

# Team notebook

15 de septiembre de 2016

## Índice

<b>1. Bonus: Input Output</b>	<b>1</b>
1.1. Scanner . . . . .	1
1.2. printWriter . . . . .	2
<b>2. Data Structures</b>	<b>2</b>
2.1. Disjoint Set . . . . .	2
2.2. RMQ . . . . .	3
<b>3. Dynamic Programming</b>	<b>3</b>
3.1. Knapsack . . . . .	3
3.2. Longest increasing subsequence . . . . .	4
3.3. Max Range Sum . . . . .	4
<b>4. Geometry</b>	<b>4</b>
4.1. Angle . . . . .	4
4.2. Area . . . . .	5
4.3. Collinear Points . . . . .	5
4.4. Euclidean Distance . . . . .	5
4.5. Gometric Vector . . . . .	5
4.6. Perimeter . . . . .	5
4.7. Point in Polygon . . . . .	6
4.8. Point . . . . .	6
4.9. Sexagesimal degrees and radians . . . . .	6
<b>5. Graphs</b>	<b>6</b>
5.1. BFS . . . . .	6
5.2. DFS . . . . .	7
5.3. Dijkstra's Algorithm . . . . .	7
5.4. FloodFill . . . . .	8
5.5. Floyd-Warshall's Algorithm . . . . .	9

5.6. Kruskal's Algorithm . . . . .	9
5.7. Trajan's Algorithm . . . . .	10
<b>6. Math</b>	<b>11</b>
6.1. Binary Exponentiation . . . . .	11
6.2. Binomial Coefficient . . . . .	11
6.3. Catalan Number . . . . .	11
6.4. Euler Totient . . . . .	12
6.5. Gaussian Elimination . . . . .	12
6.6. Greatest common divisor . . . . .	13
6.7. Lowest Common multiple . . . . .	13
6.8. Miller-Rabin . . . . .	13
6.9. Prime Factorization . . . . .	13
6.10. Sieve of Eratosthenes . . . . .	14
<b>7. String</b>	<b>14</b>
7.1. KMP's Algorithm . . . . .	14
<b>8. Tips and formulas</b>	<b>15</b>
8.1. ASCII Table . . . . .	15
8.2. Catalan Number . . . . .	16
8.3. Euclidean Distance . . . . .	16
8.4. Permutation and combination . . . . .	16
8.5. Time Complexities . . . . .	17
8.6. mod: properties . . . . .	17

## 1. Bonus: Input Output

### 1.1. Scanner

---

Libreria para recibir las entradas; reemplaza el Scanner original, mejorando su eficiencia.

Contiene los metodos next, nextLine y hasNext. Para recibir datos numericos, hacer casting de next.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class Main {

    static class Scanner{
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        StringTokenizer st = new StringTokenizer("");
        int espacios = 0;

        public String nextLine() throws IOException{
            if(espacios>0){
                espacios--;
                return "";
            }else if(st.hasMoreTokens()){
                StringBuilder salida = new StringBuilder();
                while(st.hasMoreTokens()){
                    salida.append(st.nextToken());
                    if(st.countTokens()>0){
                        salida.append(" ");
                    }
                }
                return salida.toString();
            }
            return br.readLine();
        }

        public String next() throws IOException{
            espacios=0;
            while (!st.hasMoreTokens() ) {
                st = new StringTokenizer(br.readLine() );
            }
            return st.nextToken();
        }
    }
}
```

```
public boolean hasNext() throws IOException{
    while (!st.hasMoreTokens()) {
        String linea = br.readLine();
        if (linea == null) {
            return false;
        }
        if(linea.equals("")){
            espacios++;
        }
        st = new StringTokenizer(linea);
    }
    return true;
}
}
```

---

## 1.2. printWriter

---

Utilizar en lugar del System.out.println para mejorar la eficiencia.

```
import java.io.PrintWriter;

PrintWriter so = new PrintWriter(new BufferedWriter(new
    OutputStreamWriter(System.out)));
so.print("Imprime sin salto de linea");
so.println("Imprime con salto de linea");

//Al finalizar
so.close();
```

---

## 2. Data Structures

### 2.1. Disjoint Set

---

```
import java.util.*;

class DisjointSet{
    private int [] parent, rank, size;
    private int cantSets;
```

```

//Inicializa todas las estructuras :)
public DisjointSet(int n){
    parent=new int[n];
    rank=new int[n];
    size=new int[n];
    cantSets=n;

    int i;
    for(i=0; i<n; i++){
        parent[i]=i;
        rank[i]=0; //Se usa para tener un control sobre el
                    "tamao" del arbol
        size[i]=1;
    }
}

//Operaciones
public int find(int i){
    if(parent[i]==i)    return i;
    else{
        int x=find(parent[i]);
        //Compresin de ruta
        parent[i]=x;
        return x;
    }
}

public void union(int i, int j){
    int x=find(i);
    int y=find(j);

    if(x!=y){
        cantSets--;

        //El set ms pequeno (rank) se aade al set ms
        grande
        if(rank[x]>rank[y]){
            parent[y]=x;
            size[x]=size[x]+size[y];
        }else{
            parent[x]=y;
            size[y]=size[y]+size[x];

            if(rank[x]==rank[y]){
                rank[y]=rank[y]+1;
            }
        }
    }
}

```

```

    }
}

    }

    }

    public int numDisjointSets(){
        return cantSets;
    }

    public int sizeOfSet(int i){
        return size[find(i)];
    }
}

```

## 2.2. RMQ

Range minimum query. Recibe como parametro en el constructor un array de valores. Las consultas se realizan con el mtodo rmq(indice\_inicio, indice\_final) y pueden actualizarse los valores con update\_point(indice, nuevo\_valor)

```

import java.util.*;

class SegmentTree {
    private int[] st, A;
    private int n;
    private int left (int p) { return p << 1; }
    private int right(int p) { return (p << 1) + 1; }

    private void build(int p, int L, int R) {
        if (L == R)
            st[p] = L;
        else {
            build(left(p) , L, (L + R) / 2);
            build(right(p), (L + R) / 2 + 1, R );
            int p1 = st[left(p)], p2 = st[right(p)];
            st[p] = (A[p1] <= A[p2]) ? p1 : p2;
        }
    }

    private int rmq(int p, int L, int R, int i, int j) {

```

```

if (i > R || j < L) return -1;
if (L >= i && R <= j) return st[p];
int p1 = rmq(left(p), L, (L+R) / 2, i, j);
int p2 = rmq(right(p), (L+R) / 2 + 1, R, i, j);

if (p1 == -1) return p2;
if (p2 == -1) return p1;
return (A[p1] <= A[p2]) ? p1 : p2; }

private int update_point(int p, int L, int R, int idx, int new_value) {
    int i = idx, j = idx;
    if (i > R || j < L)
        return st[p];
    if (L == i && R == j) {
        A[i] = new_value;
        return st[p] = L;
    }
    int p1, p2;
    p1 = update_point(left(p), L, (L + R) / 2, idx, new_value);
    p2 = update_point(right(p), (L + R) / 2 + 1, R, idx, new_value);

    return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
}

public SegmentTree(int[] _A) {
    A = _A; n = A.length;
    st = new int[4 * n];
    for (int i = 0; i < 4 * n; i++) st[i] = 0;
    build(1, 0, n - 1);
}

public int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); }
public int update_point(int idx, int new_value) {
    return update_point(1, 0, n - 1, idx, new_value); }
}

class Main {
    public static void main(String[] args) {
        int[] A = new int[] { 18, 17, 13, 19, 15, 11, 20 };
        SegmentTree st = new SegmentTree(A);
    }
}

```

## 3. Dynamic Programming

### 3.1. Knapsack

Dados N articulos, cada uno con su propio valor y peso y un tamaño máximo de una mochila, se debe calcular el valor máximo de los elementos que es posible llevar.

Debe seleccionarse un subconjunto de objetos, de tal manera que quepan en la mochila y representen el mayor valor posible.

```

static int MAX_WEIGHT = 40; //Peso máximo de la mochila
static int MAX_N = 1000; //Número máximo de objetos
static int N; //Número de objetos
static int prices[] = new int[MAX_N]; //precios de cada producto
static int weights[] = new int[MAX_N]; //pesos de cada producto
static int memo[][] = new int[MAX_N][MAX_WEIGHT]; //tabla dp

//El metodo debe llamarse con 0 en el id, y la capacidad de la mochila en
w
static int knapsack(int id, int w) {
    if (id == N || w == 0) {
        return 0;
    }
    if (memo[id][w] != -1) {
        return memo[id][w];
    }
    if (weights[id] > w) {
        memo[id][w] = knapsack(id + 1, w);
    } else {
        memo[id][w] = Math.max(knapsack(id + 1, w), prices[id] +
            knapsack(id + 1, w - weights[id]));
    }
    return memo[id][w];
}

//Antes de llamar al metodo, todos los campos de la tabla memo deben
iniciarse a -1

```

### 3.2. Longest increasing subsequence

Halla la longitud de la subsecuencia creciente mas larga. MAX debe definirse en el tamao limite del array, n es el tamao del array. Puede aplicarse tambien sobre strings, cambiando el parametro `int s[]` por `string s`. Si debe ser estrictamente creciente, cambiar el `<=` de `s[j] <= s[i]` por `<`

```
static int MAX = 1005;
static int memo[] = new int[MAX];

static int longestIncreasingSubsequence(int s[]){
    int n = s.length;
    memo[0] = 1;
    int output = 0;
    for (int i = 1; i < n; i++){
        memo[i] = 1;
        for (int j = 0; j < i; j++){
            if (s[j] <= s[i] && memo[i] < memo[j] + 1){
                memo[i] = memo[j] + 1;
            }
        }
        if(memo[i] > output){
            output = memo[i];
        }
    }
    return output;
}
```

### 3.3. Max Range Sum

Dado un arreglo de enteros, retorna la mxima suma de un rango de la lista.

```
static int maxRangeSum(int[] a){
    int sum = 0, ans = 0;
    for (int i = 0; i < a.length; i++){
        if (sum + a[i] >= 0) {
            sum += a[i];
            ans = Math.max(ans, sum);
        }else{
            sum = 0;
        }
    }
    return ans;
}
```

## 4. Geometry

### 4.1. Angle

Dados 3 puntos A, B, y C, determina el valor del angulo ABC (origen en B) en radianes. IMPORTANTE: Definir la clase Point y Vec (Geometric Vector). Si se desea convertir a grados sexagesimales, revisar Sexagesimal degrees and radians.

```
static double angle(Point a, Point b, Point c) {
    Vec ba = toVector(b, a);
    Vec bc = toVector(b, c);
    return Math.acos((ba.x * bc.x + ba.y * bc.y) / Math.sqrt((ba.x *
        ba.x + ba.y * ba.y) * (bc.x * bc.x + bc.y * bc.y)));
}
```

### 4.2. Area

Calcula el area de un polgono representado como un ArrayList de puntos. IMPORTANTE: Definir `P[0] = P[n-1]` para cerrar el polgono. El algortmo utiliza el metodo de determinante de la matriz de puntos de la figura. IMPORTANTE: Debe definirse previamente la clase Point.

```
public static double area(ArrayList<Point> P) {
    double result = 0.0, x1, y1, x2, y2;
    for (int i = 0; i < P.size()-1; i++) {
        x1 = P.get(i).x;
        x2 = P.get(i+1).x;
        y1 = P.get(i).y;
        y2 = P.get(i+1).y;
        result += ((x1 * y2) - (x2 * y1));
    }
    return Math.abs(result) / 2.0;
}
```

### 4.3. Collinear Points

Determina si el punto r est en la misma linea que los puntos p y q. IMPORTANTE: Deben incluirse las estructuras point y vec.

```
static double cross(Vec a, Vec b) {
    return a.x * b.y - a.y * b.x;
}
static boolean collinear(Point p, Point q, Point r) {
    return Math.abs(cross(toVector(p, q), toVector(p, r))) < 1e-9;
}
```

## 4.4. Euclidean Distance

Halla la distancia euclideana de 2 puntos en dos dimensiones (x,y). Para usar el primer mtodo, debe definirse previamente la clase Point

```
/*Trabajando con la clase Point*/
static double euclideanDistance(Point p1, Point p2) {
    return Math.hypot(p1.x - p2.x, p1.y - p2.y);
}
/*Trabajando con los valores x y y de cada punto*/
static double euclideanDistance(double x1, double y1, double x2, double y2){
    return Math.hypot(x2 - x1, y2 - y1);
}
```

## 4.5. Gometric Vector

Dados dos puntos A y B, crea el vector A->B. IMPORTANTE: Debe definirse la clase Point. Es llamado Vec para no confundirlo con vector como coleccin de elementos.

```
static class Vec {
    public double x, y;
    public Vec(double _x, double _y) {
        this.x = _x;
        this.y = _y;
    }
}
static Vec toVector(Point a, Point b) {
    return new Vec(b.x - a.x, b.y - a.y);
}
```

## 4.6. Perimeter

Calcula el permetro de un polgono representado como un vector de puntos. IMPORTANTE: Definir  $P[0] = P[n-1]$  para cerrar el polgono. La estructura point debe estar definida, al igual que el mtodo euclideanDistance.

```
public static double perimeter(ArrayList<Point> P) {
    double result = 0.0;
    for (int i = 0; i < P.size()-1; i++){
        result += euclideanDistance(P.get(i), P.get(i+1));
    }
    return result;
}
```

## 4.7. Point in Polygon

Determina si un punto pt se encuentra en el polgono P. Este polgono se define como un vector de puntos, donde el punto 0 y n-1 son el mismo. IMPORTANTE: Deben incluirse las estructuras point y vec, ademas del mtodo angle y el mtodo cross que se encuentra en Collinear Points.

```
static boolean ccw(Point p, Point q, Point r) {
    return cross(toVector(p, q), toVector(p, r)) > 0;
}
static boolean inPolygon(Point pt, ArrayList<Point> P) {
    if (P.size() == 0){
        return false;
    }
    double sum = 0;
    for (int i = 0; i < P.size()-1; i++) {
        if (ccw(pt, P.get(i), P.get(i+1))){
            sum += angle(P.get(i), pt, P.get(i+1));
        }else{
            sum -= angle(P.get(i), pt, P.get(i+1));
        }
    }
    if(Math.abs(Math.abs(sum) - 2*Math.acos(-1.0)) < 1e-9){
        return true;
    }
    return false;
}
```

```
}
```

## 4.8. Point

La clase punto ser la base sobre la cual se ejecuten otros algoritmos.

```
static class Point {
    public double x, y;
    public Point() {
        this.x = this.y = 0.0;
    }
    public Point(double _x, double _y){
        this.x = _x;
        this.y = _y;
    }
    public boolean equals(Point other){
        if(Math.abs(this.x - other.x) < 1e-9 && (Math.abs(this.y -
            other.y) < 1e-9)) {
            return true;
        }
        return false;
    }
}
```

## 4.9. Sexagesimal degrees and radians

Conversiones de grados sexagesimales a radianes y viceversa.

```
static double DegToRad(double d) {
    return d * Math.PI / 180.0;
}

static double RadToDeg(double r) {
    return r * 180.0 / Math.PI;
}
```

# 5. Graphs

## 5.1. BFS

Algoritmo de bsqueda en anchura en grafos, recibe un nodo inicial s y visita todos los nodos alcanzables desde s. BFS tambien halla la distancia ms corta entre el nodo inicial s y los dems nodos si todas las aristas tienen peso 1.

```
import java.util.ArrayList;
import java.util.Queue;
import java.util.LinkedList;

static int v, e; //vertices, arcos
static int MAX=100005;
static ArrayList<Integer> ady[] = new ArrayList[MAX]; //lista de
    Adyacencia
static long distance[] = new long[MAX];

//Mtodo para limpiar los valores de las estructuras.
static void init() {
    for (int j = 0; j <= v; j++) {
        distance[j] = -1;
        ady[j] = new ArrayList<Integer>();
    }
}

//Recibe el nodo inicial s
static void bfs(int s){
    Queue<Integer> q=new LinkedList<Integer>();
    q.add(s); //Inserto el nodo inicial
    distance[s]=0;
    int actual, i, next;

    while(!q.isEmpty()){
        actual=q.poll();
        for(i=0; i<ady[actual].size(); i++){
            next=ady[actual].get(i);
            if(distance[next]==-1){
                distance[next]=distance[actual]+1;
                q.add(next);
            }
        }
    }
}
```

```

    }
}

```

---

## 5.2. DFS

Algoritmo de bsqueda en profundidad para grafos. Parte de un nodo inicial s visita a todos sus vecinos. DFS puede ser usado para contar la cantidad de componentes conexas en un grafo y puede ser modificado para que retorne informacin de los nodos dependiendo del problema. Permite hallar ciclos en un grafo.

```

import java.util.ArrayList;

static int v, e; //vertices, arcos
static int MAX=100005;
static ArrayList<Integer> ady[] = new ArrayList[MAX];
static boolean marked[] = new boolean[MAX];

//Limpia las estructuras de datos
static void init(){
    for (int j = 0; j <= v; j++) {
        marked[j] = false;
        ady[j] = new ArrayList<Integer>();
    }
}

//Recibe el nodo inicial s
static void dfs(int s){
    marked[s]=true;
    int i, next;

    for(i=0; i<ady[s].size(); i++){
        next=ady[s].get(i);
        if(!marked[next]){
            dfs(next);
        }
    }
}

```

---

## 5.3. Dijkstra's Algorithm

Algoritmo que dado un grafo con pesos no negativos halla la ruta mnima entre un nodo inicial s y todos los dems nodos.

```

import java.util.ArrayList;
import java.util.PriorityQueue;

static int v, e; //vertices, arcos
static int MAX=100005;
static ArrayList<Node> ady[] = new ArrayList[MAX];
static int marked[] = new int[MAX];
static long distance[] = new long[MAX];
static int prev[] = new int[MAX];

//Mtodo para limpiar los valores de las estructuras.
//Lllamarlo siempre antes de utilizar el mtodo dijkstra()
static void init() {
    long max = Long.MAX_VALUE;

    for (int j = 0; j <= v; j++) {
        marked[j] = 0;
        prev[j] = -1;
        distance[j] = max;
        ady[j] = new ArrayList<Node>();
    }
}

//Recibe el nodo inicial s
static void dijkstra(int s) {
    PriorityQueue<Node> pq=new PriorityQueue<Node>();
    pq.add(new Node(s, 0)); //se inserta a la cola el nodo Inicial.
    distance[s] = 0;
    int actual, j, adjacent;
    long weight;
    Node x;

    while (pq.size() > 0) {
        actual = pq.peek().adjacent;
        pq.poll();
        if (marked[actual] == 0) {
            marked[actual] = 1;
            for (j = 0; j < ady[actual].size(); j++) {
                adjacent = ady[actual].get(j).adjacent;
                weight = ady[actual].get(j).cost;
            }
        }
    }
}

```



```

        if (marked[adjacent] == 0) {
            if (distance[adjacent] > distance[actual] + weight) {
                distance[adjacent] = distance[actual] + weight;
                prev[adjacent] = actual;
                pq.add(new Node(adjacent, distance[adjacent]));
            }
        }
    }
}

//Retorna en un String la ruta desde s hasta t
//Recibe el nodo destino t
static String path(int t) {
    String r="";
    while(prev[t]!=-1){
        r="-"+t+r;
        t=prev[t];
    }
    if(t!=-1){
        r=t+r;
    }
    return r;
}

static class Node implements Comparable<Node> {

    public int adjacent;
    public long cost;

    public Node(int ady, long c) {
        this.adjacent = ady;
        this.cost = c;
    }

    @Override
    public int compareTo(Node o) {
        if (this.cost >= o.cost) {
            return 1;
        } else {
            return -1;
        }
    }
}

```

```

}

```

## 5.4. FloodFill

Dado un grafo implícito colorea y cuenta el tamaño de las componentes conexas. Normalmente usado en rejillas 2D.

//aka Coloring the connected components

```

static int tam = 1000; //Mximo tamaño de la rejilla
static int dy[] = {1,1,0,-1,-1,-1, 0, 1}; //Estructura auxiliar
    para los desplazamientos
static int dx[] = {0,1,1, 1, 0,-1,-1,-1};
static char grid[][] = new char [tam][tam]; //Matriz de caracteres
static int X, Y; //Tamaño de la matriz

/*Este método debe ser llamado con las coordenadas x, y donde se
    inicia el
recorrido. c1 es el color que estoy buscando, c2 el color con el
    que se va
a pintar. Retorna el tamaño de la componente conexa*/
static int floodfill(int y, int x, char c1, char c2) {
    if (y < 0 || y >= Y || x < 0 || x >= X) return 0;

    if (grid[y][x] != c1) return 0; // base case

    int ans = 1;
    grid[y][x] = c2; // se cambia el color para prevenir ciclos

    for (int i = 0; i < 8; i++)
        ans += floodfill(y + dy[i], x + dx[i], c1, c2);

    return ans;
}

```

## 5.5. Floyd-Warshall's Algorithm

Algoritmo para grafos que halla la distancia mínima entre cualquier par de nodos. Matrix[i][j] guardar la distancia mínima entre el nodo i y el j. Ajustar los tipos de datos según el problema.

```

static int v, e; //vertices, arcos
static int MAX=505;
static int matrix[][]=new int [MAX][MAX];

//Mtodo para limpiar las estructuras de datos
static void init() {
    int i, j;
    for(i=0; i<v; i++){
        for(j=0; j<v; j++){
            matrix[i][j]=-1;
        }
    }
}

static void floydWarshall(){
    int i,j,k, aux;
    k=0;
    while(k<v){
        for(i=0; i<v; i++){
            if(i!=k){
                for(j=0; j<v; j++){
                    if(j!=k){
                        aux=matrix[i][k]+matrix[k][j];
                        if(aux<matrix[i][j] && aux>0){
                            matrix[i][j]=aux;
                        }
                    }
                }
            }
        }
        k++;
    }
}

```

## 5.6. Kruskal's Algorithm

Algoritmo para hallar el arbol cobertor mnimo de un grafo no dirigido y conexo. Utiliza la tcnica de Union-Find(Conjuntos disjuntos) para detectar que aristas generan ciclos.

```

import java.util.ArrayList;
import java.util.Collections;

```

```

static int v, e; //vertices, arcos
static int MAX=100005;
static int parent[]= new int [MAX];
static int rank[]= new int [MAX];
static ArrayList<Edge> edges;
static ArrayList<Edge> answer;

//limpiar las estructuras de datos
static void init() {
    edges=new ArrayList<Edge>();
    answer=new ArrayList<Edge>();
    for (int j = 0; j <= v; j++) {
        parent[j] = j;
        rank[j] = 0;
    }
}

//UNION-FIND
static int find(int i){
    if(parent[i]!=i){
        parent[i]=find(parent[i]);
    }
    return parent[i];
}

static void unionFind(int x, int y){
    int xroot = find(x);
    int yroot = find(y);

    if (rank[xroot] < rank[yroot])
        parent[xroot] = yroot;
    else if (rank[xroot] > rank[yroot])
        parent[yroot] = xroot;

    else{
        parent[yroot] = xroot;
        rank[xroot]++;
    }
}

static void kruskall(){
    Edge actual;
    int aux=0;
    int i=0;

```

```

int x,y;
Collections.sort(edges);

while(aux<(v-1)){
    actual=edges.get(i);
    x=find(actual.src);
    y=find(actual.dest);

    if(x!=y){
        answer.add(actual);
        aux++;
        unionFind(x, y);
    }
    i++;
}

static class Edge implements Comparable<Edge> {

    public int src, dest, weight;

    public Edge(int s, int d, int w) {
        this.src = s;
        this.dest = d;
        this.weight=w;
    }

    @Override
    public int compareTo(Edge o) {
        return this.weight-o.weight;
    }
}

```

## 5.7. Trajan's Algorithm

Algoritmo para hallar los puentes e itsmos en un grafo no dirigido.

```

import java.util.ArrayList;
import java.lang.Math;

static int n, e; //vertices, arcos
static int MAX=1010;

```

```

static ArrayList<Integer> ady[]=new ArrayList [MAX];
static boolean marked[]=new boolean [MAX];
static int prev[]=new int [MAX];
static int dfs_low[]=new int [MAX];
static int dfs_num[]=new int [MAX];
static int itsmos[]=new int [MAX];
static ArrayList<Edge> bridges;
static int dfsRoot, rootChildren, cont;

static void init() {
    bridges=new ArrayList<Edge>();
    cont=0;
    int i;
    for(i=0; i<n; i++){
        ady[i]=new ArrayList<Integer>();
        marked[i]=false;
        prev[i]=-1;
        itsmos[i]=0;
    }
}

static void dfs(int u){
    dfs_low[u]=cont;
    dfs_num[u]=cont;
    cont++;
    marked[u]=true;
    int j, v;

    for(j=0; j<ady[u].size(); j++){
        v=ady[u].get(j);
        if(!marked[v]){
            prev[v]=u;
            //Caso especial
            if(u==dfsRoot){
                rootChildren++;
            }
            dfs(v);
            //PARA ITSMOS
            if(dfs_low[v]>=dfs_num[u]){
                itsmos[u]=1;
            }
            //PARA PUENTES
            if(dfs_low[v]>dfs_num[u]){
                bridges.add(new Edge(Math.min(u,v),Math.max(u,v)));
            }
        }
    }
}

```

```

        dfs_low[u]=Math.min(dfs_low[u], dfs_low[v]);
    }else if(v!=prev[u]){ //Arco que no sea backtrack
        dfs_low[u]=Math.min(dfs_low[u], dfs_num[v]);
    }
}
}

static class Edge{

    public int src, dest;

    public Edge(int s, int d) {
        this.src = s;
        this.dest = d;
    }
}

```

---

## 6. Math

### 6.1. Binary Exponentiation

Realiza  $a^b$  y retorna el resultado modulo  $c$

```

static long binaryExponentiation(long a, long b, long c){
    if (b == 0){
        return 1;
    }
    if (b % 2 == 0){
        long temp = binaryExponentiation(a,b/2, c);
        return (temp * temp) % c;
    }else{
        long temp = binaryExponentiation(a, b-1, c);
        return (temp * a) % c;
    }
}

```

---

### 6.2. Binomial Coefficient

Calcula el coeficiente binomial  $nCr$ , entendido como el número de subconjuntos de  $k$  elementos escogidos de un conjunto con  $n$  elementos.

```

static long binomialCoefficient(long n, long r) {
    if (r < 0 || n < r) {
        return 0;
    }
    r = Math.min(r, n - r);
    long ans = 1;
    for (int i = 1; i <= r; i++) {
        ans = ans * (n - i + 1) / i;
    }
    return ans;
}

```

---

### 6.3. Catalan Number

Guarda en el array Catalan Numbers los números de Catalan hasta MAX.

```

static int MAX = 30;
static long catalanNumbers[] = new long[MAX+1];

static void catalan(){
    catalanNumbers[0] = 1;
    for(int i = 1; i <= MAX; i++){
        catalanNumbers[i] =
            (long)(catalanNumbers[i-1]*((double)(2*((2 * i)-
            1))/(i + 1))));
    }
}

```

---

### 6.4. Euler Totient

Función totient o indicatriz () de Euler. Para cada posición  $n$  del array result retorna el número de enteros positivos menores o iguales a  $n$  que son coprimos con  $n$  (Coprimos: MCD=1)

```

static void totient(int n, int resultados[]){
    boolean aux[]=new boolean[n];
    for(int i=0; i<n; i++) {
        resultados[i]=i;
    }
    for(int i=2; i<n; i++){

```

```

        if(!aux[i]) {
            for(int j=i; j<n ; j+=i){
                aux[j]=true;
                resultados[j]=
                    resultados[j]-(resultados[j]/i) ;
            }
            aux[i] = false;
        }
    }
}

```

---

## 6.5. Gaussian Elimination

Resuelve sistemas de ecuaciones lineales por eliminacin Gaussiana. matrix contiene los valores de la matriz cuadrada y result los resultados de las ecuaciones. Retorna un vector con el valor de las n incongnitas. Los resultados pueden necesitar redondeo.

```

import java.util.ArrayList;

static int MAX = 100;
static int n = 3;
static double matrix[] [] = new double[MAX] [MAX];
static double result[] = new double[MAX];

static ArrayList<Double> gauss() {

    ArrayList<Double> ans = new ArrayList<Double>();
    for(int i=0;i<n;i++){
        ans.add(0.0);
    }
    double temp;
    for (int i = 0; i < n; i++) {
        int pivot = i;
        for (int j = i + 1; j < n; j++) {
            temp = Math.abs(matrix[j][i]) - Math.abs(matrix[pivot][i]);
            if (temp > 0.000001) {
                pivot = j;
            }
        }
        double temp2[] = new double[n];
        System.arraycopy(matrix[i],0,temp2,0,n);
        System.arraycopy(matrix[pivot],0,matrix[i],0,n);

```

```

        System.arraycopy(temp2,0,matrix[pivot],0,n);
        temp = result[i];
        result[i] = result[pivot];
        result[pivot] = temp;

        if (!(Math.abs(matrix[i][i]) < 0.000001)) {

            for (int k = i + 1; k < n; k++) {
                temp = -matrix[k][i] / matrix[i][i];
                matrix[k][i] = 0;
                for (int l = i + 1; l < n; l++) {
                    matrix[k][l] += matrix[i][l] * temp;
                }
                result[k] += result[i] * temp;
            }
        }
        for (int m = n - 1; m >= 0; m--) {
            temp = result[m];
            for (int i = n - 1; i > m; i--) {
                temp -= ans.get(i) * matrix[m][i];
            }
            ans.set(m,temp / matrix[m][m]);
        }
        return ans;
    }
}

```

---

## 6.6. Greatest common divisor

Calcula el mximo comn divisor entre a y b mediante el algoritmo de Euclides

```

static int mcd(int a, int b) {
    int aux;
    while(b!=0){
        a %= b;
        aux = b;
        b = a;
        a = aux;
    }
    return a;
}

```

---

## 6.7. Lowest Common multiple

---

Calculo del mnimo comn mltiplo usando el mximo comn divisor REQUIERE  
mcd(a,b)

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

---

## 6.8. Miller-Rabin

---

La funcin de Miller-Rabin determina si un nmero dado es o no un nmero  
primo. IMPORTANTE: Debe utilizarse el mtodo binaryExponentiation.

```
public static long mulmod(long a, long b, long mod){  
    long x = 0;  
    long y = a % mod;  
    while (b > 0){  
        if (b % 2 == 1){  
            x = (x + y) % mod;  
        }  
        y = (y * 2) % mod;  
        b /= 2;  
    }  
    return x % mod;  
}  
  
public static boolean miller(long p){  
    if (p < 2){  
        return false;  
    }  
    if (p != 2 && p % 2==0){  
        return false;  
    }  
    long s = p - 1;  
    while (s % 2 == 0){  
        s /= 2;  
    }  
    for (int i = 0; i < 5; i++){  
        long a = (long)(Math.random() * 1000000000) % (p - 1) + 1;  
        long temp = s;  
        long mod = binaryExponentiation(a, temp, p);  
        while (temp != p - 1 && mod != 1 && mod != p - 1){
```

```
            mod = mulmod(mod, mod, p);  
            temp *= 2;  
        }  
        if (mod != p - 1 && temp % 2 == 0){  
            return false;  
        }  
    }  
  
    return true;  
}
```

---

## 6.9. Prime Factorization

---

Guarda en primeFactors la lista de factores primos del value de menor a  
mayor. IMPORTANTE: Debe ejecutarse primero la criba de Eratostenes.  
La criba debe existir al menos hasta la raiz cuadrada de value (se  
recomienda dejar un poco de excedente).

```
import java.util.ArrayList;  
  
static ArrayList<Long> primeFactors = new ArrayList<Long>();  
  
static void calculatePrimeFactors(long value){  
    primeFactors.clear();  
    long temp = value;  
    int factor;  
    for (int i = 0; (long)primes.get(i) * primes.get(i) <= value; ++i){  
        factor = primes.get(i);  
        while (temp % factor == 0){  
            primeFactors.add((long)factor);  
            temp /= factor;  
        }  
    }  
    if (temp != 1) {  
        primeFactors.add(temp);  
    }  
}
```

---

## 6.10. Sieve of Eratosthenes

---

Guarda en primes los nmeros primos menores a MAX

```

import java.util.ArrayList;

static int MAX = 10000000;
static ArrayList<Integer> primes = new ArrayList<Integer>();
static boolean sieve[] = new boolean[MAX+5];

static void calculatePrimes() {
    sieve[0] = sieve[1] = true;
    int i;
    for (i = 2; i * i <= MAX; ++i) {
        if (!sieve[i]) {
            primes.add(i);
            for (int j = i * i; j <= MAX; j += i)
                sieve[j] = true;
        }
    }
    for(; i <= MAX; i++){
        if (!sieve[i]) {
            primes.add(i);
        }
    }
}

```

---

## 7. String

### 7.1. KMP's Algorithm

Encuentra si el string pattern se encuentra en el string cadena.

```

import java.util.ArrayList;

static ArrayList<Integer> table(String pattern){
    int m=pattern.length();
    ArrayList<Integer> border = new ArrayList<Integer>();
    border.add(0);
    int temp;
    for(int i=1; i<m; ++i){
        border.add(border.get(i-1));
        temp = border.get(i);
        while(temp>0 && pattern.charAt(i)!=pattern.charAt(temp)){
            if(temp <= i+1){

```

```

                border.set(i,border.get(temp-1));
                temp = border.get(i);
            }
        }
        if(pattern.charAt(i) == pattern.charAt(temp)){
            border.set(i,temp+1);
        }
    }
    return border;
}

static boolean kmp(String cadena, String pattern){
    int n=cadena.length();
    int m=pattern.length();
    ArrayList<Integer> tab=table(pattern);
    int seen=0;

    for(int i=0; i<n; i++){
        while(seen>0 && cadena.charAt(i)!=pattern.charAt(seen)){
            seen=tab.get(seen-1);
        }
        if(cadena.charAt(i)==pattern.charAt(seen))
            seen++;
        if(seen==m){
            return true;
        }
    }
    return false;
}

```

---

## 8. Tips and formulas

### 8.1. ASCII Table

Caracteres ASCII con sus respectivos valores numéricos.

No.	ASCII	No.	ASCII
0	NUL	16	DLE
1	SOH	17	DC1
2	STX	18	DC2
3	ETX	19	DC3
4	EOT	20	DC4
5	ENQ	21	NAK

6	ACK	22	SYN
7	BEL	23	ETB
8	BS	24	CAN
9	TAB	25	EM
10	LF	26	SUB
11	VT	27	ESC
12	FF	28	FS
13	CR	29	GS
14	SO	30	RS
15	SI	31	US

No.	ASCII	No.	ASCII
32	(space)	48	0
33	!	49	1
34	"	50	2
35	#	51	3
36	\$	52	4
37	%	53	5
38	&	54	6
39	'	55	7
40	(	56	8
41	)	57	9
42	*	58	:
43	+	59	;
44	,	60	i
45	-	61	=
46	.	62	¿
47	/	63	?

No.	ASCII	No.	ASCII
64	@	80	P
65	A	81	Q
66	B	82	R
67	C	83	S
68	D	84	T
69	E	85	U
70	F	86	V
71	G	87	W
72	H	88	X
73	I	89	Y

74	J	90	Z
75	K	91	[
76	L	92	\
77	M	93	]
78	N	94	^
79	O	95	-

No.	ASCII	No.	ASCII
96	'	112	p
97	a	113	q
98	b	114	r
99	c	115	s
100	d	116	t
101	e	117	u
102	f	118	v
103	g	119	w
104	h	120	x
105	i	121	y
106	j	122	z
107	k	123	{
108	l	124	
109	m	125	}
110	n	126	~
111	o	127	

## 8.2. Catalan Number

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

Primeros 30 números de Catalán:

n	$C_n$
0	1
1	1
2	2
3	5
4	14
5	42



6	132
7	429
8	1.430
9	4.862
10	16.796
11	58.786
12	208.012
13	742.900
14	2.674.440
15	9.694.845
16	35.357.670
17	129.644.790
18	477.638. 700
19	1.767.263.190
20	6.564.120.420
21	24.466.267.020
22	91.482.563.640
23	343.059.613.650
24	1.289.904.147.324
25	4.861.946.401.452
26	18.367.353.072.152
27	69.533.550.916.004
28	263.747.951.750.360
29	1.002.242.216.651.368
30	3.814.986.502.092.304

8.3. Euclidean Distance

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

8.4. Permutation and combination

Combinación (Coeficiente Binomial): Número de subconjuntos de k elementos escogidos de un conjunto con n elementos

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$$

Combinación con repetición: Número de grupos formados por n elementos,

partiendo de m tipos de elementos.

$$CR_m^n = \binom{m+n-1}{n} = \frac{(m+n-1)!}{n!(m-1)!}$$

Permutación: Número de formas de agrupar n elementos, donde importa el orden y sin repetir elementos

$$P_n = n!$$

Elegir r elementos de n posibles con repetición

$$n^r$$

Permutaciones con repetición: Se tienen n elementos donde el primer elemento se repite a veces , el segundo b veces , el tercero c veces, ...

$$PR_n^{a,b,c...} = \frac{P_n}{a!b!c!...}$$

Permutaciones sin repetición: Número de formas de agrupar r elementos de n disponibles, sin repetir elementos

$$\frac{n!}{(n-r)!}$$

8.5. Time Complexities

Aproximación del mayor número n de datos que pueden procesarse para cada una de las complejidades algoritmicas. Tomar esta tabla solo como referencia.

Complexity	n
$O(n!)$	11
$O(n^5)$	50
$O(2^n * n^2)$	18
$O(2^n * n)$	22
$O(n^4)$	100
$O(n^3)$	500
$O(n^2 \log_2 n)$	1.000
$O(n^2)$	10.000
$O(n \log_2 n)$	$10^6$
$O(n)$	$10^8$

$O(\sqrt{n})$	$10^{16}$
$O(\log_2 n)$	-
$O(1)$	-

### 8.6. mod: properties

1.  $(a \% b) \% b = a \% b$  (Propiedad neutro)
2.  $(ab) \% c = ((a \% c)(b \% c)) \% c$  (Propiedad asociativa en multiplicación)
3.  $(a + b) \% c = ((a \% c) + (b \% c)) \% c$  (Propiedad asociativa en suma)