Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

1 de octubre de 2016

1. La iteración del método de Newton es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Dadas nuestras condiciones sobre la función, expandiendo en serie de Taylor alrededor del cero x^* tenemos:

$$\begin{aligned} f(x) &= f(x^*) + \frac{f'(x^*)}{1!}(x - x^*) + \frac{f''(x^*)}{2!}(x - x^*)^2 + O((x - x^*)^3) \\ &= \frac{f''(x^*)}{2}(x - x^*)^2 + O((x - x^*)^3) \\ f'(x) &= f'(x^*) + \frac{f''(x^*)}{1!}(x - x^*) + O((x - x^*)^2) \end{aligned}$$

Substituyendo, llamando $e_n = x_n - x^*$ al error y absorbiendo algunas constantes en los $O(\cdot)$ queda:

$$\begin{aligned} e_{n+1} &= e_n - \frac{f''(x^*)e_n^2 + O(e_n^3)}{2f''(x^*)e_n + O(e_n^2)} \\ &= e_n\left(1 - \frac{1}{2}\right) + O(e_n^2) \end{aligned}$$

O sea, en este caso la convergencia es lineal.

Puntajes

Total		20
Iteración de Newton	5	
Aproximar $f(x)$, $f'(x)$ vía Taylor	10	
Orden de convergencia es lineal	5	

2. La idea es que el mayor subárbol completo que tiene la raíz en un nodo dado está formado por los máximos subárboles completos con raíz en sus hijos, podados al mínimo de sus dos alturas, junto a este nodo. Mientras calculamos éstos en un recorrido en postorden, registramos en variables globales los máximos hallados hasta el momento. Resulta natural asignar "altura" -1 a los punteros nulos, da altura 0 para las hojas en forma automáticamente.

   Esto da el algoritmo siguiente:

   ```
   maxtree ← nil
   maxheight ← −1
   procedure subtree (root, height)

   if root = nil then
       height ← −1
   else
       subtree(root−>left, hl)
       subtree(root−>right, hr)
       height ← 1 + min(hl, hr)
       if height > maxheight then
           maxheight ← height
           maxtree ← root
       end
   end
   ```

Puntajes

Total		20
Idea general (máximo árbol en cada nodo, combinar)	5	
Caso base	2	
Registrar máximo en variables globales	3	
Algoritmo	10	

3. Lo sugerido en la pregunta corresponde a:

   function F *(*n*)*

   if n ≤ 1 **then**
       return *1*
   else
       return F *(*n- *1)* + F *(*n- *2)*
   end

   Llamemos t_n al número de llamadas a F al invocar $F(n)$. Se ve que si $n = 0$ o $n = 1$ se hace una llamada a F, si $n \geq 2$ se hacen $t_{n-1} + t_{n-2}$ llamadas recursivas. O sea, tenemos la recurrencia:

   $$t_{n+2} = t_{n+1} + t_n \qquad t_0 = t_1 = 1$$

   Para resolver esta recurrencia, recurrimos a funciones generatrices:

   $$T(z) = \sum_{n \geq 0} t_n z^n$$

   Aplicando las propiedades de funciones generatrices ordinarias:

   $$\frac{T(z) - t_0 - t_1 z}{z^2} = \frac{T(z) - t_0}{z} + T(z) + \frac{1}{1-z}$$

   Substituyendo los valores iniciales y despejando tenemos:

   $$\begin{aligned} T(z) &= \frac{1 - z + z^2}{(1-z)(1-z-z^2)} \\ &= \frac{2}{1-z-z^2} - \frac{1}{1-z} \end{aligned}$$

   Sabiendo que la función generatriz de los números de Fibonacci es:

   $$F(z) = \frac{z}{1-z-z^2}$$

   al ser $F_0 = 0$ resulta la función generatriz de los F_{n+1}:

   $$\frac{F(z) - F_0}{z} = \frac{1}{1-z-z^2}$$

   lo que nos permite concluir:

   $$t_n = 2F_{n+1} - 1$$

Puntajes

Total		20
Plantear la función recursiva	4	
Número de llamadas en cada caso	3	
Recurrencia para t_n	6	
Condiciones iniciales	4	
Explicar cómo seguir	3	

4. Aplicando directamente la idea de backtracking resulta:

```
free ← {1,2,...,n}
function derangements (k)

if k = n then
    for i ← 1 to n do
        print (d [i ])
    end
else
    for i ∈ free do
        free ← free ∖ {i}
        d[k] ← i
        derangements(k + 1)
        free ← free ∪ {i}
    end
end
```

Se llama originalmente con argumento 1.

Puntajes

Total		20
Idea de backtracking	3	
Caso base	5	
Case general	12	

5. Paso a paso.

   a) Considerando la subsecuencia creciente más larga que llega hasta $a[i]$, hay dos casos según este último:

   Es menor que todos los anteriores: En este caso, la secuencia es únicamente este elemento.

   Es mayor que algunos de los anteriores: En este caso, es la secuencia más larga que termina en un elemento anterior menor a $a[i]$ junto con $a[i]$.

   Esto da la recurrencia:

$$\operatorname{lis}(i) = \begin{cases} 1 & \text{si } a[i] \leq a[j] \text{ para } 1 \leq j \leq i-1 \\ 1 + \max_{\substack{1 \leq j \leq i-1 \\ a[j] < a[i]}} \{\operatorname{lis}(j)\} \end{cases}$$

   Además de el valor de $\operatorname{lis}(i)$ debemos registrar el valor de j que da lugar al máximo para poder reconstruir la secuencia.

   b) Las suposiciones son:

   Casos exhaustivos: Consideramos todas las alternativas para el último elemento: Siempre lo incluimos, aún si es en una secuencia de largo uno.

   Subestructura inductiva: Agregar un nuevo elemento no puede interferir con las secuencias crecientes que terminan en cada uno de los elementos existentes.

   Subestructura óptima: Si la secuencia creciente más larga incluye a $a[n]$, la secuencia creciente que termina en el elemento anterior en ella claramente debe ser de largo máximo.

   c) La idea es llenar un arreglo con el largo de la secuencia más larga que llega a $a[i]$, y paralelamente registrar el valor de j que da lugar a ese máximo. Es claro que la recurrencia de 5*a* que podemos ir llenando este arreglo ordenadamente desde $i = 1$ hasta $i = n$, solo haremos uso de valores calculados anteriormente.

   Conviene ir registrando el máximo hallado hasta el momento durante el proceso de llenado, este es LIS de la secuencia.

   d) Para obtener la secuencia de lo registrado, buscamos el elemento del arreglo definido en el punto 5*c* el índice que da el máximo, y seguimos los índices j que referencian las secuencias previas.

   e) Recorrer el arreglo para hallar j según lo esbozado por la recurrencia de 5*a* tiene costo $O(i)$, como esto se repite para cada uno de los n elementos el total de esta fase es $O(n^2)$. Construir la secuencia creciente máxima de los datos registrados claramente es $O(n)$. En total, el tiempo de ejecución es $O(n^2)$.

Puntajes

Total				35
a)	Recurrencia		5	
b)	Supuestos		10	
	Casos exhaustivos	3		
	Subestructura inductiva	3		
	Subestructura óptima	4		
c)	Esbozo de algoritmo		10	
d)	Extraer la subsecuencia		5	
e)	Complejidad		5	

HvB/$\LaTeX\,2_\varepsilon$