Prueba de oposición

Victoria Klimkowski

Septiembre 2024

1 Enunciado

Ejercicio 2 de la práctica 6 1C 2023 de Algoritmos y Estructuras de Datos (ex Algo 2).

Tenemos un arreglo $a = [a_1, a_2, \ldots, a_n]$ de n enteros distintos (positivos y negativos) en orden estrictamente creciente. Queremos determinar si existe una posición i tal que $a_i = i$. Por ejemplo, dado el arreglo $a = [-4, -1, 2, 4, 7]$, $i = 4$ es esa posición.

Diseñar un algoritmo *Dividir y Conquistar* eficiente (de complejidad de orden estrictamente menor que lineal) que resuelva el problema. Calcule y justifique la complejidad del algoritmo dado.

El approach de esta prueba de oposición es cercana a una transcripción de las cosas que diría en una clase para la resolución, marcando en **violeta** las cosas que haría. Se asume haber ido a la teórica donde se introduce el teorema maestro pero sin asumir ejemplos, se puede empezar repasando lo que enuncia y los casos que contempla. Además se asume un moderado manejo de recursión y de algoritmos. Muchas de las preguntas que aparecen a lo largo del documento son, en efecto, preguntas que haría en la clase a los alumnos para que resulte más dinámica y mantener la atención.

2 Resolución

En este ejercicio contamos con elementos que están ordenados y queremos saber si dado uno en particular lo podemos localizar. Veamos qué estrategias se nos ocurren.

Sabemos que tenemos n enteros **(mostraría un array ordenado, sin pérdida de generalidad, de 6 enteros $a = [-5, -3, -1, 1, 2, 3]$ que iría descubriendo de a uno en el proyector)**. Pregunto, ¿hay algún elemento que coincida con su posición? y... Veamos, ¿En la primera posición? No, sigo buscando. Capaz en la segunda, mo... sigo, ¿la tercera? No... ¿En la cuarta? ¿Quinta? ¿Sexta? No... Qué mal, recorrimos todos los elementos del array. ¿Se nos permitía eso?

¿Cuál es la complejidad de esa estrategia? Es lineal ¿Alguien no lo ve? ¿Que por qué es lineal? Porque visitamos todos los elementos del array una vez. ¿Qué nos pedía la consigna sobre la complejidad? Complejidad estrictamente menor a lineal. Perdimos. ¿Qué nos faltó? ¿Aprovechamos que estaba ordenado crecientemente? No.

Veamos si encontramos ejemplos de la vida cotidiana donde tengamos que aprovechar la existencia de elementos ordenados para pensar si intuitivamente se nos ocurre una mejor estrategia que mirar todos los elementos.

Remontémonos a 3er grado, cuando no nos dejaban usar celulares en el aula y nos pedían que buscaramos algo en un diccionario. Nadie de acá se iba a poner a leer flor de libro para buscar una palabra. ¿No? ¿Cómo era? Si teníamos que buscar la palabra "profano" por ejemplo. Agarrabas el libro y te ponías a buscar más o menos de la mitad para adelante, si la primer palabra que veías era una que empezaba con T sabías que te habías pasado, si había una que empezaba con N sabías que estabas más atrás. ¿De dónde viene esta intuición? **(mostraría el abecedario en el proyector)** ¡De saber que las letras tienen un orden! Y en nuestra búsqueda aprovechábamos que las palabras estuvieran ordenadas lexicográficamente.

Quedémonos con la idea de que al mirar un elemento podemos decidir si seguir viendo para adelante o para atrás. En el diccionario, quizás podemos asumir que alguna letra vaya a estar "un poco después de la mitad" pero si a priori no conocemos nada sobre el array no lo podemos asumir. El elemento que vamos a mirar para decidir será el del medio. Sin pérdida de generalidad, si no hay exactamente uno en el medio, redondeamos para abajo.

Veamos qué pasa en nuestro contexto. ¿Qué pasa si miro el del medio (en posición "medio") y coincide con la posición? Gané, terminé.

¿Qué pasa si me quedé sin elementos para ver? Terminé, no habrá posiciones que cumplan lo pedido.

¿Qué pasa si miro al del medio y $a[medio] < medio$? Como es estrictamente creciente, todas las posiciones anteriores tendrán valores menores a la posición "medio" en la que estamos pero también a su propio índice. ¿Por qué? Pensemos que como son enteros ordenados de forma estricta, cuanto poco, si nos corremos una posición a la izquierda vamos a disminuir una unidad. ¿Y la posición? ¿Si me muevo a la izquierda? Esa también disminuirá en una unidad. Entonces no vamos a poder disminuir la distancia entre ningún $a[i]$ y su respectivo i, nunca nos vamos a acercar a nuestro objetivo, por lo cual sabremos que, en caso de existir, estará de "medio" en adelante. Podemos asumir que podemos "tirar" hasta la primera posición (o sea, la primer mitad) y quedarnos con la segunda de $\frac{n}{2}$ elementos y buscar allí.

Análogamente, ¿qué pasa si $a[medio] > medio$? Piénsenlo ustedes, es el mismo razonamiento y llegamos a que tendremos que seguir explorando pero esta vez sobre la primera mitad. Los primeros dos casos que vimos serán los casos base del algoritmo recursivo que va fijándose por dónde buscar. ¿Les parece que va a ser mejor que lineal? A priori, en el peor caso, ya no estamos mirando los n elementos.

Les doy un rato para que piensen el algoritmo, ponemos ideas en común y deberíamos llegar a algo así.

```python
def busqueda_indice(arr, izquierda, derecha): # Ojo: Python indexa en 0 y la consigna indexa en 1
    if izquierda > derecha:  # Nos quedamos sin elementos para ver (caso base)
        return False
    medio = (izquierda + derecha) // 2 # Índice medio del subarray actual
    if arr[medio] == (medio + 1) : # Vemos el elemento que queríamos (caso base)
        return True # Podríamos también devolver esta posición
    elif arr[medio] > (medio + 1): # Nos pasamos, buscamos en la primera mitad del array
        return busqueda_indice(arr, izquierda, medio - 1)
    else: # No llegamos, buscamos en la segunda mitad del array
        return busqueda_indice(arr, medio + 1, derecha)

arr = [-4, -1, 2, 4, 7] # Ejemplo
resultado = busqueda_indice(arr, 0, len(arr) - 1)
```

¿Podemos usar el **Teorema maestro**? **(lo mostraría en el proyector, así como cada "paso" de ir descubriendo su aplicación en el problema descripta a continuación)**.

¿De qué tamaño nos van a quedar los llamados recursivos? $\frac{n}{2}$, ya lo habíamos visto hace un rato porque vamos a resolver el mismo problema pero con la mitad del array. ¿Cuántos subproblemas vamos a explorar en cada iteración? 1 solo. Solo nos interesa una de las mitades del array, habíamos visto que la otra la podíamos tirar porque por acá no íbamos a llegar a nuestro objetivo.

¿Y la f(n)? ¿De qué tamaño nos queda? ¿Cuánto nos cuesta un llamado? Tenemos solo comparaciones elementales y accesos a posiciones en O(1). Tampoco se nos suman costos de combinación. Es O(1).

¿Cuál es la cuenta de la complejidad? Hagámosla. **(Las cuentas las haría de a una mostrándolas en el proyector con el Teorema Maestro puesto al lado para que se vea qué corresponde a qué y para qué necesitamos calcular cada una)**.

$$c_{\text{crit}} = \log_2 1 = 0, \quad f(n) = 1, \quad n^{c_{\text{crit}}} = 1, \quad f(n) = n^{c_{\text{crit}}}, \quad k = 0$$

Con la tercer cuentita nos damos cuenta que estamos en el segundo caso del teorema maestro, con k=0 porque la f(n) no está asociada con una complejidad que involucre logaritmos. Luego, vale que:

$$T(n) = \Theta\left(n^{c_{\text{crit}}} \log^{k+1} n\right) = \Theta\left(n^0 \log^1 n\right) = \Theta\left(\log n\right)$$

que es estrictamente mejor que lineal. Ganamos. Resolvimos mediante divide and conquer descomponiendo un problema complejo en subproblemas más simples para llegar a la solución del problema original. Es más eficiente que lineal: mejoramos la complejidad que teníamos antes.

3 Conclusión

Elegí este problema por la analogía inmediata con la vida cotidiana permitiendo, además, que deduzcan fácilmente a partir del ejercicio el algoritmo de búsqueda binaria. Además, al ser un ejemplo sencillo es un buen ejercicio introductorio a la ténica al no asumir que se vieron ejemplos y, además, permite que las personas que faltaron a la teórica (que, lamentablemente, suelen ser bastantantes) no se queden atrás en el rumbo de la clase y logren mantener el hilo.

Asimismo, me parece un buen ejemplo para hacer el paralelismo entre el cálculo de complejidad con árbol de recursión y lo que hace el Teorema Maestro. Agregaría también que es sencillo dibujar el árbol de recursión para visualizarlo gráficamente y se los mostraría.

Añadiría diapositivas que reflejen eso, mostrando en cada slide cómo vamos bajando en el árbol, las hojas que vamos teniendo, en qué nodos dejamos de mirar las ramas siguientes y lograr visualizar los cálculos asociados a la altura y a las hojas y que la cota del Teorema Maestro generaliza la sumatoria a la que se llega con árbol de recursión.