Problema 2

Link
https://codeforces.com/problemset/problem/1777/F

Definicion
Se te da un arreglo `a` que consiste en `n` enteros no negativos.

La "apatía" de un subarreglo $a_l, a_{l+1}, ..., a_r$ se define como

$$\max(a_l, a_{l+1}, ..., a_r) \oplus (a_l \oplus a_{l+1} \oplus ... \oplus a_r)$$

donde $\oplus$ denota la operación XOR bit a bit.

Encuentra la apatía máxima entre todos los subarreglos.

Conceptos claves

Prefijo XOR
El prefijo XOR se define como la operacion XOR a todos los elementos desde el inicio de un array `a` hasta un indice `i`. Formalmente queda

$$\text{prefix}[i] = a_1 \oplus a_2 \oplus ... \oplus a_i$$

La funcion XOR possee propiedades conmitativa, transitiva y distributiva, y no solo eso, sino tambien posee la propiedad de que $a \oplus a = 0$. Lo cual hace esta operacion muy maleable a la hora de tratar con el prefijo ya que dado `l` y `r` enteros donde `l < r`

$$\text{XOR}(a_1, a_2, ..., a_l) = \text{prefix}[l]$$

$$\text{XOR}(a_1, a_2, ..., a_l, a_{l+1}, ..., a_r) = \text{prefix}[r]$$

$$\text{Y como } a_i \oplus a_i = 0$$

$$\therefore \text{XOR}(a_l, a_{l+1}, ..., a_r) = \text{prefix}[r] \oplus \text{prefix}[l-1]$$

Por tanto hallar el XOR de un subarreglo es equivalente a hallar el XOR de los prefijos de los indices `l` y `r` y hacer la operacion XOR entre ellos.

Trie
El Trie es una estructura de datos que permite almacenar y buscar cadenas de strings y en este case en concreto secuencias de bits de manera eficiente. En este caso, cada valor de prefijo XOR puede verse como una secuencia de bits, lo que convierte al Trie en una herramienta adecuada para buscar combinaciones óptimas de XOR entre subarreglos.

Solucion usando DP
Voy primero a describir la solucion y despues mas adelante por que es correcta a profundidad.

Pasos del Algoritmo:

1. **Inicializar el Arreglo de Trie**:
   - El arreglo Trie `t` se inicializa de tal manera que para cada índice `i`, almacena los valores de prefijo XOR hasta el índice `i-1`.
   - Esta parte es crucial porque nos permite calcular eficientemente el XOR de subarreglos que terminan antes o en el índice `i`. La idea clave es que el XOR de cualquier subarreglo puede derivarse de la diferencia entre los prefijos XOR.

- La razón por la que insertamos el prefijo XOR hasta `i-1` es que, al calcular la "apatía" de los subarreglos que comienzan en o después del índice `i`, queremos comparar el prefijo XOR actual con los de los subarreglos anteriores, demostrado anteriormente.

2. **Ordenar los Elementos en Orden Ascendente**:
   - Ordenar los elementos por valor se hace para procesar los subarreglos en un orden específico. Al ordenar, procesamos primero los elementos más pequeños, asegurando que podamos usar técnicas de programación dinámica (como la fusión de Tries) de manera más eficiente. Esto es similar al **algoritmo de Mo**, donde ordenar las consultas ayuda a reducir el reprocesamiento innecesario.
   - Procesar primero los elementos más pequeños garantiza que cuando procesamos un elemento más grande, ya tenemos los prefijos XOR de los subarreglos dominados por elementos más pequeños. Esto nos permite manejar cada elemento una sola vez y calcular la "apatía" de manera eficiente.

3. **Calcular los Límites Izquierdo y Derecho Usando una Pila Monótona**:
   - Por cada elemento del arreglo, usamos una pila monotona para encontrar los elementos más cercanos a la izquierda y a la derecha que son mayores que el elemento actual. Esto ayuda a determinar el rango de subarreglos donde el elemento actual es el máximo.
   - Específicamente:
     - El límite izquierdo es el elemento más grande más cercano a la izquierda, por lo que cualquier subarreglo que contenga el elemento actual como máximo solo puede extenderse a la izquierda hasta este límite.
     - El límite derecho es el elemento más grande más cercano a la derecha, por lo que el subarreglo solo puede extenderse a la derecha hasta este límite.
   - Esto reduce el espacio de búsqueda para cada elemento, asegurando que solo calculemos la "apatía" para subarreglos donde el elemento actual está garantizado como máximo.

4. **Elegir el Subarreglo Más Pequeño entre el Izquierdo y el Derecho**:
   - Para cada valor (en el arreglo ordenado), se verifica si la mitad izquierda o la derecha del subarreglo (definida por los límites izquierdo y derecho) es más pequeña.
   - Por qué se hace esto: Esta optimización minimiza el número de operaciones al actualizar el Trie. Siempre trabajamos primero en el subarreglo más pequeño para reducir el número de inserciones y consultas de prefijo XOR.
   - Una vez que determinamos el subarreglo más pequeño, realizamos lo siguiente:
     - Calculamos la "apatía" máxima consultando el Trie opuesto (es decir, si estamos procesando el lado izquierdo, consultamos el lado derecho y viceversa).
     - La consulta usa $\text{prexor}_j \oplus a[x]$, como se mencionó correctamente, para calcular el XOR del subarreglo que incluye `a[x]`.

5. **Fusionar los Tries**:
   - Después de calcular la "apatía" para los subarreglos que incluyen a (a[x]) como el máximo, fusionamos los dos Tries (izquierdo y derecho).
   - El Trie fusionado ahora almacenará los prefijos XOR para ambos subarreglos y permitirá realizar consultas de manera eficiente sobre subarreglos más grandes.
   - Esta fusion se hace desde el Trie mas pequeno al mas grande, para minimizar la cantidad de operaciones de insercion, algo exactamente igual a lo que se hace en un **disjoint set** con sus uniones.