# Olimpiada Informática Española
# Soluciones de los problemas

2013

# Día 1

## Bolsa [Implementación]

**Problema:** Dados los valores iniciales de cotización de algunas empresas y para cada día si su valor ha subido o bajado una unidad, dibujar los cambios de valor que toman las acciones.

**Solución:** Es un problema de implementación, cuya dificultad recae en colocar caracteres /, \y X en el lugar adecuado.

## El caballo hambriento [Grafos]

**Problema:** Contar el mínimo número de movimientos que un caballo de ajedrez debe dar para llegar a casillas especiales sin pasar por casillas prohibidas.

**Solución:** La solución esperada consiste en realizar un BFS sin pasar por las casillas prohibidas.

## Códigos ambiguos [Implementación]

**Problema:** Si asignamos a cada letra un valor $a = 1, b = 2 \ldots$ Tenemos códigos ambiguos como 11 que podría ser *aa* o *k* y códigos no ambiguos como 1010 que solo podría ser *jj*. Se pide que dado *n* y *p* se escriban los *p* primeros códigos de *n* dígitos. Hay un segundo caso en el que se pide escribir solo los no ambiguos.

**Solución:** Es suficiente con iterar por todos los números de *n* dígitos mirando si es o no un código que cumpla las especificaciones hasta llegar a *p* códigos.

## Mecanografía [Implementación]

**Problema:** Dada una palabra y para cada letra una indicación de si la letra correspondiente es correcta o debería ser en realidad la de la izquierda o la de la derecha a la que se encontraría en un teclado, escribir la palabra correcta.

**Solución:** Simplemente iteramos por la palabra cambiando las letras según las especificaciones.

## Evaluación continua (1) [Ad-hoc]

**Problema:** Dado un conjunto ordenado de $n$ números, escribir la mayor suma de $k$ números consecutivos.

**Solución:** Empezamos con la suma de los $k$ primeros números y según avanzamos por los números, actualizamos la suma restando el número de la posición $i - k$ y sumando el de la posición $i$. La solución es la mayor suma que hemos obtenido. De modo que obtenemos la solución en $O(n)$

## El elemento no repetido [Sort, Set, Map]

**Problema:** Dado un conjunto de números en el que todos aparecen 2 veces salvo uno, escribir el número que no está repetido.

**Solución:** Una opción es crear un set y cuando leemos un número lo insertamos si no está previamente en el set y si no lo eliminamos, al final solo quedará el elemento no repetido. Otra opción sería aumentar el valor asignando a un map en una unidad cada vez que veamos un número y después recorrerlo para ver quien tiene asignado el número 1. También se puede resolver ordenando los valores y viendo que número no tiene un valor adyacente igual a sí mismo. Las tres soluciones tienen una complejidad de $O(n \log n)$

Otra solución sería hacer el xor de todos los elementos, obteniendo una solución $O(n)$. Eso es así porque el xor es conmutativo y el xor de dos números iguales es 0.

## Portales interdimensionales [Estructuras de datos]

**Problema:** Dado $n$ números, se realizan $q$ preguntas de dos tipos: cambiar el valor de un número o bien dadas dos posiciones $i, j$ calcular la suma de la diferencia del valor de los números entre $i$ y $j$, es decir si $i \leq j$ deberíamos calcular $|a_i - a_{i+1}| + \cdots + |a_{j-1} - a_j|$

**Solución:**

- Para lograr 60 puntos es suficiente con simular el proceso. La solución sería $O(n \cdot q)$
- Para lograr 80 puntos se puede precalcular la suma cumulativa de las diferencias.
- La solución esperada consiste en realizar un Segment Tree de sumas donde a las hojas del árbol les asignamos las diferencias entre valores adyacentes. Deberemos actualizar las dos hojas que corresponda cuando haya un cambio de valor, así como sus padres. La complejidad es de $O((n+q)\log n)$

# Día 2

## Intersección de agendas [Implementación]

**Problema:** Dado dos conjuntos de intervalos de tiempo en un mismo día, calcular el número de minutos de su intersección.

**Solución:** Basta con iterar por todos los minutos que tiene un día y ver si el minuto pertenece a alguno de los intervalos de ambos conjuntos.

## Árboles [Grafos]

**Problema:** Calcular el mínimo tamaño del mayor subárbol que se puede obtener asignando como raíz del árbol a un nodo cualquiera.

**Solución:** Una solución sería realizar un DFS para saber, para cada arista dirigida, el tamaño del subárbol al que se dirige. Después para cada nodo podemos obtener fácilmente el mayor tamaño de sus subárboles observando sus aristas. El resultado es el mínimo valor que obtengamos.

## Evaluación continua (2) [Estructuras de datos]

**Problema:** Dado un conjunto ordenado de $n$ números tanto positivos como negativos ($a_i$), escribir la mayor suma de $k$ o más números consecutivos.

**Solución:** Para cada posición nos guardamos la mayor suma que podemos conseguir hasta ese número incluyéndolo.

$$prec[i] = max \sum_{j \leq i}^{i} a_j$$

modificando una suma cumulativa. Entonces, consideramos la suma de los primeros $k$ números y, a cada paso, actualizamos la suma, guardando el máximo de la suma y suma + prec[i-k].

## sèver leD [Implementación]

**Problema:** Leer un conjunto de líneas y escribir las lineas leídas de derecha a izquierda.

**Solución:** Leemos el input con getline y giramos las palabras a mano o con la función reverse.

## Typematrix [Dinámica]

**Problema:** Dado un teclado y la posición de dos dedos, asumiendo que se tarda 1ms en moverse de una tecla a una adyacente, calcular el mínimo tiempo que se tarda en escribir una palabra. En algunos casos cada dedo queda restringido a cada mitad del teclado.

**Solución:**

- Si hay la resticción, simular el proceso.
- Si no hay la restricción, hacemos una dinámica con estados: la posición de los dedos y la posición de la letra por la que vamos de la palabra. Y con recurrencia:

$$dp[x_1, y_1, x_2, y_2, pos] = min(|x_1 - l_{pos_x}| + |y_1 - l_{pos_y}| + dp[l_{pos_x}][l_{pos_y}][x_2][y_2][pos+1],$$
$$|x_2 - l_{pos_x}| + |y_2 - l_{pos_y}| + dp[x_1][y_1][l_{pos_x}][l_{pos_y}][pos + 1])$$

dónde $l_{pos}$ es la posición en el teclado de la l-ésima letra.