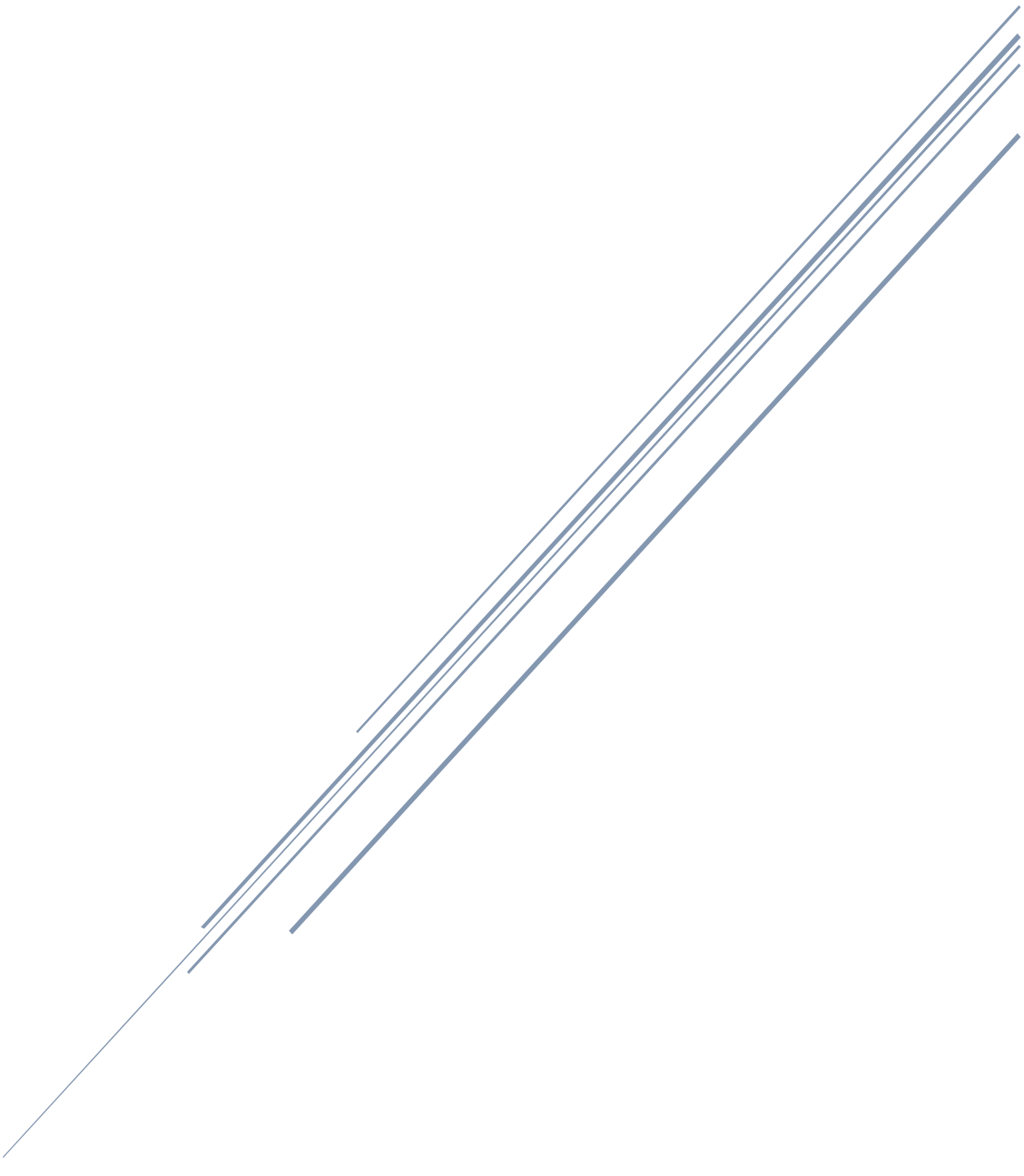
CHULETA DE ALGORITMOS

PROGRAMACION COMPETITIVA

Contenido



1. Documentación Java

BIGDECIMAL Y BIGINTEGER

Constructores con Strings

Funciones:

A = A.abs();

A = A.add(B); divide, multiply, min, max,negate,pow,...

equals, compare to,...

BIG-DECIMAL

stripTrailingZeros -> Quita los ceros del final

remainder -> como modulo

.setScale(int nuevaEscala, [int modoRedondeo, RoundingMode modo]) -> Para redondear a tu gusto

- ➔ RoundingMode.CEILING, FLOOR, UP, DOWN, HALF_UP,HALF_DOWN,HALF_EVEN (Diferencias UP/CEILING y DOWN/FLOOR tienen que ver con la referencia de redondeo [0 o el infinito] (para numeros positivos es igual cual usar))

BIG-INTEGER

gcd, isProbablePrime, mod

DECIMAL FORMAT

```
DecimalFormat df = (DecimalFormat) DecimalFormat.getNumberInstance(Locale.ENGLISH);
df.applyPattern("0.0000"); # para que no salgan 0s
-> pw.print(df.format(DOUBLE));
```

---

2. Mates

ALGUNAS FORMULAS MATEMÁTICAS

$$\sum_{1}^{n} i = \frac{n(n+1)}{2} \quad (Num\ triangulares) \qquad \sum_{1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad a + \cdots + b = \frac{n(a+b)}{2}$$

$$Pirámide = \frac{n(n+1)(n+2)}{6} \qquad \sum_{1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 \qquad \binom{a}{b} = \frac{a!}{b!\,(a-b)!}$$

$$Cn = \frac{(2n)!}{(n+1)!\,n!} \quad (Nº\ de\ paréntesis\ balanceados\ o\ árboles\ binarios\ de\ n\ nodos(Tb\ rooted))$$

//MCM

```
static int mcm (int a, int b) {
	return a * b / mcd(a, b);
}
```

//Numeros de catalan

```
int catalan(int n) {
	int res = 0;
```

```
        if (n <= 1) { return 1;  }
        for (int i = 0; i < n; i++) {
            res += catalan(i) * catalan(n - i - 1);
        }
        return res;
    }
```

3. Grafos

//DFS

```
public static void DFS(ArrayList<Arista>[] grafo,int N){
       ArrayDeque<State> pila = new ArrayDeque<>();
       boolean visitado[] = new boolean[N];
       pila.push(new State(0,0));
       visitado[0] = true;
       while(!pila.isEmpty()){
                    State aux = pila.pop();
                    for(Arista ady: grafo[aux.nodo]){
                    int destino = ady.to;
                           if(!visitado[destino]){
                                        pila.push(new State(destino,0));
                                        visitado[destino] = true;
                    }
             }//for
       }//while
}//DFS
```

//BFS

```
public static void BFS(ArrayList<Arista>[] grafo,int N){
       ArrayDeque<State> cola = new ArrayDeque<>();
       boolean visitado[] = new boolean[N];
       cola.offer(new State(0,'0'));
       visitado[0] = true;
       while(!cola.isEmpty()){
             State aux = cola.poll();
             System.out.println(aux.nodo);
             for(Arista ady: grafo[aux.nodo]){
                    int destino = ady.to;
                    if(!visitado[destino]){
```

```
                        cola.offer(new State(destino,'0'));
                        visitado[destino] = true;
                    }
                }
            }
}
```

//Ciclo (y Camino) Euleriano: solo si C(v) es par (C(v) par con impar en 2 vertices)

//Ciclo (u Camino) Hamiltoniano:

//Componentes Conexas: modificar dfs, pasando por referencia visitado[] y el nodo inicio

```
public static int cc(ArrayList<Arista>[] grafo,int N){
        boolean visitado[] = new boolean[N];
        int cuenta = 0;
        for (int i = 0; i < N; i++) {
            if(!visitado[i]){
                DFS(grafo,N,i,visitado);
                cuenta++;
            }
        }
        return cuenta;
}
```

//TopoSort

```
public static void topoUtil(int[][] grafo,int actual,boolean[] visitado,ArrayDeque<Integer>
pila){
        visitado[actual] = true;
        for (int i = 0; i < grafo.length; i++) {
            if(grafo[actual][i]>0 && !visitado[i]){
                topoUtil(grafo,i,visitado,pila);
            }
        }
        pila.push(actual);
}
public static ArrayDeque<Integer> topoSort(int[][] grafo){
        ArrayDeque<Integer> pila = new ArrayDeque<>();
        ArrayDeque<Integer> output = new ArrayDeque<>();
        boolean visitado[] = new boolean[grafo.length];
        for (int i = 0; i < grafo.length; i++) {
```

```
            if(!visitado[i]){
                topoUtil(grafo,i,visitado,pila);
            }
            while(!pila.isEmpty()){
                output.offer(pila.pop());
            }
        }
        return output;
}
```

//SCC

```
public static int scc(int[][] grafo,int N){
       boolean visitado[] = new boolean[N];
       ArrayDeque<Integer> cola = topoSort(grafo);
       int cuenta = 0;
       for (int i: cola) {
             if(!visitado[i]){
                   DFSt(grafo,N,i,visitado);
                   cuenta++;
}     } return cuenta;  }//Fin scc
```

//DFSTranspuesto

```
public static void DFSt(int[][] grafo,int N,int inicio,boolean visitado[]){
       ArrayDeque<Integer> pila = new ArrayDeque<>();
       pila.push(inicio);
       visitado[inicio] = true;
       while(!pila.isEmpty()){
             int aux = pila.pop();
             for(int ady=0;ady<N;ady++){
                   if(grafo[ady][aux]>0){
                         int destino = ady;
                         if(!visitado[destino]){
                               pila.push(destino);
                               visitado[destino] = true;
                         }}}}}
class UnionF{
```

/*ESTRUCTURA UNION FIND*/

```
    int N;
```

```
int parent[];
int size[];
public UnionF(int N){
    this.N = N;
    this.parent = new int[N];
    this.size = new int[N];
}

public void init(){
    for (int i = 0; i < N; i++) {
        parent[i] = i;
        size[i] =1;
    }
}
public int find(int nodo){
    while(nodo!=parent[nodo]){
        nodo = parent[nodo];
    }
    return nodo;
}
public int getSize(int nodo){
    return this.size[nodo];
}
public void join(int A,int B){
    A = find(A);
    B = find(B);
    if(size[A]<size[B]){
        parent[A] = B;
        size[B]+=size[A];
    }else{
        parent[B] = A;
        size[A]+=size[B];
    }
}
public boolean isConnected(int A, int B){
    return (find(A) == find(B));
```

```
    }
    public int connect(int A, int B){//0 ya estaban conectados, 1 conectado correctamente
        if(isConnected(A,B)){
            return 0;
        }else{
            join(A,B);
            return 1;
        }
    }
}
class Arista implements Comparable<Arista>{
    int from,to,peso;

    public Arista(int from, int to, int peso) {
        this.from = from;
        this.to = to;
        this.peso = peso;
    }
    @Override
    public int compareTo(Arista o) {
        return this.peso-o.peso;
    }
    @Override
    public String toString(){
        return "[" + this.from + "->" + this.to +"||"+ this.peso + "]";
    }
}
public class UF {  /*Clase principal*/
    public static void main(String args[]){
        Scanner scn = new Scanner(System.in);
        int N = scn.nextInt();
        int M = scn.nextInt();
        ArrayList<Arista>[] grafo = new ArrayList[N];
        PriorityQueue<Arista> grafoK = new PriorityQueue<>();
        for (int i = 0; i < N; i++) {
            grafo[i] = new ArrayList<>();
```

```
        }
        for (int i = 0;i<M;i++){
            int from = scn.nextInt();
            int to = scn.nextInt();
            int peso = scn.nextInt();

            grafo[from].add(new Arista(from,to,peso));
            grafo[to].add(new Arista(to,from,peso));

            grafoK.offer(new Arista(from,to,peso));
        }
        System.out.println(KuruskalRM(grafoK,N));
    }
```

/*ALGORITMO DE KURUSKAL*/

```
    public static int KuruskalCC(PriorityQueue<Arista> grafo,int N){//Cuenta componentes
Conexas
        UnionF uf = new UnionF(N);
        uf.init();
        while (N>1 && !grafo.isEmpty()){
            Arista aux = grafo.poll();
            N -= uf.connect(aux.from,aux.to);
        }
        return N;
    }
    public static int KuruskalRM(PriorityQueue<Arista> grafo,int N){//Version camino minimo
        UnionF uf = new UnionF(N);
        uf.init();
        int pesos = 0;
        while (N>1 && !grafo.isEmpty()){

            Arista aux = grafo.poll();
            if(uf.connect(aux.from,aux.to) == 1){
                pesos+=aux.peso;
                N--;
            }
        }
```

```
        return pesos;
    }
}
```

//DIJSKTRA (matriz)

```
PriorityQueue<State> cola = new PriorityQueue<>();
boolean visitado[] = new boolean[N];
int distancia[] = new int[N];
int INF = -1;
for (int i = 1; i < N; i++) {
      distancia[i] = INF;
}
cola.offer(new State(0,0));
while(!cola.isEmpty()){
State actual = cola.poll();  //a[0] i del grafo a[1] valor del grafo
      for(int j = 0;j<N;j++){
            if(actual.nodo!= j && grafo[actual.nodo][j]!=INF && !visitado[actual.nodo]){
if(distancia[j]>grafo[actual.nodo][j] + actual.distancia || distancia[j] == INF){
                        distancia[j] = grafo[actual.nodo][j] + actual.distancia;
}
cola.offer(new State(j,distancia[j]));
            }
      }
      visitado[actual.nodo]=true;
}//Fin del while
```

//DIJSKTRA (listas)

```
PriorityQueue<State> cola = new PriorityQueue<>();
boolean visitado[] = new boolean[N];
int distancia[] = new int[N];
int INF = -1;
for (int i = 1; i < N; i++) {
      distancia[i] = INF;
}
cola.offer(new State(0,0));
while(!cola.isEmpty()){
      State actual = cola.poll();  //a[0] i del grafo a[1] valor del grafo
      for(Arista arista: grafo[actual.nodo]){
```

```
            int destino = arista.to;
            int peso = arista.peso;
            if(!visitado[actual.nodo]){
if(distancia[destino]>peso + actual.distancia || distancia[destino] == INF){
                        distancia[destino] = peso + actual.distancia;
}
      cola.offer(new State(destino,distancia[destino]));
            }
      }
      visitado[actual.nodo]=true;
}
```

//PRIM

```
//Previo
class State{  int nodo, distancia; }
class Arista{ int from,to,peso; }
ArrayList<Arista>[] grafo = new ArrayList[N];
grafo[ini].add(new Arista(ini,fin,coste));
grafo[fin].add(new Arista(fin,ini,coste));
//Algoritmo
PriorityQueue<State> cola = new PriorityQueue<>();
            boolean visitado[] = new boolean[N];
            int suma = 0;
            cola.offer(new State(0,0));
            while(!cola.isEmpty()){
                  State aux = cola.poll();
                  if(!visitado[aux.nodo]){
                              suma+=aux.distancia;
                              visitado[aux.nodo] = true;
                              for (int i = 0; i < grafo[aux.nodo].size(); i++) {
                                    int destino = grafo[aux.nodo].get(i).to;
                                    int peso = grafo[aux.nodo].get(i).peso;
                                    if(!visitado[destino]){
                                          cola.offer(new State(destino,peso));
            }           }     }     }
            int f = 0;
            for (int i = 0; i < N; i++) {
```

```
                if(!visitado[i]){
                        f = 1;
                        break;
                }
            }
```

//FLOYD-WARSHALL

```
for(int k = 0; k<N;k++){
      for(int i = 0; i<N;i++){
            for(int j = 0; j<N;j++){
                  grafo[i][j] = Math.min(grafo[i][j], grafo[i][k] + grafo[k][j]);
            }
      }
}
```

4. Programacion Dinámica

EDIT DISTANCE:

```
function F(i, j):
      if i >= S.length && j >= T.length:
            return 0
      if i >= S.length or j >= T.length:
            return INF
      if memo[i][j] != -1
            return memo[i][j]
      if S[i] == T[j]:
            memo[i][j] = F(i+1, j+1)
      else:
            memo[i][j] = min(F(i+1, j)+1, F(i, j+1)+1, F(i+1,j+1)+1)
      return memo[i][j]
```

LONGEST COMMON SUBSEQUENCE

```
function LCS(i, j):
      if i >= S.length or j >= T.length:
            return 0
      if memo[i][j] != -1
            return memo[i][j]
      if S[i] == T[j]:
            memo[i][j] = F(i+1, j+1) + 1
```

```
	else:
		memo[i][j] = max(F(i+1, j), F(i, j+1))
	return memo[i][j]
```

LONGEST INCREASING SUBSEQUENCE

```
function F(i, j):
	if i >= A.length:
		return 0
	if memo[i][j] != -1
		return memo[i][j]
	if A[i] > A[j]:
		memo[i][j] = F(i+1, i) + 1
	else:
		memo[i][j] = max(F(i+1, i), F(i+1, j))
	return memo[i][j]
```

PROBLEMA DE LA MOCHILA CON CHOICE

```
function f(i, weight):
	if weight > W: return -INF
	if weight == W or i >= N: return 0
	if memo[i,weight] != -1: return memo[i,weight]
	T1 = f(i+1, weight + W[i]) + V[i]
	T2 = f(i+1, weight)
	if T1 > T2:
		memo[i, weight] = T1
		choice[i, weight] = (i+1, weight + W[i])
	else:
		memo[i, weight] = T2
		choice[i, weight] = (i+1, weight)
	return memo[i,weight]
```