Universidad Nacional Autónoma de México

Facultad de Ciencias

Análisis y Diseño de Algoritmos

Tarea 5

Martínez Avila Santiago

Reynoso Sánchez Arturo Yitzack

26 de noviembre de 2021

1

Peso: 10 puntos. Diseña un algoritmo divide y vencerás que, dado un arreglo de enteros $A = [a_1, a_2, ..., a_n]$ cuyos elementos son todos distintos y están ordenados de menor a mayor, determine si existe o no, un índice i tal que $a_i == i+1$. Tu algoritmo debe tener complejidad $O(\log n)$. Ejemplos de instancias. Entrada: [-1, 1, 3] Salida: Falso. Entrada: [0, 3, 5] Salida: Verdadero.

Tip: Establece y demuestra un lema que te permita, con sólo averiguar si la entrada i-ésima del arreglo tiene i) el valor objetivo (i + 1), o ii) un valor mayor, o iii) un valor menor, concluir que, o bien puedes decidir el valor de retorno del algoritmo, o bien puedes descartar la mitad de los índices candidatos a tener el correspondiente valor objetivo.

Rúbrica por problema

Delimitar explícitamente cada sección.

1. Algoritmo: 3 puntos.
2. Análisis de correción: 3 puntos.
3. Análisis de complejidad: 3 puntos.
4. Claridad en la escritura: 1 punto.

2 Algoritmo

Debido a que el planteamiento del Problema establece que $A = [a_1, a_2, ..., a_n]$ pero los arreglos se indexan desde cero, tenemos que $A[0] = a_1$, $A[1] = a_2$, ..., $A[n-1] = a_n$, es decir, $A[i] = a_{i+1}$ con $i \in [0, n-1]$.
Y como buscamos un índice i tal que $a_i = i + 1$, al indexar desde cero, buscamos un índice i tal que $A[i] = a_{i+1} = (i + 1) + 1 = i + 2$

Algorithm 1 Inicialización

```
procedure INDICE(A)
    int n ← len(A)                                   ▷ O(1)
    if A[0] > 2 or A[n − 1] < n + 1 then             ▷ O(1)
        return False                                 ▷ O(1)
    else:                                            ▷ O(1)
        return INDICEAUX(A, 0, n − 1)                ▷ O(1)
    end if
end procedure
```

Algorithm 2 Auxiliar

```
procedure INDICEAUX(A, low, high)
    int n ← A[low : high].length                                   ▷ O(1)
    if n == 1 and low == high and A[low] ≠ low + 2 then            ▷ O(1)
        return False                                               ▷ O(1)
    end if
    int c ← (low + high)//2                                        ▷ O(1)
    if A[c] == c + 2 then                                          ▷ O(1)
        return True                                                ▷ O(1)
    else if A[c] < c + 2 then                                      ▷ O(1)
        return INDICEAUX(A, c + 1, high)                           ▷ T(n/2)
    else:                                                          ▷ O(1)
        return INDICEAUX(A, low, c − 1)                            ▷ T(n/2)
    end if
end procedure
```

3 Análisis de corrección

Como el arreglo está ordenado de menor a mayor y todos los elementos son distintos, si $k < l$ entonces $a_k < a_j$.

Lema 1: Si $a_i > i + 1$ entonces para toda $j > i$, $a_j > j + 1$.
Sea $a_i > i + 1$. Procederemos por inducción.

- j=i+1
  $a_i < a_{i+1} \implies a_i + 1 \leq a_{i+1}$ pues los $a_l's$ son enteros.
  Por otro lado, $i + 1 < a_i \implies j + 1 = (i + 1) + 1 = i + 2 < a_i + 1 \leq a_{i+1} = a_j$
  $\therefore j + 1 < a_j$
- H.I.
  Supongamos que para toda k, $a_{i+k} > (i + k) + 1$
- P.I.
  $a_{i+k} < a_{i+(k+1)} \implies a_{i+k} + 1 \leq a_{i+(k+1)}$ pues los $a_l's$ son enteros.
  Por otro lado, $(i + k) + 1 = i + k + 1 < a_{i+k} \implies [i + (k + 1)] + 1 = (i + k + 1) + 1 < a_{i+k} + 1 \leq a_{i+(k+1)}$
  $\therefore [i + (k + 1)] + 1 < a_{i+(k+1)}$

Así, para toda $j > i$, $a_i > i + 1$ implica que $a_j > j + 1$.

Lema 2: Si $a_i < i + 1$ entonces para toda $j < i$, $a_j < j + 1$.
Sea $a_i < i + 1$. Procederemos por inducción.

- j=i-1
  Primero notemos que $a_i < i+1 \implies a_i - 1 < i$.
  Además, $a_i > a_{i-1} \implies a_i - 1 \geq a_{i-1}$ pues los $a_l's$ son enteros.
  Por otro lado, $i+1 > a_i \implies j+1 = (i-1)+1 = i > a_i - 1 \geq a_{i-1} = a_j$
  $\therefore j+1 > a_j$

- H.I.
  Supongamos que para toda k, $a_{i-k} < (i-k)+1$

- P.I.
  Primero notemos que $a_{i-k} < (i-k)+1 \implies a_{i-k} - 1 < i-k$.
  Además, $a_{i-k} > a_{i-(k+1)} \implies a_{i-k} - 1 \geq a_{i-(k+1)}$ pues los $a_l's$ son enteros.
  Por otro lado, $(i-k)+1 = i-k+1 > a_{i-k} \implies [i-(k+1)]+1 = i-k > a_{i-k} - 1 \geq a_{i-(k+1)}$
  $\therefore [i-(k+1)]+1 < a_{i-(k+1)}$

Así, para toda $j < i$, $a_i < i+1$ implica que $a_j < j+1$.

Demostración de corrección

Si $A[0] > 2$, es decir, si $a_1 > 2$, por el Lema 1 tenemos que para toda $j > 1$, $a_j > j+1$. Por lo tanto, el algoritmo nos tiene que devolver falso.

Ahora, si $A[n-1] < n+1$, por el Lema 2 tenemos que para toda $j < i$, $a_j < j+1$. Entonces para los índices $\{1, 2, ..., n\}$ eso se cumple. Por lo tanto, el algoritmo nos debe devolver falso.

Si no ocurren los casos anteriores, entonces entramos en el algoritmo recursivo INDICEAUX que busca el índice en el Arreglo A dentro de índices desde $i = low$ hasta $i = high$.

Vamos a demostrar el algoritmo INDICEAUX por inducción sobre el tamaño de $A[low : high]$.

Caso base $n = 1$

En este caso llamamos INDICEAUX con $low = high$.

- Si $a_{low+1} \neq low + 2$, tenemos que $A[low] = a_{low+1} \neq low + 2$ y el algoritmo nos regresa falso.
- Si $a_{low+1} == low + 2$, tenemos que $A[low] = a_{low+1} == low + 2$ y el algoritmo nos regresa verdadero.

Por lo tanto, en el caso base, el algoritmo regresa la salida correctamente.

Hipótesis de inducción

Supongamos que para subarreglos $A[low : high]$con tamaño $n-1$ el algoritmo devuelve correctamente la salida.

Paso inductivo

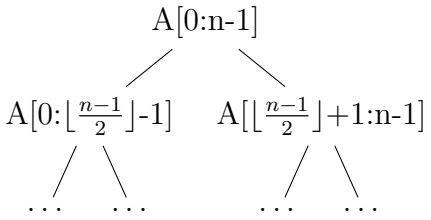
Sea $A[low : high]$ un arreglo de tamaño n y $c = \lfloor \frac{low+high}{2} \rfloor$. Si $A[c] = a_{c+1} = c + 2$, entonces hemos encontrado el índice buscado y nos regresa verdadero. Si no, ocurren dos posibilidades: o $A[c] < c + 2$ o $A[c] > c + 2$.

- Si $A[c] < c+2$, significa que $a_{c+1} < c+2$ y por el Lema 2, para $j < c+1$, $a_j < j+1$. Por lo tanto, el índice que buscamos no es menor a c. Por lo tanto, buscamos recursivamente en $A[c + 1 : high]$, y como es un subarreglo con tamaño menor a n, por la hipótesis de inducción, nos regresa la salida correcta.
- Si $A[c] > c+2$, significa que $a_{c+1} > c+2$ y por el Lema 1, para $j > c+1$, $a_j > j+1$. Por lo tanto, el índice que buscamos no es mayor a c. Por lo tanto, buscamos recursivamente en $A[low : c - 1]$, y como es un subarreglo con tamaño menor a n, por la hipótesis de inducción, nos regresa la salida correcta.

Por lo tanto, el algoritmo auxiliar INDICEAUX es correcto. Por lo tanto, el algoritmo INDICE que llama al algoritmo auxiliar, inicializado en INDICEAUX(A, 0, N-1) es correcto.

4 Análisis de complejidad

Notemos que en INDICE, solamente se declara la variable n y se realiza un *If* en donde se regresa un valor. Lo anterior tiene complejidad *O(1)*. Por lo que nos importa es analizar la complejidad de INDICEAUX.
En este último, cada vez que es llamado, se realiza un *If* que regresa un valor, se declara la variable c y se realiza otro *If* que regresa un valor; todo lo anterior tiene complejidad *O(1)*. Lo relevante ocurre, en el *Else If* y en el *Else*, los cuales vuelven a llamar a INDICEAUX. Analicemos el árbol de recurrencia:



En cada llamada, si el algoritmo no termina, se divide el arreglo en dos partes iguales (o que difieren en 1) hasta que *high=low*, es decir, hasta que el subarreglo tenga tamaño 1. Así dado un arreglo de n elementos, el árbol de recursión tiene profundidad $log_2(n)$. Por lo tanto, como ya se vio que cada llamada tiene complejidad *O(1)*, la complejidad de INDICEAUX y, en consecuencia, la de INDICE es de $O(log(n))$.