Pauta de Corrección

Primer Certamen
Algoritmos y Complejidad

5 de mayo de 2018

1. Por turno.

   a) Un punto fijo es la solución de la ecuación:

   $$\begin{aligned} x &= 2x - ax^2 \\ 0 &= x - ax^2 \\ &= x(1 - ax) \end{aligned}$$

   Uno de los puntos fijos es $x = 0$, el otro es $x = 1/a$.

   b) Sabemos que la iteración $x_{n+1} = g(x_n)$ converge a x^* si $|g'(x^*)| < 1$. En nuestro caso, $g(x) = 2x - ax^2$, con $g'(x) = 2 - 2ax$. Veamos los dos puntos fijos:

   $\boldsymbol{x^* = 0}$**:** Acá $g'(x^*) = 2$, nunca converge a él.

   $\boldsymbol{x^* = 1/a}$**:** Tenemos:

   $$\begin{aligned} g'(x^*) &= 2 - 2a \cdot \frac{1}{a} \\ &= 0 \end{aligned}$$

   La convergencia es superlineal.
   Nuestra expansión estándar es:

   $$\begin{aligned} g(x_{n+1}) &= g(x^*) + g'(x^*)(x_n - x^*) + \frac{1}{2} g''(x^*)(x_n - x^*)^2 \\ &\quad + O((x_n - x^*)^3) \\ x^* + e_{n+1} &= g(x^*) + g'(x^*)e_n + \frac{1}{2} g''(x^*)e_n^2 + O(e_n^3) \\ e_{n+1} &= -ae_n^2 + O(e_n^3) \end{aligned}$$

   La convergencia es cuadrática.

   c) Para que converja desde un x_0 dado, debe ser que $|x_1 - x^*| < |x_0 - x^*|$.

Veamos:

$$\begin{aligned}
|x_0 - x^*| &= \left| x_0 - \frac{1}{a} \right| \\
&= \frac{1}{a} \cdot |ax_0 - 1| \\
|x_1 - x^*| &= \left| 2x_0 - ax_0^2 - \frac{1}{a} \right| \\
&= \frac{1}{a} \cdot |2ax_0 - a^2 x_0^2 - 1| \\
&= \frac{1}{a} \cdot |a^2 x_0^2 - 2ax_0 + 1| \\
&= \frac{1}{a} \cdot (ax_0 - 1)^2
\end{aligned}$$

Una manera de comparar es mediante la razón entre los dos:

$$\begin{aligned}
\left| \frac{x_1 - x^*}{x_0 - x^*} \right| &= \frac{(ax_0 - 1)^2}{|ax_0 - 1|} \\
&= |ax_0 - 1|
\end{aligned}$$

Si ésto es menor a 1, converge desde x_0, o sea, como a es positivo:

$$\begin{aligned}
-1 < ax_0 - 1 &< 1 \\
0 < ax_0 &< 2 \\
0 < x_0 &< \frac{2}{a}
\end{aligned}$$

Para todo x_0 en este rango converge.

Puntajes

Total	25
a) Puntos fijos	8
b) Convergencia a 0 y $1/a$	8
c) Puntos iniciales x_0	9

2. Sabemos que la convergencia de iteración de punto fijo está dada por:

$$e_{n+1} = g'(x^*)e_n + O(e_n^2)$$

Por tanto, si $g'(x^*) = 0$, tenemos convergencia superlineal. En nuestro caso, eso significa:

$$g(x) = x + \alpha f(x)$$
$$g'(x^*) = 1 + \alpha f'(x^*)$$

De lo anterior:

$$\alpha = -\frac{1}{f'(x^*)}$$

Si aplicamos el mismo desarrollo que usamos al ver la convergencia del método de Newton, vemos que la convergencia es cuadrática. Claro que rara vez tenemos idea de cuánto vale $f'(x^*)$...

Puntajes

Total		15
Convergencia de FPI	5	
Anular coeficiente de e_n	8	
Valor de α	2	

3. Estamos pidiendo que la fórmula sea exacta para polinomios de grado hasta n. En particular, es exacta para los polinomios ℓ_k, de grado n. Pero estos son tal que:

   $$\ell_k(x_j) = [j = k]$$

   Para ellos la fórmula de cuadratura propuesta se reduce a:

   $$\begin{aligned}\int_{x_0}^{x_n} \ell_k(x)\,\mathrm{d}x &= \sum_{0\le j\le n} A_j \ell_k(x_j) \\ &= A_k\end{aligned}$$

   como se pedía.

   Es claro que los polinomios son linealmente independientes, por lo que podemos representar todo Π_n como combinaciones lineales de ellos.

Puntajes

Total	25
Valores de $\ell_k(x_j)$	5
Aplicar la fórmula propuesta a los ℓ_k	7
Resultado para A_k	8
Funciona con todos los polinomios por ser los ℓ_k base de Π_n	5

4. Una idea es ordenar las tareas en orden de ganancia decreciente, y programar ejecutar la tarea i justo antes de su plazo fatal si estamos a tiempo para cumplirlo. De esta forma la tarea programada interfiere lo menos posible con las demás. Si la programación resultante tiene tiempos muertos, podemos compactar al final adelantando tareas.

   Nuestro problema general es entonces que tenemos una secuencia de ranuras de tiempo, que pueden estar libres u ocupadas. Hay una secuencia de tareas, ordenadas en orden de ganancia decreciente; si hay empate en ganancias las ordenamos por plazo fatal decreciente.

   Elejimos la primera tarea de la secuencia, y la programamos en la ranura libre más tardía antes de su plazo fatal, si la hay. En caso contrario la descartamos.

   Nuestras tres propiedades son:

   Greedy Choice: Nuestro algoritmo elige la tarea $\hat{t}$ que más ganancia da de entre las que aún pueden cumplir su plazo fatal. Demostramos por contradicción que hay una solución óptima que incluye la tarea $\hat{t}$ así elegida.

   Tomemos una solución óptima. Es claro que las ranuras antes del plazo fatal de $\hat{t}$ están todas ocupadas, ya que en caso contrario podemos programar $\hat{t}$ en una libre, contradiciendo que la solución es óptima.

   Si la solución óptima incluye a $\hat{t}$, estamos listos. Si no la incluye, podemos tomar una tarea que termina antes del plazo fatal de $\hat{t}$. Si su ganancia es menor que la ganancia de $\hat{t}$, intercambiándola con $\hat{t}$ mejoramos la ganancia total, contradicción con que la solución sea óptima. La única posibilidad es que tenga la misma ganancia de $\hat{t}$, podemos intercambiarlas obteniendo una solución óptima que incluye a $\hat{t}$.

   Inductive Substructure: Sea P el problema original, $\hat{t}$ la tarea elegida por el criterio voraz, y el problema P' lo que queda al asignar $\hat{t}$ a la última ranura libre antes de su plazo fatal. Una solución viable a P', o sea, una colección de tareas a programar entre las restantes, nunca puede entrar en conflicto con la programación de $\hat{t}$; podemos combinar una solución a P' con $\hat{t}$ para dar una solución viable a P.

   En realidad, en este caso no hay restricciones "cruzadas", esto se cumple automáticamente al eliminar tareas que ya no pueden cumplir sus plazos fatales.

   Optimal Substructure: Consideremos una solución óptima Π^* al problema P. Para simplificar notación, llamemos $|\Pi|$ a la ganancia de la solución viable Π al problema P, y similarmente $|t|$ la ganancia que reporta la tarea t.

   Por **Greedy Choice**, podemos suponer sin pérdida de generalidad que Π^* incluye la elección voraz $\hat{t}$. Sea P' el problema que queda al eliminar $\hat{t}$ y las tareas que ya no pueden completarse. Sea Π' una solución óptima para P', por **Inductive Substructure** es compatible con $\hat{t}$; la ganancia de esa solución es:

   $$|\Pi'| \leq |\Pi^*| - |\hat{t}|$$

(si fuera mayor, junto con $\hat{t}$ daría una solución mejor que la óptima). Pero la solución $\Pi^* \setminus \{\hat{t}\}$ da el valor $|\Pi^*| - |\hat{t}|$, y la combinación Π' con $\hat{t}$ es óptima.

Como se cumplen las tres propiedades, el algoritmo da una solución óptima.

Puntajes

Total		30
Propuesta de algoritmo voraz	6	
Propiedad **Greedy Choice**	8	
Propiedad **Inductive Substructure**	8	
Propiedad **Optimal Substructure**	8	

5. El algoritmo recursivo obvio es el 1. Si llamamos T_n al número de llamadas a la

Algoritmo 1: Algoritmo recursivo para calcular a_n

```
function a(n)
    if n ≤ 1 then
        return 1
    else
        return (3 a(n − 1) + 2 a(n − 2)) mód 42
    end
```

función hechas para calcular a_n (el tiempo total está acotado por un múltiplo de esto), vemos que cumple:

$$T_{n+2} = T_{n+1} + T_n \qquad T_0 = T_1 = 1$$

Pero esta es la recurrencia de Fibonacci, en realidad es:

$$T_n = F_{n+1}$$

Sabemos que los números de Fibonacci crecen exponencialmente.

Alternativamente, una recurrencia de la forma:

$$U_{n+2} = \alpha U_{n+1} + \beta U_n \qquad U_0, U_1 \text{ dados}$$

tiene una solución de la forma:

$$U_n = c_1 r_1^n + c_2 r_2^n$$

donde r_1, r_2 son las raíces de la ecuación:

$$r^2 = \alpha r + \beta$$

y c_1, c_2 dependen de los valores iniciales. Si $|r_1| \geq |r_2|$, y $|r_1| > 1$, solo por una coincidencia extraordinaria es $c_1 = 0$, y:

$$|U_n| \sim |c_1 r_1^n|$$

O sea, crecimiento exponencial. En este caso la ecuación es:

$$r^2 = r + 1$$

con ceros:

$$\begin{aligned} r_{1,2} &= \frac{1 \pm \sqrt{1^2 + 4 \cdot 1 \cdot 1}}{2} \\ &= \frac{1 \pm \sqrt{5}}{2} \end{aligned}$$

Algoritmo 2: Programación dinámica para calcular a_n

```
function a(n)
    if n ≤ 1 then
        return 1
    else
        a_0 ← 1
        a_1 ← 1
        for i ← 0 to n − 1 do
            t ← (3 · a_1 + 2 · a_0) mód 42
            a_0 ← a_1
            a_1 ← t
        end
        return a_1
    end
```

Es claro que:

$$\frac{1+\sqrt{5}}{2} > 1$$

Una forma eficiente de calcular los valores es registrar los últimos dos (no se requieren los anteriores), el algoritmo 2 claramente es lineal en n.

Puntajes

Total	35
Algoritmo obvio	5
Demostración que es exponencial	10
Propuesta eficiente	20

HvB/LaTeX2ε