Facultad de Ciencias, UNAM
Análisis de Algoritmos
Tarea 1

Rubí Rojas Tania Michelle

14 de octubre de 2020

1. Sea $M[1 \dots n][1 \dots n]$ una matrix de $n \times n$, en el que cada renglón y cada columna están ordenados en orden creciente. Suponga que no hay dos elementos iguales.

   a) Diseña un algoritmo que encuentre la posición de un valor k en M o que determine si que no está. ¿Cuántas comparaciones usa tu algoritmo en el peor caso?

   b) Describe y analiza un algoritmo para resolver el siguiente problema en tiempo lineal. Dados 4 indices i, j, i', j' como entrada, calcule el número de elementos de M que son más pequeños que $M[i][j]$ y más grandes que $M[i'][j']$.

2. **Permutaciones de Josephus**: Supongamos que n personas están sentadas alrededor de una mesa circular con n sillas, y que tenemos un entero positivo $m \leq n$. Comenzando con la persona con etiqueta 1, (moviéndonos siempre el la dirección de las manecillas del reloj) comenzamos a remover los ocupantes de las sillas como sigue: Primero eliminamos la persona con etiqueta m. Recursivamente, eliminamos al m-ésimo elemento de los elementos restantes. Este proceso continua hasta que las n personas han sido eliminadas. El orden en que las personas han sido eliminadas, se le conoce como la (n, m)-permutación de Josephus. Por ejemplo si $n = 7$ y $m = 3$, la $(7, 3)$-permutación de Josephus es: $\{3, 6, 2, 7, 5, 1, 4\}$.

   a) Supongamos que m es constante. De un algoritmo lineal para generar la (n, m)-permutación de Josephus.

   b) Supongamos que m no es constante. Describa un algoritmo con complejidad $O(n \log n)$ para encontrar la (n, m)-permutación de Josephus.

3. Queremos ordenar una lista S de n enteros que contiene muchos elementos duplicados. Supongamos que los elementos de S sólo toman $O(\log n)$ valores distintos.

   - Encuentre un algoritmo que toma a lo más $O(n \log n)$ tiempo para ordenar S.
   - ¿Por qué esto no viola la cota inferior de $O(n \log n)$ para el problema de ordenación?

4. Dado un arreglo A de n números, queremos contestar la pregunta ¿Hay algún elemento de A que aparezca al menos $\frac{n}{3}$ veces? Encuentre un algoritmo lineal para resolver este problema.

5. Sea $A[1, \dots, n]$ un arreglo de números reales. Diseña un algoritmo que realice cualquier secuencia de las siguientes operaciones:

   - $Add(i, y)$, suma el valor y al i-ésimo número.

- $Partial - sum(i)$, regresa la suma de los primeros i números, es decir, $Partial - sum(i) = A[1] + \ldots + A[i]$

Considera que no hay ni inserciones ni borrado de elementos, sólo se cambia el valor de los números. Cada operación debe tomar $O(\log n)$ pasos. Puedes usar un arreglo de espacio extra de tamaño n.

6. Sea A un arreglo de n números enteros distintos. Suponga que A tiene la siguiente propiedad: existe un indice $1 \leq k \leq n$ tal que $A[1], \ldots, A[k]$ es una secuencia incremental y $A[k+1], \ldots, A[n]$ es una secuencia decremental.

   a) Diseña y analiza un algoritmo eficiente para encontrar k.

   SOLUCIÓN: Tenemos que nuestro arreglo está dividido en dos *subarreglos*, el primero ordenado en forma creciente y el segundo está ordenado en forma decreciente. Supongamos que n es la longitud del arreglo A y que B y C son los subarreglos contenidos en el arreglo A. Por ejemplo, supongamos que A es el siguiente arreglo:

-5	-2	0	2	3	8	6	4	1

   Entonces el arreglo B corresponde a los valores que están de color azul, y el arreglo C corresponde a los valores de color rojo. En este ejemplo, $k = 5$ y $A[k] = 8$.

   Para encontrar el valor de k, lo que debemos hacer es buscar al elemento $A[i]$ tal que

$$A[i-1] < A[i] > A[i+1] \tag{1}$$

   es decir, al elemento en el arreglo cuyos elementos adyacentes son menores que él; y por lo tanto, es el elemento que está al final del subarreglo B. Esto se debe a que los subarreglos B y C están ordenados de forma creciente y decreciente, respectivamente.

   Ahora bien, utilizaremos el método *divide y vencerás* para resolver este problema: Realizamos la operación $\lceil \frac{n}{2} \rceil$ para encontrar el índice del elemento que se encuentra a la mitad del arreglo. Si el elemento $A[\lceil \frac{n}{2} \rceil]$ cumple las condiciones de la expresión 1, entonces hemos encontrado al elemento en el índice k. Terminamos. En caso contrario, lo que hacemos es verificar si el elemento $A[\lceil \frac{n}{2} \rceil]$ pertenece al subarreglo B o al subarreglo C. Si el elemento $A[\lceil \frac{n}{2} \rceil]$ pertenece al subarreglo B, entonces cumple la propiedad

$$A[i-1] < A[i] \tag{2}$$

   es decir, el elemento anterior a $A[\lceil \frac{n}{2} \rceil]$ debería ser menor que él. Pero si el elemento $A[\lceil \frac{n}{2} \rceil]$ pertenece al subarreglo C, entonces debe cumplir la propiedad

$$A[i] > A[i+1] \tag{3}$$

   es decir, el elemento siguiente a $A[\lceil \frac{n}{2} \rceil]$ debería ser menor que él. Luego, si $A[\lceil \frac{n}{2} \rceil]$ pertence al subarreglo B, entonces realizamos este procedimiento recursivamente sobre la mitad derecha del arreglo A (ya que todavía no alcanzamos el final de este subarreglo), pero si $A[\lceil \frac{n}{2} \rceil]$ pertenece al subarreglo C, entonces realizamos este procedimiento recursivamente sobre la mitad izquierda del arreglo A (ya que nos pasamos del final del subarreglo B).

   A esto se debe añadir un caso especial: Si el arreglo es de longitud $n = 1$, entonces simplemente regresamos el índice del primer (y único) elemento de A.

   Durante la ejecución de este algoritmo realizamos operaciones constantes (las comparaciones) y trabajamos con todo el arreglo A sólo para encontrar la mitad del arreglo original, después

vamos trabajando con subarreglos de longitud $\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, 1$; por lo que la complejidad de nuestro algoritmo es de $O(\log n)$. Es decir, realizamos una variación de *búsqueda binaria*. Y funciona porque cada subarreglo sobre el cual vamos trabajando eventualmente nos llevará al elemento que está en el índice k, que es el que estamos buscando.

b) Si no conoces el valor de n, cómo resuelves el problema.

SOLUCIÓN: Como no sabemos cuál es la longitud del arreglo A, entonces necesitamos ver una forma de cómo irnos moviéndo a través del arreglo. Para solucionar esto, los índices que vamos a ir revisando son aquellos que sean potencia de 2, es decir, los índices 2^i tales que $i \in \{0, 1, 2, 3. \ldots\}$. Ahora bien , para poder encontrar el valor de k, primero obtendremos un rango $[a, b]$ de elementos donde poder aplicar el algoritmo del inciso anterior. Para esto, necesitaremos dos contadores a y b, los cuales nos indicarán la posición de inicio y final del intervalo.

El algoritmo que seguiremos para encontrar el intervalo $[a, b]$ será el siguiente: inicializamos nuestros contadores a, b e i en 0. Luego, realizamos la operación

$$2^i = 2^0 = 1$$

e incrementamos nuestro contador i en una unidad. Además, como b nos indicará la posición final del intervalo, entonces

$$b = b + 2^i = 0 + 1 = 1$$

Así, nuestro primer candidato de intervalo será $[a, b] = [0, 1]$. Luego, verificamos que el elemento en la posición $A[2^i]$ cumpla la propiedad 1. Si lo hace, entonces hemos encontrado al elemento cuyo índice es k. En caso contrario, debemos verificar si el elemento $A[2^i]$ pertenece al subarreglo B o al subarreglo C. Esto lo hacemos aplicándole las propiedades 2 y 3, respectivamente. Ahora bien,

- Si el elemento $A[2^i]$ pertenece al subarreglo B, entonces el valor de a será el valor que tenga b en este momento, ya que queremos ir moviéndo el intervalo de tal forma que podamos asegurar que el elemento en el índice k se encuentra dentro de éste. Luego, realizamos la operación 2^i con el valor que tenga i en este momento, lo aumentamos en una unidad y volvemos a actualizar nuestro contador b como

  $$b = b + 2^i$$

  Esto lo hacemos porque aún nos falta camino por recorrer para llegar al final del subarreglo B. Así, nuestro nuevo intervalo candidato es $[a, b]$. Luego, volvemos a la parte de verificación para saber si el elemento $A[b]$ es el que estamos buscando (o para saber qué hacer en casao de que no).
- Si el elemento $A[2^i]$ pertenece al subarreglo C, entonces ya hemos encontrado el intervalo que necesitamos. Esto se debe a que hemos estado moviéndo el intervalo hasta que nos encontramos con un elemento del subarreglo C, y aquí se detiene la búsqueda del intervalo porque ya no es necesario buscar más allá, ya que nuestro contador a estará posicionado en un elemento del subarreglo B (que no es el que buscamos) y b estará posicionado en un elemento del subarreglo C. Entonces, así podemos garantizar que el elemento en el índice k se encuentra en el intervalo $[a, b]$, donde a y b serán los valores que éstos contadores tengan al momento de encontrar al elemento en el subarreglo C, y éstos serán de la forma

  $$a = 2^0 + 2^1 + \ldots \quad \text{y} \quad b = 2^0 + 2^1 + 2^2 + \ldots$$

  Es decir, a tendrá un valor 2^i menor que b. Una vez que tenemos este intervalo, podemos mandar a llamar el algoritmo descrito en el inciso anterior y así obtener el valor k que estamos buscando.

Como nota especial, si al momento de querer acceder al elemento $A[2^i]$ no lo logramos (no existe), entonces sólo hay que volver a iniciar desde cero desde la posición 2^{i-1}; es decir, tendríamos $a = 2^{i-1}$ como el inicio del arreglo y empezamos a buscar las potencias de 2 desde $i = 0$. Así, podemos continuar con el algoritmo descrito arriba.

Este algoritmo funciona porque las potencias de dos nos ayudan a encontrar el intervalo que necesitamos (el inicio es un elemento de B y el final es un elemento de C), y el resto lo termina el algoritmo del inciso anterior; por lo que así obtenemos correctamente nuestro valor k.

Para ilustrar un poco el algoritmo, mostraremos cómo funcionaría con el mismo arreglo A del inciso anterior:

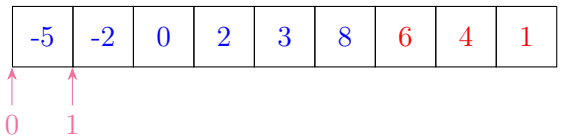

Figura 1: $A[2^0] = -2$ pertenece a B, actualizamos $a = 1$, $b = b + 2^1 = 3$, $i = 2$

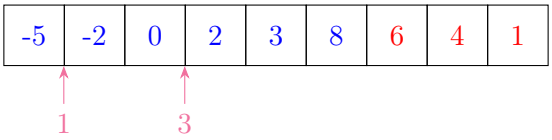

Figura 2: $A[2^i] = 2$ pertenece a B, actualizamos $a = 3$, $b = b + 2^2 = 3 + 4 = 7$, $i = 3$

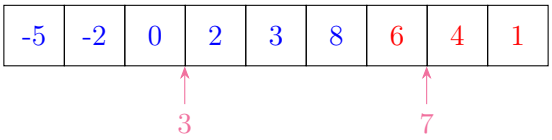

Figura 3: $A[2^i] = 4$ pertenece a C, ya encontramos el intervalo y aplicamos algoritmo del inciso a)

7. You are a young scientist who just got a new job in a large team of 100 people (you the 101- st). A friend of yours who you believe told you that you have more honest colleagues than liars, and that that's all what he can tell you, where a liar is a person who can either lie or tell the truth, while an honest person is one who always tells the truth. Of course, you'd like to know exactly your honest colleagues and the liars, so that you decide to start an investigation, consisting of a series of questions you are going to ask your colleagues. Since you don't wish to look suspicious, you decide to ask only questions of the form "Is Mary an honest person?" and of course, to ask as few questions as possible. Can you sort out all your honest colleagues? What's the minimum number of questions you'd ask in the worst case? You can assume that your colleagues know each other well enough to say if another person is a liar or not. (Hint: Group people in pairs (X,Y) and ask X the question "Is Y honest?" and Y the question "Is X honest?". Analyze all the four possible answers. Once you find an honest person, you can easily find all the others. Challenge: can you solve this enigma asking less than 280 questions in total?)

   Generalize the strategy above and show that given n people such that less than half are liars, you can sort them out in honest persons and liars by asking $\theta(n)$ questions.

8. Suponga que tenemos dos arreglos ordenados $A[1 \dots n]$ y $B[1 \dots n]$ y un entero k. Describe un algoritmo para encontrar el k-ésimo elemento en la unión de A y B. Por ejemplo, si $k = 1$, tu

algoritmo debe regresar al elemento más pequeño de $A \cup B$; si $k = n$, tu algoritmo debe regresar la mediana de $A \cup B$. Puedes suponer que los arreglos no contienen duplicados. Tu algoritmo debe tener complejidad de tiempo $\Theta(\log n)$. Hint: Primero resuelve el caso especial $k = n$.

9. Considera que un río fluye de norte a sur con caudal constante. Suponga que hay n ciudades en ambos lados del río, es decir n ciudades a la izquierda del río y n ciudades a la derecha. Suponga también que dichas ciudades fueron numeradas de 1 a n, pero se desconoce el orden. Construye el mayor número de puentes entre ciudades con el mismo número, tal que dos puentes no se intersecten.