Análisis de Algoritmos 2022-1
Tarea 5

Alumnos:
Hernández Sánchez Oscar José
Altamirano Niño Luis Enrique

17 de junio de 2022

Ejercicios

- Propón un algoritmo de complejidad $O(\log n)$ que resuelva el problema.

  Respuesta:

  Para poder definir nuestro algoritmo nos ayudaremos del siguiente lema:

  Lema Dada la entrada i-ésima de un arreglo A cuyas características son las que se describen en el problema, entonces:

  - Si $A[i] = i + 1$ podemos regresar `verdadero`.
  - Si $A[i] > i + 1$, entonces podemos descartar todos los índices $j > i$.
  - Si $A[i] < i + 1$, entonces podemos descartar todos los índices $j < i$.

  Demostración

  Sea $A[i]$ la entrada i-ésima de un arreglo A, tal que $A = [a_1, a_2, \ldots, a_n]$, donde todos los elementos son distintos y están ordenados de menor a mayor, entonces tenemos que mostrar 3 casos:

  - Si $A[i] = i + 1$, entonces es claro que podemos regresar `verdadero`, pues es justo lo que el algoritmo requiere.

  - Si $A[i] > i + 1$, entonces debemos mostrar que podemos descartar todos los índices $j > i$, es decir, debemos mostrar que $\forall j > i(A[j] \neq j + 1)$, entonces:

    Sea $j > i$, supongamos para llegar a una contradicción que $A[j] = j + 1$, y como los elementos del arreglo no se repiten y están ordenados de menor a mayor se tiene que:

    $$\begin{aligned}
    A[i] &< A[j] && \\
    A[i] &< j + 1 && \text{(Sustituimos } A[j] = j + 1) \\
    A[i] - i &< j + 1 - j && \text{(Ya que } j > i = i < j = -i < -j) \\
    A[i] - i &< 1 && \\
    A[i] - i + i &< 1 + i && \\
    A[i] &< i + 1 && \textbf{Contradicción!}
    \end{aligned}$$

    Por lo tanto como llegamos a una contradicción se tiene que $\forall j > i(A[j] \neq j + 1)$.

- Si $A[i] < i + 1$, entonces debemos mostrar que podemos descartar todos los índices $j < i$, es decir, debemos mostrar que $\forall j < i(A[j] \neq j + 1)$, entonces:

  Sea $j < i$, supongamos para llegar a una contradicción que $A[j] = j + 1$, y como los elementos del arreglo no se repiten y están ordenados de menor a mayor se tiene que:

$$\begin{aligned} A[i] &> A[j] \\ A[i] &> j + 1 && \text{(Sustituimos } A[j] = j + 1) \\ A[i] - i &> j + 1 - j && \text{(Ya que } j < i = i > j = -i > -j) \\ A[i] - i &> 1 \\ A[i] - i + i &> 1 + i \\ A[i] &> i + 1 && \textbf{Contradicción!} \end{aligned}$$

  Por lo tanto como llegamos a una contradicción se tiene que $\forall j < i(A[j] \neq j + 1)$.

Por lo tanto el lema es verdadero. ■

Apoyándonos del lema anterior definimos nuestro algoritmo:

Algoritmo 1 Algoritmo recursivo que resuelve el problema requerido.

```
def indice(A,inicio,fin):
    if fin - inicio == 1 or inicio == fin:
        return A[inicio] == inicio + 2 or A[fin] == fin + 2
    else:
        enmedio = inicio + (fin-inicio)//2
        #El +2 es porque los indices comienzan en 1.
        if A[enmedio] == enmedio + 2:
            return True
        if A[enmedio] > enmedio + 2:
            #Busca a la izquierda y descarta los indices de la derecha.
            return indice(A,inicio,enmedio)
        if A[enmedio] < enmedio + 2:
            #Busca a la derecha y descarta los indices de la izquierda.
            return indice(A,enmedio,fin)
```

Algoritmo 2 Se encarga de iniciar el algoritmo recursivo con sus parámetros iniciales.

```
def indiceRec(A):
    return indice(A,0,len(A)-1)
```

- Demuestra su correción.

  Demostración

  Por inducción sobre el tamaño del arreglo.

  Caso Base: Si el tamaño del arreglo es 1. Entonces `inicio=0`, `fin=0`, entonces `inicio=fin` y estamos en el caso base de nuestro algoritmo recursivo, por lo que el algoritmo verificará si el único elemento del arreglo es igual a $1 + 1 = 2$, en tal caso, devolverá `true`, si no es así, entonces devolverá `false`. Lo mismo sucede para el otro caso base, cuando tenemos un arreglo de longitud 2, verificamos si alguno de los dos elementos cumplen la igualdad y regresamos `true` si alguno la cumple o `false` si ninguno la cumple. Por lo que el algoritmo es correcto para los casos base.

  Hipótesis de Inducción: Supongamos que el algoritmo es correcto para un arreglo de tamaño a lo más

k.

Paso Inductivo: Ahora debemos mostrar que el algoritmo es correcto para un arreglo de tamaño $k+1$, entonces el algoritmo obtiene el elemento del medio en el arreglo y se tienen dos casos:

- Si el elemento del medio es mayor que el índice del elemento del medio mas uno, entonces el algoritmo descarta los índices de la derecha, lo cual es correcto por el lema mostrado anteriormente, y se manda a llamar recursivamente con la mitad izquierda del arreglo, además el tamaño de la mitad izquierda será $\lfloor \frac{k+1}{2} \rfloor \leq k$, por lo que por la hipótesis de inducción sabemos que esa llamada recursiva devolverá el resultado correcto.

- Si el elemento del medio es menor que el índice del elemento del medio mas uno, entonces el algoritmo descarta los índices de la izquierda, lo cual es correcto por el lema mostrado anteriormente, y se manda a llamar recursivamente con la mitad derecha del arreglo, además el tamaño de la mitad derecha será $\lceil \frac{k+1}{2} \rceil \leq k$, por lo que por la hipótesis de inducción sabemos que esa llamada recursiva devolverá el resultado correcto.

Por lo tanto el algoritmo es correcto. ■

- Demuestra su complejidad.

Respuesta:

Para un arreglo de tamaño n el árbol de recursión del algoritmo tendrá como raíz el elemento de en medio del arreglo y sus hijos izquierdo y derecho serán los elementos de en medio de las mitades izquierda y derecha del arreglo, el resto del árbol será construida de la misma manera, entonces como el algoritmo busca por mitades hasta llegar a un arreglo de tamaño 2, si y es la altura del árbol, entonces se tiene:

$$\begin{aligned} 2 &= \frac{n}{2^y} \\ 2 \cdot 2^y &= 2^y \frac{n}{2^y} \\ 2 \cdot 2^y &= n \\ 2^{y+1} &= n \\ \log_2 (2^{y+1}) &= \log_2 (n) \\ y + 1 &= \log_2 (n) \\ y &= \log_2 (n) - 1 \end{aligned}$$

por lo que la altura del árbol de recursión será $\log_2 (n) - 1$, entonces llegaremos al caso base después de $\log_2 (n) - 1$ pasos recursivos, por lo que cada trayectoria del árbol tendrá profundidad logaritmica, y como en cada llamada recursiva elegimos buscar solo a la izquierda o a la derecha, entonces tenemos una sola trayectoria del árbol, entonces como el resto de las líneas del algoritmo toman tiempo $O(1)$, entonces la complejidad del algoritmo será $O(\log n)$.