Pauta de Corrección

Certamen Recuperativo
Algoritmos y Complejidad

5 de diciembre de 2016

1. Por turno:

   a) Si converge a algún valor, es a un punto fijo:

   $$x^* = \frac{1}{2}\left(x^* + \frac{a}{x^*}\right)$$
   $$x^* = \sqrt{a}$$

   b) Sea:

   $$g(x) = \frac{1}{2}\left(x + \frac{a}{x}\right)$$

   Por el teorema de *contraction mapping*, podemos aplicar el criterio de la derivada de g y esto converge en todo rango $[u, v]$ tal que $|g'(x)| < 1$. Vemos que:

   $$g'(x) = \frac{1}{2}\left(1 - \frac{a}{x^2}\right)$$

   Es claro que $g'(x) < 1$ para todo $x > 0$. Interesa ver dónde $g'(x) = -1$:

   $$\frac{1}{2}\left(1 - \frac{a}{x^2}\right) > -1$$
   $$x > \sqrt{\frac{a}{3}}$$

   En consecuencia, converge al menos para:

   $$\sqrt{\frac{a}{3}} < x_0$$

Puntajes

Total		25
a)	Determinar punto fijo	10
b)	Condiciones de convergencia	15

2. **Algoritmos voraces** son aplicables cuando nos enfrentamos a una secuencia de decisiones, y podemos tomar la decisión correcta usando solo información "local". Un ejemplo es el algoritmo de Prim para obtener el árbol recubridor mínimo: se agregan arcos en orden de costo creciente siempre que no formen ciclos con los ya agregados.

Programación dinámica es aplicable cuando debemos elegir entre diversas opciones, y una exploración recursiva directa al elegir una de las opciones divide el problema en subproblemas, muchos de los cuales se repiten. Una aplicación se dió en la tarea de determinar el cambio óptimo.

Backtracking también es aplicable para explorar alternativas, aplicable en situaciones en las que los subproblemas a resolver no se repiten. Lo vimos en el caso de las n reinas, o para resolver Sudoku.

Puntajes

Total	25
Algoritmos voraces	8
Programación dinámica	9
Backtracking	8

3. Por turno.

   a) Nos interesa OPT(n) (en realidad, las ubicaciones de los restaurantes de la cadena). Valor base obvio es OPT(0) = 0 (si no hay posiciones posibles para restaurantes, la ganancia es cero).

   b) Si la distancia entre x_{k-1} y x_k es mayor a d, simplemente incluya un restaurante en k; en caso contrario, vea la mejor entre no incluir k (simplemente el para $k-1$) o dejar fuera los "demasiado cercanos" anteriores (esto es considerar solo $1,\ldots,i$ tal que i esté lejos) e incluir éste. Es natural crear un arreglo OPT que contenga los valores. La recurrencia es:

   $$\text{OPT}(k) = \text{máx}\{\text{OPT}(k-1), g_k + \text{OPT}(i)\}$$

   donde i es tal que $x_i < x_k + d < x_{i+1}$.

   c) Nos conviene mantener el valor actual de i en vez de recalcularlo.

```
i ← 1
OPT[0] ← 0
OPT[1] ← g_1
for k ← 2 to n do
    while x_k − d > x_{i+1} do
        i ← i + 1
    end
    if OPT[k − 1] ≥ g_k + OPT[i] then
        OPT[k] ← OPT[k − 1]
    end
    OPT[k] ← g_k + OPT[i]
end
```

Algoritmo 1: Algoritmo para *Dish at Step*

Puntajes

Total	25
Caso base y resultado buscado	5
Recurrencia para OPT	10
Algoritmo	10

4. El costo al intercalar dos listas es simplemente proporcional a la suma de sus largos (el largo de la lista resultante). Así:

   a) El costo es:

$$\begin{aligned} 2n + 3n + \cdots + kn &= \left(\frac{k(k+1)}{2} - 1 \right) n \\ &= \Theta(k^2 n) \end{aligned}$$

   b) Si usamos la idea de dividir en dos grupos lo más iguales posibles de listas, intercalamos éstas recursivamente e intercalamos los resultados, el tiempo total (en unidades de n) está dado por:

$$T(k) = 2T(k/2) + nk$$

   Es aplicable el teorema maestro con $a = b = 2$, $d = 1$:

$$T(k) = \Theta(nk \log k)$$

Puntajes

Total		30
a)	Costo total	10
b)	Planteo dividir y conquistar, solución	15

5. Un polinomio no cero de grado n sobre $\mathbb{F}_p$ tiene a lo más n ceros. Si r y s coiniciden en x, quiere decir que x es un cero de $r(x) - s(x)$, un polinomio de grado a lo más n. Como $\mathbb{F}_p$ tiene p elementos, la probabilidad que un x elegido uniformemente al azar en $\mathbb{F}_p$ sea un cero de $r(x) - s(x)$ es a lo más $1/p$.

Puntajes

Total	20
Idea del algoritmo	10
Cota al error	10

HvB/LaTeX2ε