David Marquez Mínguez

47319570Z

Ejercicios TEMA 3

ALGORITMIA Y COMPLEJIDAD

# **EJERCICIO 4**

Se tiene un robot cuyo único objetivo es poner corchos en botellas. Existen N corchos y N botellas. Hasta ahora el problema parece simple, pero existen una serie de restricciones las cuales lo complican. E primer lugar las botellas y los corchos no son todos del mismo tamaño, es decir, cada corcho es especifico de cada botella. Por lo tanto, lo único que debemos hacer es comprar corchos con botellas, ya que nuestro robot es capaz de detectar si un corcho es más grande, más pequeño o del tamaño perfecto para fada botella. Otra importante restricción es que el robot no puede comparar botellas entre ellas, tampoco corchos.

El objetivo es embotellar todas las botellas de la manera más optima posible, y con optimo nos referimos a lo más rápido posible.

Se nos ocurre emplear dos estructuras de datos, una para las botellas y otra para los corchos. La idea de dicho algoritmo será ir creando parejas de botellas y tapones donde ambas tengan el mismo valor. Tendremos dos pivotes, uno para las botellas y otro para los tapones. Algo importante y a tener en cuenta sobre esta idea es que el pivote\_tapon == pivote\_botella, es decir si elegimos como pivote para los tapones el tapón de tamaño 5, debemos elegir como pivote para las botellas, la botella de tamaño 5.

Los pasos del algoritmo son los siguientes:

1. Elegimos el pivote de los tapones de forma aleatoria
2. Determinamos de que tapón se trata buscándolo en la lista de tapones
3. Buscamos el pivote de las botellas que coincida con el pivote de los tapones, siguiendo la misma filosofía que antes, es decir, pivote\_tapon == pivote\_botella.
4. Aplicamos divide y vencerás.

Para entender mejor el algoritmo, vamos a realizar un pequeño ejemplo, donde empelaremos 4 botellas de distintos tamaños y 4 corchos correspondiente a cada botella.

Tenemos varias botellas y varios corchos, donde cada botella corresponde con un corcho determinado. Los corchos y las botellas están desordenados por lo que no sabemos que botella corresponde con cada corcho. A continuación, vamos a explicar el algoritmo que se ha creado.

|  |  |  |  |
| --- | --- | --- | --- |
| Tamaños de botellas: | | | |
| 6 | 4 | 3 | 9 |

Lista de botellas, usando como pivote el tapón:

Imaginemos que el pivote de tapones elegido es el 6, todas aquellas botellas menores a 6, las almacenamos en una estructura de datos que llamaremos botellas\_izq, en este caso serian, botella4 y botella3. Todas aquellas botellas mayores a 6, en este caso la botella9, las almacenamos en botellas\_der. En cuanto a la botella6, como estamos usando el tapon6 como pivote, buscamos directamente la botella6 y las emparejamos.

|  |  |  |  |
| --- | --- | --- | --- |
| Tamaños de corchos: | | | |
| 4 | 9 | 6 | 3 |

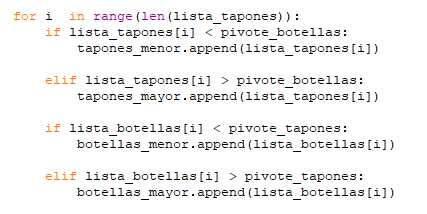
Lista de tapones usando como pivote la botella:

Para este caso aplicamos la misma filosofía que antes, si el pivote de botellas elegido es el 6, todos aquellos tapones menores a 6, los almacenamos en una estructura de datos que llamaremos tapones\_izq. Todos aquellos tapones mayores a 6, los almacenamos en tapones\_der. En cuanto a latapon6, como estamos usando botella6 como pivote, buscamos directamente el tapon6 y los emparejamos.

Una vez sabemos cómo funciona el algoritmo, creamos un script de Python que lo implemente. Debemos tener varias consideraciones en cuenta. En primer lugar, las condiciones de parada del algoritmo, las cuales serán las siguientes:

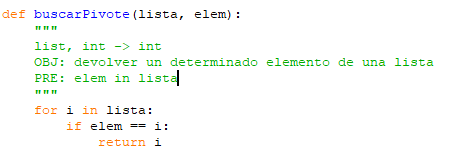
1. Si el tamaño del array es 1, emparejamos directamente la botella con el tapón ya que tendrán el mismo número indicativo.
2. Si una de las dos listas esta vacía, mostraremos que están vacías simplemente.
3. En caso contrario haremos una iteración más del algoritmo.

Antes de mostrar el algoritmo al completo, vamos a explicar una de las partes más importantes del mismo.



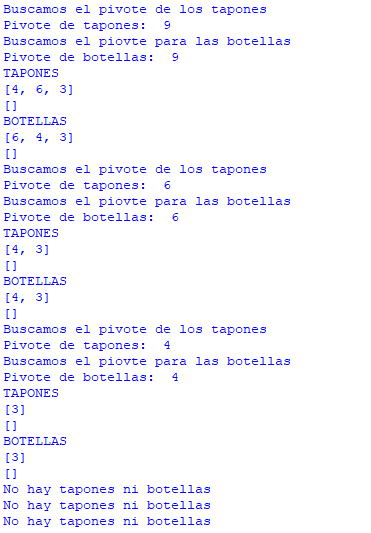
En esta parte, nos encargamos de separar los elementos menores que el pivote a la izquierda y los elementos mayores que el pivote a la derecha. Como no podemos comprar tapones y botellas entre si compararemos el pivote del tapón con el numero de botella. En el caso en el que el numero de botella sea menor que el del tapón, lo colocaremos en la lista de botellas menor. En caso contrario, lo almacenaremos en la lista de botellas mayor. Se realiza el mismo proceso con los tapones y el pivote de la botella.

A continuación, se muestra el algoritmo al completo, donde empleamos una función auxiliar que se encarga de buscar un elemento en concreto dentro de una lista:





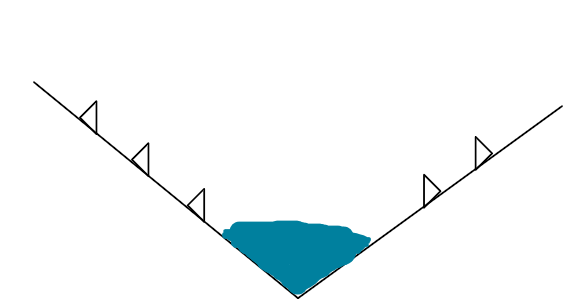
La salida del programa usando el ejemplo propuesto anteriormente, es la siguiente:



De esta forma, implementamos este algoritmo de divide y vencerás, muy parecido a quicksort, para estas dos listas; teniendo en cuenta las condiciones del problema que nos limitaban a comparar botellas con tapones y tapones con botellas.

# EJERCICIO 6

En este problema se pretende que nuestro individuo India Croft pueda cruzar el llamado Valle de las Sombras. Un puente cruza dicho valle que tiene una pendiente de bajada y otra pendiente de subida al salir. Justo debajo del puente hay un rio, pero no se sabe su ubicación. En las pendientes hay rocas afiladas, por lo que no podemos bajarlas sin más. El objetivo es que nuestro individuo encuentre lo más rápido posible la ubicación del rio y se descuelgue del puente para caer en él. Para ello, se dispone de un puntero laser que mide distancias. Para conocer mejor el problema se presenta un dibujo de este:



Para ello creamos un algoritmo que nos devuelve la distancia desde el comienzo del puente desde la cual India Croft deberá saltar.

Las estructuras de datos creadas para el algoritmo son las siguientes:

* Una lista de pares de distancias-alturas que contara con las medidas que se han realizado de ambas, es decir, medidas de la distancia desde el comienzo del puente al punto dado y la altura que hay en el valle en ese punto. Por ejemplo:

|  |  |
| --- | --- |
| Distancia desde el inicio del puente | Altura |
| 1 | 5 |
| 5 | 10 |
| 15 | 200 |

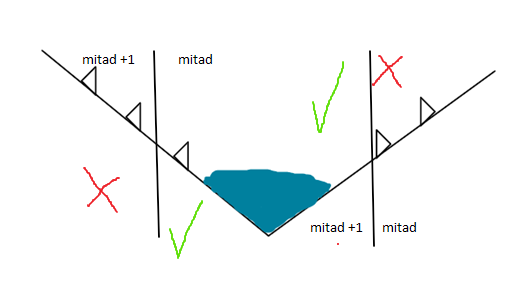
En este ejemplo, estamos diciendo que a un metro de distancia del puente hay una altura de 10 metros, a 5 metros del puente hay una altura de 10 metros, y que, a 15 metros del puente, hay una altura de 200 metros.

* Una posición inferior de la lista que entra en nuestro estudio, en nuestro caso 0.
* Una posición superior de la lista que entra en nuestro estudio en nuestro caso, el tamaño de la lista -1.

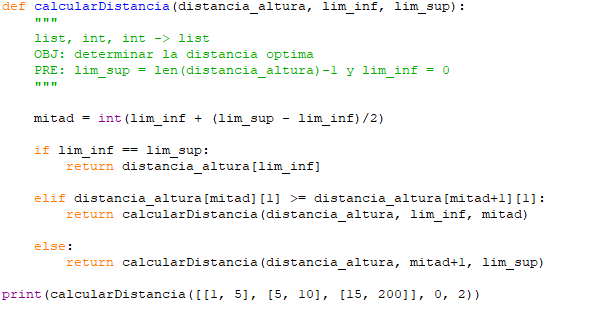
Una vez explicado el problema y las estructuras de datos que se usaran en el mismo, pasamos a explicar el funcionamiento del algoritmo:

1. En primer lugar, se calcula la posición intermedia de nuestra lista de distancia\_altura.
2. Si solo disponemos de una medición, es decir, de un solo par, distancia-altura, devolvemos dicho valor.
3. Si disponemos de más de una medición, tendremos que ver cual es la medida que mas nos interesa. En este caso la medida que mas nos interesa es aquella más cercana al rio.

Para la posición intermedia de la lista de medidas tenemos que ver si las alturas en esa zona siguen descendiendo o empiezan a ascender. Por ejemplo: si la posición medio corresponde a la medida X tenemos que ver si la altura de la medida X+1 es menor que la altura de la medida X. Si eso es así entonces tendremos que estudiar la mitad izquierda del vector. En caso contrario, es decir, si la altura de la medida X+1 es mayor que la altura de la medida X entonces nos quedaremos con la mitad derecha del vector de medidas. Para entender mejor el algoritmo mostramos una imagen:



Se muestra el algoritmo implementado a continuación, así como la salida por pantalla del ejemplo expuesto anteriormente.



La salida del programa es la siguiente:



Esta es la distancia más optima que nuestro individuo debería tomar.

A continuación, se realizará un análisis de complejidad de nuestro algoritmo para asegurarnos de que nuestro individuo escapará en algún momento.

Teniendo en cuenta que en este algoritmo solo se realiza una llamada recursiva, ya que, o tenemos una sola medida, o nos quedamos con la mitad izquierda, o nos quedaos con la mitad derecha, podemos estar seguros de que k = 1. En cuanto al resto de operaciones, como no hay bucles y son todo operaciones básicas, O(1), p = 0. El tamaño del subproblema es la mita del original, b = 2. Por lo tanto , la complejidad del algoritmo es de orden logarítmica, es decir, nuestro individuo podrá escapar del Valle de las Sombras.