# Competencia Peruana de Informática Online 2021 - Solucionario

Federación Olímpica Peruana de Informática

02 de mayo de 2021

# Tutorial del problema: "Pares bellos"

## Autor: Racsó Galván

## Grupo 1 (Complejidad Esperada: $O\left(\binom{n+m}{n} \cdot (n+m)^2\right)$)

Para resolver los casos de prueba de este grupo basta con simular todos los posibles resultados usando fuerza bruta y obtener la máxima cantidad de inversiones en $O((n+m)^2)$. Ya que $a$ debe ser una subsecuencia del resultado y el orden relativo de los elementos de ambos arreglos se debe mantener, la cantidad de posibles resultados es $\binom{n+m}{n}$ (estamos considerando tomar $n$ posiciones para los elementos de $a$ de las $n+m$).

## Grupo 2 (Complejidad Esperada: $O\left(nm(n+m)\right)$)

Para resolver los casos de prueba de este grupo debemos notar lo siguiente: Consideremos que hemos agregado los primeros $i$ elementos de $a$ y los primeros $j$ elementos de $b$, entonces si agregamos un elemento $x$ (que puede ser $a_i$ o $b_j$), la cantidad de inversiones en el resultado final aumentará en la cantidad de elementos menores que $x$ en el subarreglo $a[i,n]$ y el subarreglo $b[j,m]$.

Si definimos la función $dp(i,j)$ que nos dará la máxima cantidad de inversiones posibles dado que ya agregamos los primeros $i$ elementos de $a$ y los primeros $j$ elementos de $b$, tendremos la posibilidad de elegir $a_i$ o $b_j$:

- La respuesta si agregamos $a_i$ será igual a $\sum_{k=i+1}^{n}[a_i > a_k] + \sum_{k=j}^{m}[a_i > b_k] + dp(i+1,j)$.
- La respuesta si agregamos $b_j$ será igual a $\sum_{k=i}^{n}[b_j > a_k] + \sum_{k=j+1}^{m}[b_j > b_k] + dp(i,j+1)$.

Donde $[P]$ es 1 si la condición $P$ es verdadera y 0 en caso contrario.

Ya que el hallar los elementos tomará $O((n+m))$, la complejidad final será $O(nm(n+m))$.

## Grupo 3 (Complejidad Esperada: $O\left(\max\{n,m\}^2\right)$)

Para resolver los casos de prueba de este grupo debemos notar que para un estado $(i,j)$ dado, podemos preprocesar el aporte de los elementos de $a$ y de $b$ por separado, de manera que definiremos dos arreglos $A(i,j)$ y $B(i,j)$:

- $A(i,j) = \sum_{k=j}^{m}[a_i > b_k]$.
- $B(i,j) = \sum_{k=i}^{n}[b_j > a_k]$.

Notemos que podemos calcular todos los elementos de ambas tablas en $O(nm)$ aprovechando las recursiones:

- $A(i,j) = A(i,j+1) + [a_i > b_j]$.
- $B(i,j) = B(i+1,j) + [b_j > a_i]$.

Ahora, separaremos el aporte de inversiones recordando que como los dos arreglos deben mantener su orden relativo, entonces la cantidad de inversiones final se puede expresar como:

$$R = \text{Inversiones de } a + \text{Inversiones de } b + \text{Inversiones cruzadas}$$

Donde las inversiones cruzadas son entre elementos de arreglos diferentes. Además los dos primeros términos son constantes para el resultado final.

Podemos calcular las inversiones de ambos arreglos en $O(n^2+m^2)$ y usar la función $dp(i,j)$ para maximizar las inversiones cruzadas. Entonces, replantearemos la forma de calcular la función $dp(i,j)$:

- La respuesta si agregamos $a_i$ será igual a $\sum_{k=j}^{m}[a_i > b_k] + dp(i+1,j)$.
- La respuesta si agregamos $b_j$ será igual a $\sum_{k=i}^{n}[b_j > a_k] + dp(i,j+1)$.

Sin embargo, dadas nuestras definiciones de $A$ y $B$, podemos reemplazar:

- La respuesta si agregamos $a_i$ será igual a $A(i,j) + dp(i+1,j)$.
- La respuesta si agregamos $b_j$ será igual a $B(i,j) + dp(i,j+1)$.

Finalmente la complejidad será de $O(nm + n^2 + m^2)$ o $O(\max\{n,m\}^2)$.

# Tutorial del problema: "Reporte de Tareas"

## Autor: Miguel Miní

### Grupo 1 (Complejidad Esperada: $O\left(n + m + \max_{i=1}^{n} r_i\right)$)

Para resolver los casos de prueba de este grupo debemos aprovechar el hecho de que los identificadores de las tareas no son mayores que $10^6$ y además los intervalos dados son disjuntos, así que se puede mantener un arreglo de tamaño $10^6 + 1$ para marcar cada una de las tareas realizadas.

Luego del paso anterior, podemos obtener los nuevos intervalos haciendo un bucle extra y tomando cada secuencia consecutiva de 1s máxima en cada vez.

### Grupo 2 (Complejidad Esperada: $O\left(n + m\right)$)

Para resolver los casos de prueba de este grupo debemos mejorar la idea anterior en el sentido de que no es necesario marcar las tareas visitadas, pues tenemos los reportes en orden ascendente de identificador, así que podemos unir ambos conjuntos de intervalos (podemos considerar las tareas sueltas como intervalos con un solo punto) tomando el de menor límite inferior a la vez y mezclándolos cuando sea necesario.

La complejidad se reducirá a $O(n+m)$, pues la idea es similar a la función `merge` del algoritmo MergeSort.

**Nota:** Uno puede usar un algoritmo de compresión de coordenadas para resolver el problema de manera similar al grupo 1 en $O((n+m)\log(n+m))$ pero requiere una implementación muy optimizada para poder pasar tanto el TL como el ML.

# Tutorial del problema: "Fiesta OPI"

## Autor: Renato Chavez

### Grupo 1 (Complejidad Esperada: $O\left(a\right)$)

Para resolver los casos de prueba de este grupo nos basta iterar sobre cada posible valor entre 1 y $a$ para verificar en $O(1)$ si el número en cuestión es múltiplo de $n$ o $m$.

### Grupo 2 (Complejidad Esperada: $O\left(1\right)$)

Para resolver los casos de prueba de este grupo debemos usar principio inclusión-exclusión para la respuesta:

$$R = a - |\text{Múltiplos de } n| - |\text{Múltiplos de } m| + |\text{Múltiplos de } n \text{ y } m \text{ simultáneamente}|$$

Los dos primeros conjuntos son sencillos de determinar: La cantidad de múltiplos de $x$ entre 1 y $a$ es $\lfloor \frac{a}{x} \rfloor$, donde $\lfloor r \rfloor$ es igual al máximo entero menor o igual a $r$. Además, si un entero $x$ es múltiplo de $n$ y $m$ al mismo tiempo, es múltiplo de su mínimo común múltiplo, el cual es el menor número que tiene a $n$ y a $m$ como divisores.

Ya que $n$ y $m$ son coprimos, su mínimo común múltiplo es igual a su producto, así que la respuesta final se puede obtener con la siguiente expresión:

$$R = a - \left\lfloor \frac{a}{n} \right\rfloor - \left\lfloor \frac{a}{m} \right\rfloor + \left\lfloor \frac{a}{nm} \right\rfloor$$

En términos de implementación, el producto $nm$ puede generar overflow, así que se puede usar la siguiente igualdad:

$$\left\lfloor \frac{a}{nm} \right\rfloor = \left\lfloor \frac{\lfloor \frac{a}{n} \rfloor}{m} \right\rfloor$$

Así podremos calcular la respuesta en $O(1)$ y sin riesgo a overflow.

# Tutorial del problema: "Racoon City"

## Autor: Walter Erquinigo

## Grupo 1 (Complejidad Esperada: $O\left(n^2|E|\right)$)

Para resolver los casos de prueba de este grupo, nos basta con plantear una función $dp(u, v)$ que nos dé la mayor cantidad de nodos en común que se pueden visitar hasta llegar al $n$. Podemos aplicar programación dinámica sin problemas puesto que el grafo siempre será un DAG (Grafo dirigido y acíclico), así que nunca iremos a un estado anterior.

Consideremos algunas situaciones posibles para ello:

- Si $u = n$ y $v = n$, la respuesta es 1, pues tendríamos que agregar el nodo $n$.
- Si $u = v$, podemos mover cualquiera de los dos nodos a un nuevo estado $(u, x)$ $((x, v))$ y sumarle 1 al máximo de los valores $dp(u, x)$ $(dp(x, v))$.
- Si $u \neq v$, podemos mover cualquiera de los dos nodos a un nuevo estado $(u, x)$ $((x, v))$ y tomar el máximo de los valores $dp(u, x)$ $(dp(x, v))$.

Así, tendremos la siguiente recursión:

$$dp(u, v) = \begin{cases} 1 & u = n \text{ y } v = n \\ \max\left\{ \max_{(u,x)\in E} dp(x, v), \max_{(v,x)\in E} dp(u, x) \right\} + [u == v] & \text{En caso contrario} \end{cases}$$

Esto nos da un total de $O(n^2)$ estados y en cada uno recorremos las dos listas de adyacencia de ambos nodos, así que la complejidad final será de $O(n^2|E|)$.

## Grupo 2 (Complejidad Esperada: $O\left(n^3|E|\right)$)

Para resolver los casos de prueba de este grupo, no podemos aplicar el algoritmo anterior descuidadamente, pues ahora tenemos que considerar 3 supervivientes. Sin embargo, podemos tener un concepto similar al de la solución anterior pero modificado ligeramente para que funcione.

Para empezar, solo aumentaremos 1 a la respuesta si los 3 supervivientes están en el mismo nodo, es decir, un estado de la forma $(u, u, u)$. Por otro lado, consideraremos el orden topológico del grafo para definir cuáles de los 3 nodos van a poder moverse a través de una arista: Siempre moveremos a los supervivientes cuyos nodos aparezcan antes en el orden topológico que los nodos de los demás (en caso de empate, movemos a todos los posibles). Esto nos permitirá contabilizar correctamente los encuentros en común de todos los supervivientes (Pues es como mantener 3 punteros sobre el orden topológico y si siempre movemos el que está más a la izquierda y si se encuentran en un nodo $w$, el estado $(w, w, w)$ siempre será visitado de esa forma).

Esto nos da un total de $O(n^3)$ estados con un procesamiento de $O(|E|)$ por estado y un preprocesamiento de $O(n + E)$, así que la complejidad final es de $O(n^3|E|)$.

**Bonus**: ¿Pueden resolverlo en una mejor complejidad?