

# Algoritmos en Java Programación Competitiva

Triforce X

7 de enero de 2016

## Índice

1. I/O	1
1.1. Scanner Rápido . . . . .	1
2. Teoría de Grafos	1
2.1. DFS: Depth First Search . . . . .	1
3. Programación Dinámica (DP)	2
3.1. LIS . . . . .	2
4. Teoría de Números	2
4.1. MulFast . . . . .	2
4.2. PowFast . . . . .	2

## 1. I/O

### 1.1. Scanner Rápido

```
static class Scanner{
    BufferedReader br=null;
    StringTokenizer tk=null;
    public Scanner(){
        br=new BufferedReader(new InputStreamReader(System.in));
    }
    public String next() throws IOException{
        while(tk==null || !tk.hasMoreTokens())
            tk=new StringTokenizer(br.readLine());
        return tk.nextToken();
    }
    public int nextInt() throws NumberFormatException, IOException{
        return Integer.parseInt(next());
    }
    public long nextLong() throws NumberFormatException, IOException{
```

```
        return Long.parseLong(next());
    }
    public double nextDouble() throws NumberFormatException, IOException{
        return Double.parseDouble(next());
    }
}
```

## 2. Teoría de Grafos

### 2.1. DFS: Depth First Search

Búsqueda en Profundidad:

```
static boolean[] visited; //Arreglo de Visitados
static int AdjList[][]; //Lista de Adyacencias

/*
Inputs:
    u: nodo de inicio
    vec: Arreglo que almacena el recorrido
    n: Número de nodos
*/
public static void dfs(int u, Vector<Integer>vec, int n){
    visited[u] = true;
    vec.add(u);
    for(int j=0; j<n; j++){
        if(AdjList[u][j] == 1 && !visited[j]){
            dfs(j,vec, n);
        }
    }
}
```

## 3. Programación Dinámica (DP)

### 3.1. LIS

Multiplicación Rápida:

```
public static int lis(int[] arr, int n) {
    int[] lis = new int[n];
    int i, j, max = 0;

    for (i = 0; i < n; i++) lis[i] = 1;

    for (i = 1; i < n; i++) {
        for (j = 0; j < i; j++) {
            if (arr[i] > arr[j] && lis[i] < lis[j] + 1) lis[i] = lis[j] + 1;
        }
    }

    for (i = 0; i < n; i++) {
        if (max < lis[i]) max = lis[i];
    }

    return max;
}
```

.....

## 4. Teoría de Números

### 4.1. MulFast

Multiplicación Rápida:

```
/*
Inputs:
    a, b: Factores
Outputs:
    ans: Producto de a*b
*/
public static int mul_fast(int a, int b){
    int ans = 0;
    String x;
    while(b>0){
        x = Integer.toBinaryString(b);
        if(x.charAt(x.length()-1) == '1') ans+=a;
        a <<= 1;
        b >>= 1;
    }
}
```

```
    }
    return ans;
}
```

.....

### 4.2. PowFast

Potenciación Rápida:

```
/*
Inputs:
    base: Base de la operación Potenciación
    exp: Exponente de la operación
Outputs:
    pow: base^exp
*/
public static int pow_fast(int base, int exp){
    int pow;

    if(exp == 0) return 1;
    pow = pow_fast(base, exp/2);
    pow = pow*pow;
    if(exp%2==1) pow*=base;
    return pow;
}
}
```

.....