Competencia Peruana de Informática Online 2017 - Solucionario

Federación Olímpica Peruana de Informática

28 de mayo de 2017

Tutorial del problema: "Triste"

Autor: Jonathan Durand

Grupo 1 (Complejidad Esperada: $O\left(2^n n^2 \log n\right)$)

Para resolver los casos de prueba de este grupo podemos usar BFS para obtener todas las posibles configuraciones con la menor cantidad de movimientos, guardando la distancia a cada configuración en un `map<string, int>`.

Ya que hay $O(2^n)$ configuraciones posibles, $O(n)$ posibles transiciones (pues solo podemos elegir $n-k+1$ rangos para invertir) y buscar una cadena en el mapa toma $O(n \log n)$, tendremos una complejidad final de $O(2^n n^2 \log n)$.

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

Para resolver los casos de prueba de este grupo debemos notar que el conjunto de pasos para obtener la configuración deseada es único (solo podríamos variar el orden, pero los rangos elegidos serían los mismos), así que solo necesitamos determinar dicho conjunto de pasos para obtener la respuesta.

Consideremos que x_i es igual a 0 si no se invierte el rango $[i, i+k-1]$ y 1 si es que sí lo invertimos. Entonces, podemos analizar las posiciones:

- La primera posición solo es cubierta por un rango, así que x_1 estará definido como 1 si $s_1 = T$ y 0 en caso contrario.
- Luego de este paso, podemos reflejar la inversión usando un `for` en $O(k)$ desde la primera posición.
- Ahora podemos pasar a la siguiente posición, la cual tiene fijados todos los x_i anteriores a la misma, así que solamente resta asignar el valor a su x_i correspondiente, es decir, volvemos al primer paso pero desde la siguiente posición.

Al final debemos verificar que toda la cadena esté llena de F para determinar si se puede o no. La complejidad final es $O(kn)$.

Bonus: Se puede resolver el problema en $O(n)$ usando prefix sums para la función Bitwise XOR.

Tutorial del problema: "El desafío de Jaquerman"

Autor: Rodolfo Mercado

Grupo 1 (Complejidad Esperada: $O\left(n^2 \log n\right)$)

Para resolver los casos de prueba de este grupo nos basta con fijar el valor de i usando un `for` y usar otro `for` para obtener el MCD de todos los enteros en el rango $[i, n]$. Ya que obtener el MCD toma $O(\log n)$, la complejidad final será de $O(n^2 \log n)$.

Grupo 2 (Complejidad Esperada: $O\left(n \log n\right)$)

Para resolver los casos de prueba de este grupo debemos recordar que el MCD es asociativo, así que podemos plantear la siguiente recursión:

$$MCD(n) = n$$

$$MCD(i, \ldots, n) = MCD(MCD(i+1, \ldots, n), i), \forall i < n$$

Así que podemos almacenar en un arreglo de tamaño $n+1$ los resultados de cada i, procesándolos de manera decreciente, para finalmente sumarlos.

La complejidad será de $O(n \log n)$.

Grupo 3 (Complejidad Esperada: $O(1)$)

Para resolver los casos de prueba de este grupo debemos notar que $n-1$ y n son coprimos para todo $n > 1$, así que a partir de $i = n-1$ hacia abajo, todos los MCD serán 1, siendo la única excepción el caso $i = n$, cuyo MCD es n.

Lo anterior nos permite deducir que la respuesta siempre es $2n-1$. La complejidad de operaciones aritméticas constantes es $O(1)$.

Tutorial del problema: "Estaciones Espaciales"

Autor: Aldo Culquicondor

Grupo 1 (Complejidad Esperada: $O(2^n n)$)

Para resolver los casos de prueba de este grupo podemos probar todos los subconjuntos (usando bitmask o backtracking, es indiferente) y verificar si las cápsulas del subconjunto elegido cubren todos los túneles. Ya que hay $O(n)$ túneles, la complejidad se mantiene en $O(2^n n)$.

Grupo 2 (Complejidad Esperada: $O(n)$)

Para resolver los casos de prueba de este grupo podemos definir una función $f(u, usada)$, que nos hallará la cantidad mínima de cápsulas a usar para cubrir todos los túneles del subárbol que tiene como raíz al nodo u dado que si $usada = 0$, entonces la cápsula u no pertenece al subconjunto actual y $usada = 1$ si es que sí pertenece. Es importante notar que definiremos la relación ancestro-descendiente considerando un DFS desde la cápsula 1 (pues esta tiene la puerta de entrada).

Podemos plantear algunos casos dependiendo del valor de $usada$:

- Si $usada = 0$, entonces los túneles de u hacia cada uno de sus hijos deben ser cubiertos por ellos mismos (es la única forma de lograrlo), así que:

$$f(u,0) = \sum_{v \text{ es hijo de } u} f(v,1)$$

- Si $usada = 1$, entonces los túneles de u hacia cada uno de sus hijos ya están cubiertos, así que podemos darnos la libertad de agregarlos o no al conjunto uno por uno, es decir:

$$f(u,1) = 1 + \sum_{v \text{ es hijo de } u} \min\{f(v,0), f(v,1)\}$$

La respuesta total será el mínimo entre $f(1,0)$ y $f(1,1)$.

Si usamos programación dinámica para almacenar las respuestas en una tabla, la complejidad se reduce a la misma de un DFS, que es $O(V+E)$, pero en este caso tenemos que $V = n$, $E = n-1$. Finalmente, la complejidad es $O(n)$. Podemos usar una solución iterativa aprovechando que para todo i, su padre tiene un identificador menor, así que podemos procesar las cápsulas en orden de identificador decreciente y se garantizará que tendremos procesadas las respuestas de sus hijos.

Tutorial del problema: "Sumas"

Autor: Aldo Culquicondor

Grupo 1 (Complejidad Esperada: $O(2^n)$)

Para resolver los casos de prueba de este grupo nos basta probar todos los subconjuntos usando backtracking, es indiferente) y verificar si el subconjunto elegido tiene una suma divisible por n ($\equiv 0 \mod n$).

Grupo 2 (Complejidad Esperada: $O\left(n^2\right)$)

Para resolver los casos de prueba de este grupo podemos usar programación dinámica de tipo problema de la mochila para elegir un subconjunto cuya suma sea igual a 0 módulo n. Para ello podemos definir la función $f(pos, rem)$, que nos dirá si se puede o no elegir un subconjunto de elementos de a cuya suma sea igual a 0 módulo n dado que ya procesamos los primeros pos elementos y al elegir el subconjunto actual se llegó a un residuo rem módulo n. Las transiciones serán de la siguiente manera:

- Si elegimos el elemento a_{pos} para nuestro subconjunto, iremos a $f(pos + 1, (rem + a_{pos}) \mod n)$.
- Si no elegimos el elemento a_{pos} para nuestro subconjunto, iremos a $f(pos + 1, rem)$.

Finalmente, tendremos que reconstruir la respuesta de un DP (problema clásico) para imprimirla. Ya que hay $O(n^2)$ posibles estados y cada uno se procesa en $O(1)$, nuestra complejidad es de $O(n^2)$.

Grupo 3 (Complejidad Esperada: $O\left(n\right)$)

Para resolver los casos de prueba de este grupo podemos usar principio del palomar para notar que siempre existe un subarreglo (elementos con posiciones consecutivas) cuya suma es igual a 0 módulo n. Para ello hay que analizar algunos casos respecto a los prefijos (sea la suma de los primeros i elementos p_i):

- Si existe algún $p_i \equiv 0 \mod n$, entonces el rango $[1, i]$ es una respuesta válida.
- Si no existe algún $p_i \equiv 0 \mod n$, entonces todos los p_i están en el rango $[1, n-1]$. Sin embargo, tenemos n valores p_i, así que por principio del palomar existirán dos posiciones i y j $(i < j)$ tales que $p_i \equiv p_j \mod n$, así que si restamos p_i de p_j obtendremos $p_i - p_j \equiv 0 \mod n$, lo cual significa que el subarreglo $[i + 1, j]$ es una respuesta válida.

Podemos almacenar en un arreglo auxiliar pos_r la última posición i tal que $p_i \equiv r \mod n$ y así consultar en $O(1)$ la existencia de un valor repetido. La complejidad final será de $O(n)$, pues igual no imprimiremos más de n elementos.