Problema #2: Fixed Points

September 20, 2024

Contents

1 Problemática

1.1 ¿De qué trata el problema?

Considera una secuencia de números enteros $a_1, a_2, ..., a_n$. En un movimientopuedes seleccionar cualquier elemento de la secuencia y eliminarlo. Luego de que un elemento se elimine todos los demás elementos a su derecha son desplazados a la izquierda una posición, por lo que no quedan elementos vacíos en la secuencia. Luego de hacer un movimiento la longitud de la secuencia didminuye en 1. Los índices de los elementos se recalculan luego de esto.

Dada la secuencia $a_1, a_2, ..., a_n$ y un número k, debes encontrar la menor cantidad de movimientos que se deben hacer para que la secuencia resultante tenga al menos k elementos que sean iguales a sus índices, es decir, que la secuencia resultante $b_1, b_2, ..., b_m$ tenga al menos k índices i tal que $b_i = i$

1.2 Entrada

Se reciben dos enteros n (longitud de la cadena) y k (número de elementos que deben coincidir con su índice), asi como la secuencia $a_1, a_2, ..., a_n$ $(1 \leq a_i \leq n)$. Los números de la secuencia no tienen que ser diferentes necesariamente.

1.3 Salida

Se debe devolver un entero x $(0 \leq x \leq n)$, que represente la cantidad mínima de movimientos que se pueden hacer para que en la secuencia resultante existan al menos k elementos iguales a su índice. Si no es posible encontrar dicha secuencia se devuelve -1.

2 Solución

2.1 Idea general de la solución

Para resolver el problema vamos a utilizar un efoque de programación dinámica (dp). Vamos a definir como **índice fijo (Fixed Point)** de un array a a un elemento i tal que $a[i] = i$. El problema consiste en lograr que en la secuencia de entrada hayan al menos k índices fijos haciendo la menor cantidad de movimientos.

Para esto entonces vamos a crear una tabla de dp de $n \times n + 1$ donde $dp[i][j]$ representa la máxima cantidad de índices fijos que se pueden con-

seguir en los primeros i elementos de la secuencia de entrada haciendo exactamente j movimientos. Son n filas porque hay n elementos en la secuencia y $n+1$ columnas porque al final se pueden hacer $n+1$ movimientos en la secuencia (porque se empieza por 0, es decir, se pueden hacer 0 movimientos, o 1 movimiento, o 2 movimientos, o ..., o n movimientos, siendo un total de $n+1$ casos).

La respuesta estará en la última fila del dp, que es la que contiene información sobre los índices fijos que se pueden conseguir en los n elementos de la secuencia. Más concretamente la respuesta será el primer j (menor cantidad de movimientos) de esta fila que cumpla que el dp en esa posición es mayor o igual que k.

2.2 Explicación del código

Luego de leer los datos de entrada se realiza una transformación sobre la secuencia de entrada *nums* que consiste en lo siguiente: $nums[i] = i - nums[i] + 1$. Con esto lo que se logra es que en cada posición de *nums* se guarde la distancia que hay entre el valor original y el índice. Por ejemplo si $nums = [5, 2, 2, 4]$ entonces luego de la transformación $nums = [-4, 0, 1, 0]$, nótese que cada 0 es un índice fijo. Esto será útil en la construcción de la tabla de dp.

Procedemos entonces a crear la tabla de dp. Se definen primeramente los casos base, es decir, los índices fijos que puede haber analizando solo el primer elemento de *nums* y luego se construye el resto de la siguiente manera: $dp[i][j]$ es decir, la cantidad máxima de índices fijos que se pueden lograr con los primeros i elementos de la secuencia, es el máximo entre:

- La cantidad de índices que había en los primeros $i-1$ usando j movimientos, incrementado en 1 si el elemento i es un índice fijo. Nótese que `int(nums[i] == j)` realiza precisamente el cómputo de si el elemento i es índice fijo, ya que, si $nums[i] = j$ significa que la distancia que había entre el valor original de la secuencia con su índice, y la cantidad de elementos eliminados a su izquierda es igual, por tanto dicha distancia se ha reducido a 0, y este elemento es entonces un índice fijo.

- La cantidad de índices que había en los primeros $i-1$ usando $j-1$ movimientos, ignorando el caso en que $j = 0$.

Se verá con más detalles en la demostración formal pero básicamente el primer caso se corresponde con la acción de no eliminar el elemento i

de la secuencia y usar todos los movimientos en los elementos anteriores, mientras que el segundo caso es la acción de eliminar el elemento i (gastando un movimiento), y usando los $j-1$ movimientos restantes en los elementos anteriores.

Finalmente se devuelve la respuesta de la forma que se explicó anteriormente, el menor j tal que $dp[-1][j] \geq k$. De no cumplirse esto entonces el ejercicio no tiene solución y se devuelve -1.

2.3 Demostración formal

Para demostrar la correctitud del algoritmo debemos no solo demostrar que el enfoque de dp resuelve el problema de manera correcta, sino que debemos demostrar las propiedades de **subestructura óptima** y **subproblemas superpuestos**, para que sea válido la aplicación del enfoque dp al problema. Sea a el array de entrada y k la cantidad de índices fijos que se necesitan para resolver el ejercicio.

2.3.1 Subestructura óptima

Esta propiedad se refiere a que la solución óptima de un problema se basa en la solución de subproblemas más pequeños. Sea $F(n, j)$ el máximo número de índices fijos que se pueden obtener en los primeros n elementos de a utilizando exactamente j movimientos. Consideremos los siguientes subproblemas:

- $S_1 = F(n-1, j)$: El máximo número de índices fijos que hay en los primeros $n-1$ elementos usando j movimientos.
- $S_2 = F(n-1, j-1)$: El máximo número de índices fijos que hay en los primeros $n-1$ elementos usando $j-1$ movimientos

Podemos definir entonces:

$$F(n, j) = max(S_1 + int(a[n] == j), S_2)$$

.

Veamos por qué. Para una instancia (n, j) del problema existen 3 formas posibles de proceder:

1. No eliminar el elemento $a[n]$ siendo este un índice fijo ($a[n] = n$): En este caso para calcular $F(n, j)$ debemos tomar la cantidad de índices fijos que teníamos en los primeros $n-1$ elementos usando j movimientos

(ya que no usaremos ningún movimiento para n, por tanto usaremos los j movimientos en los $n-1$ primeros valores). Guardaremos esta cantidad $+1$ ya que el elemento n es un índice fijo. Entonces en este caso: $F(n,j) = S_1 + 1$

2. No eliminar el elemento $a[n]$ sin ser este un índice fijo ($a[n] \neq n$): Mismo razonamiento que en el caso anterior pero esta vez no sumamos 1 al final, ya que $a[n]$ no incrementa la cantidad de índices fijos. En este caso: $F(n,j) = S_1 + 0$

3. Eliminar el elemento $a[n]$ sin importar si es un índice fijo o no: En este caso para calcular $F(n,j)$ debemos tomar la cantidad de índices fijos que teníamos en los primeros $n-1$ elementos usando $j-1$ movimientos (ya que usamos un movimiento en n, nos quedan $j-1$ movimientos para usar en los restantes $n-1$). Esta decición es importante ya que la eliminación de $a[n]$ puede hacer que elementos posteriores de a se conviertan en índices fijos. En este caso $F(n,j) = S_2$

Notemos que en el caso 1 la solución óptima de $F(n,j)$ se apoya en la solución del subproblema S_1+1, en el caso 2 se apoya en S_1 y en el caso 3 se apoya en S_2. Luego $F(n,j)$ será la mayor cantidad de índices fijos que haya en $S_1 + int(a[n] == j)$ (recordemos que esa expresión booleana devueve 1 si $a[n]$ es índice fijo) y en S_2, por tanto $F(n,j) = max(S_1 + int(a[n] == j), S_2)$. Vemos que calcular los índices fijos en n depende de que se hallan calculado correctamente los índices en $n-1$, luego se cumple la propiedad de subestructura óptima.

2.3.2 Subproblemas superpuestos

Esta propiedad se refiere a que la solución de un subproblema es utilizada en más de un problema mayor, y que por tanto sin el enfoque dp se estarían realizando cálculos repetidos. Sean los problemas:

- $F(n,j) = max(F(n-1,j) + int(a[n] == j), F(n-1,j-1))$
- $F(n,j+1) = max(F(n-1,j+1) + int(a[n] == j), F(n-1,j))$

Como podemos ver, el resultado del subproblema $F(n-1,j)$ es reutilizado en ambos subproblemas, lo cual es un ejemplo de que el dp está ayudando a evitar la redundancia en los cálculos, y por tanto se cumple la propiedad de problemas superpuestos.

2.3.3 Correctitud del dp

Vamos a demostrar que una vez terminado el dp, la solución se encuentra en la última fila (fila $n-1$), en el menor i que cumpla que $dp[n-1][i] \geq k$. Para esto vamos a hacer una inducción por los subproblemas que trabaja el dp:

Caso base: $n = 0$

En este caso estamos analizando la cantidad de índices fijos que se pueden conseguir mirando solo el primer elemento del a. Tenemos:

- $dp[0][1] = 0$: Ya que si solo tenemos un elemento y lo eliminamos no se tiene ningún índice fijo
- $dp[0][0] = int(nums[0] == 0)$: Es decir, con un elemento y sin movimientos se tiene un índice fijo si dicho elemento es un índice fijo, de lo contrario se tiene 0.

Luego para $n = 0$ la tabla de dp está bien construida para todo j, es decir, el resto de valores de j no son relevantes pues no se puede eliminar más de un elemento si solo se tiene 1. En general es fácil ver que $dp[i][j]$ no es relevante para el problema si $j - i > 1$.

Hipótesis de inducción:

Para todo i ($i \leq n-1$) y para todo j relevante se cumple que $dp[i][j]$ contiene la mayor cantidad de índices fijos que se pueden conseguir considerando los i primeros elementos y realizando j movimientos.

Paso inductivo: Demostremos que $dp[n][j]$ está bien calculado. De la demostración de la propiedad de subestructura óptima tenemos que:

$$F(i,j) = max(F(i-1,j) + int(a[n] == j), F(i-1,j-1))$$

Donde $F(i,j)$ calcula correctamente el máximo número de índices fijos que se pueden obtener en los primeros n elementos de a utilizando exactamente j movimientos. Luego $dp[i][j] = F(i,j)$ y en particular $dp[n][j] = F(n,j)$. Sustituyendo en la fórmula de F tenemos:

$$dp[n][j] = max(dp[n-1][j] + int(a[n] == j), dp[n-1][j-1])$$

Como $dp[n-1][j]$ y $dp[n-1][j-1]$ están bien calculados por hipótesis de inducción entonces podemos decir que $dp[n][j]$ está bien calculado para todo j, o lo que es lo mismo, en la fila n del dp se tienen calculados las máximas cantidades de índices fijos que se pueden obtener en todo el array a usando todas las posibles cantidades de movimientos.

Por tanto la respuesta al ejercicio es el menor j tal que $dp[n][j] \geq k$, es decir, la menor cantidad de movimientos que se pueden hacer para que en a hayan al menos k índices fijos.

2.3.4 Cerrando la demostración

Como podemos ver, el algoritmo satisface las restricciones del problema y devuelve una solución correcta. En todo momento se tienen en cuenta todos los casos posibles (eliminar o no cada elemento), y siempre se detecta a los índices fijos en el momento y la situación correcta, por lo que cada subproblema se resuelve de manera óptima y los problemas superiores eligen la solución más óptima de los subproblemas de los que depende, lo que hace que cada instancia del problema esté correctamente calculada en el dp. Por esta razón la solución devuelta es la óptima, y es fácil ver que el algoritmo siempre para, ya que la tabla de dp se llena se manera progresiva en $O(n^2)$ operaciones.

2.4 Complejidad espacial y temporal

La transformación inicial de la secuencia de entrada se hace en $O(n)$, y la respuesta se da leyendo solo una fila del dp, lo que tambien es $O(n)$, por tanto la complejidad temporal va estar dada por el coste de la construcción del dp. Como la matriz del dp es de $n \times n+1$ y se construye de una sola pasada, la complejidad temporal total del algoritmo es de $O(n \times n+1)$, o lo que es lo mismo $O(n^2)$.

Con la complejidad espacial pasa lo mismo, el array de entrada ocupa $O(n)$ mientras que la tabla del dp ocupa $O(n^2)$, siendo esta segunda la complejidad espacial total del algoritmo.