Análisis de Algoritmos
Tarea 3

17 de junio de 2022

Ejercicios

Peso: 10 puntos. Considera el problema búsqueda en arreglo decreciente. Entrada: un arreglo $A = [a_1,]$

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

  Respuesta:

Algoritmo 1 Función que se encarga de mandar a llamar el algoritmo de búsqueda en arreglo decreciente con los parámetros iniciales.

```
    def busquedaDecreciente(a,x):
        return br(a,x,0,len(a)-1)
```

Algoritmo 2 Algoritmo recursivo que resuelve el problema de búsqueda en arreglo decreciente.

```
def br(a,x,inicio,fin):
   if fin-inicio==1 or inicio==fin:
       if a[inicio]<=x:
           return inicio
       elif a[fin]<=x:
           return fin
       else:
           return len(a)
   else:
       enmedio = math.floor(inicio + (fin-inicio)/2)
       if x<a[enmedio]:
           #Busca a la mitad derecha.
           return br(a,x,enmedio,fin)
       else:
           #Busca a la mitad izquierda.
           return br(a,x,inicio,enmedio)
```

- Demuestra su corrección.

  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 menor o igual al elemento buscado, en tal caso, devolverá 0, si no es así, entonces devolverá la

angle θ\ratio	$\sin\theta$	$\cos\theta$	$\tan\theta$	$\csc\theta$	$\sec\theta$	$\cot\theta$
$0°$	0	1	0	undefined	1	undefined
$30°$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$	2	$\frac{2\sqrt{3}}{3}$	$\sqrt{3}$
$45°$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1	$\sqrt{2}$	$\sqrt{2}$	1
$60°$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$	$\frac{2\sqrt{3}}{3}$	2	$\frac{\sqrt{3}}{3}$
$90°$	1	0	undefined	1	undefined	0

longitud del arreglo (1 en este caso). Lo mismo sucede para el otro caso base, cuando tenemos un arreglo de longitud 2.

Hipótesis de Inducción: Supongamos que el algoritmo es correcto para un arreglo de tamaño a lo más k, y un elemento x.

Paso Inductivo: Ahora debemos mostrar que el algoritmo es correcto para un arreglo de tamaño a lo más $k+1$, y un elemento x, entonces el algoritmo obtiene el elemento de en medio del arreglo y dependiendo de si este es mayor que x o no, busca recursivamente en la mitad derecha o izquierda del arreglo, sin importar en que mitad busque, como la longitud de la mitad del arreglo será menor que $k+1$, entonces por la hipótesis de inducción, la llamada recursiva con esa mitad del arreglo es correcta y devuelve el resultado esperado.

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:

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

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 $O(\log n)$.

- Si tu algoritmo anterior fue recursivo, da la solución iterativa del problema.

Respuesta:

Algoritmo 3 Algoritmo iterativo que resuelve el problema de búsqueda en arreglo decreciente.

```
def busquedaIterativa(a,x):
    inicio = 0
    fin = len(a)-1
    while fin-inicio!=1:
        if inicio==fin:
            break
        enmedio = math.floor(inicio + (fin-inicio)/2)
        if x<a[enmedio]:
            #Busca a la mitad derecha.
            inicio = enmedio
        else:
            #Busca a la mitad izquierda.
            fin = enmedio
    if a[inicio]<=x:
        return inicio
    elif a[fin]<=x:
        return fin
    else:
        return len(a)
```