Integer> criba(int fin) {
            int i, j;
            List<Integer> primos = new ArrayList<Integer>();
            boolean cribaArr[] = new boolean[fin];
            Arrays.fill(cribaArr, true);
```

```
		for (i = 0; i < fin; i += 2) {
			cribaArr[i] = false;
		}
		cribaArr[1] = false;
		cribaArr[2] = true;
		primos.add(2);
		int maxDiv = (int) Math.sqrt(fin);
		for (i = 3; i <= maxDiv; i += 2) {
			if (cribaArr[i]) {
				primos.add(i);
				for (j = i + i; j < fin; j += i) {
					cribaArr[j] = false;
				}
			}
		}
		for (; i < fin; i++) {
			if (cribaArr[i]) {
				primos.add(i);
			}
		}
		return primos;
	}
/////
	public static List<Integer> criba(int inicio, int fin) {
		int i, j, cantN;
		cantN = fin - inicio + 1;
		List<Integer> primos = new ArrayList<Integer>();
		boolean cribaArr[] = new boolean[cantN];
		Arrays.fill(cribaArr, true);
		for (i = (inicio % 2); i < cantN; i += 2) {
			cribaArr[i] = false;
		}
		int maxDiv = (int) Math.sqrt(fin);
		for (i = 3; i <= maxDiv; i += 2) {
			if (i > inicio && !cribaArr[i - inicio]) {
				continue;
			}
			j = inicio / i * i;
			if (j < inicio) {
				j += i;
			}
			if (j == i) {
				j += i;
			}
			j -= inicio;
			for (; j < cantN; j += i) {
				cribaArr[j] = false;
			}
		}
		if (inicio <= 1) {
			cribaArr[1 - inicio] = false;
		}
		if (inicio <= 2) {
			cribaArr[2 - inicio] = true;
		}
		for (i = 0; i < cantN; i++) {
			if (cribaArr[i]) {
```

```
                        primos.add(inicio + i);
                    }
                }
                return primos;
        }
```

GEOMATRIA

```
        public static double logaritmo(double valor, double base) {
                return Math.log(valor) / Math.log(base);
        }

        public static int GCD(int a, int b) {// MÁXIMO COMUN DIVISOR
                BigInteger _a = BigInteger.valueOf(a);
                BigInteger _b = BigInteger.valueOf(b);
                return (_a.gcd(_b)).intValue();
        }

        public static int GCD(int a, int b) {//OTRO MÁXIMO COMUN DIVISOR
                while (b > 0) {
                        a = a % b;
                        a ^= b;
                        b ^= a;
                        a ^= b;
                }
                return a;
        }

        public static int LCM(int a, int b) {// MÍNIMO COMUN MÚLTIPLO
                return (a * b) / GCD(a, b);
        }


        public static double raizNsimaDeX(double x, double n){//RAIZ NSIMA DE X
                return Math.pow(x, 1.0 / n);
        }
```

COMBINATORIA

TEORÍA DE NÚMEROS

GRAFOS

```
public class Main {

    public static void main(String[] args) {
        //leer el grafo y utilizar el Dijkstra
    }
    /*
```

```
     * RECIBE UN NODO DE INICIO Y UNA MATRIZ DE ADYACENCIA CON -1 DENOTANDO
QUE
     * NO HAY ADYACENCIA ENTRE LOS NODOS Y RETORNA UN ARREGLO CON LA DISTANCI
A
     * CADA NODO.
     */

    public static int[] Dijkstra(int inicio, int adyMatriz[][]) {
        int cantNodos = adyMatriz.length;
        int distancia[] = new int[cantNodos];
        boolean visitado[] = new boolean[cantNodos];
        Arrays.fill(visitado, false);
        Arrays.fill(distancia, Integer.MAX_VALUE / 2);

        PriorityQueue<Nodo> cola = new PriorityQueue<Nodo>();
        cola.add(new Nodo(inicio, 0));
        distancia[inicio] = 0;
        int cantVis = 0;
        while (cantVis < cantNodos && !cola.isEmpty()) {
            int u = cola.poll().index;
            if (visitado[u]) {
                continue;
            }

            visitado[u] = true;
            cantVis++;
            for (int i = 0; i < cantNodos; i++) {
                if (adyMatriz[u][i] == -1);//SI NO ES ADYACENTE NO TENERLO EN
CUENTA
                int nuevaDist = distancia[u] + adyMatriz[u][i];
                if (distancia[i] > nuevaDist) {
                    distancia[i] = nuevaDist;
                    cola.add(new Nodo(i, distancia[i]));
                }
            }
        }
        return distancia;
    }
}
class Nodo implements Comparable<Nodo> {

    int index;
    Integer distance;

    public Nodo(int index, int distance) {
        this.index = index;
        this.distance = distance;
    }

    @Override
    public int compareTo(Nodo o) {
        return this.distance.compareTo(o.distance);
    }
}
```

//////////////////

```java
public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
    }
    /*
     * RECIBE UNA LISTA DE ARISTAS Y LA CANTIDAD DE NODOS
     * Y DEVUELVE EL COSTO DE HACER UN ARBOL DE EXPANCIÓN MÍNIMO O MST
     * ADEMÁS ACUALIZA EL ATRIBUTO USADA DE LAS ARISTAS DICIENDO SIN ESTÁN O
NO
     * PRESENTES EN EL ARBOL FINAL
     */

    public static long Kruskal(List<Arista> aristas, int cantNodos) {
        DisjoinSet ds = new DisjoinSet(cantNodos);
        long tcost = 0;
        Collections.sort(aristas);
        for (Arista arista : aristas) {
            if (ds.FIND(arista.inicio) != ds.FIND(arista.fin)) {
                ds.UNION(arista.inicio, arista.fin);
                tcost += arista.peso;
                arista.usada = true;
            }
            arista.usada = false;
        }
        return tcost;
    }
}

class DisjoinSet {
    int ds[];
    int cantElementos;
    public DisjoinSet(int cantElementos) {
        this.cantElementos = cantElementos;
        this.ds = new int[cantElementos];
        for (int i = 1; i <= cantElementos; ++i) {
            ds[i] = i;
        }
    }

    public int FIND(int x) {
        if (ds[x] == x) {
            return x;
        }
        return ds[x] = FIND(ds[x]);
    }

    public void UNION(int x, int y) {
        ds[FIND(x)] = FIND(y);
    }
}

class Arista implements Comparable<Arista> {
    int inicio;
    int fin;
    Integer peso;
```

```
    boolean usada;

    public Arista(int inicio, int fin, int peso) {
        this.inicio = inicio;
        this.fin = fin;
        this.peso = peso;
    }

    @Override
    public int compareTo(Arista o) {
        return this.peso.compareTo(o.peso);
    }
}
```