Pauta de Corrección

Segundo Certamen
Algoritmos y Complejidad

23 de diciembre de 2017

1. Llamemos $S(n)$ al tiempo que toma con un arreglo de largo n. Del programa vemos que las tres llamadas son sobre arreglos de $2/3$ del largo, además de trabajo básicamente constante. Esto da la recurrencia:

   $$S(n) = 3S(2n/3) + \Theta(1)$$

   Para el teorema maestro, tenemos $a = 3$, $b = 3/2$, $f(n) = \Theta(1)$. El valor crítico es $\log_{3/2} 3 > 1$ (el valor es $2,7095$), tenemos $f(n) = O(n)$, es el primer caso, y:

   $$\begin{aligned} S(n) &= \Theta(n^{\log_{3/2} 3}) \\ &= \Theta(n^{2,7095}) \end{aligned}$$

   Esto es aún mucho peor que Bubblesort...

Puntajes

Total	25
Plantear la recursión para tiempo de ejecución	8
Aplicar teorema maestro	12
Conclusión de orden de tiempo de ejecución	5

2. Llamemos $A(n)$, $B(n)$ y $C(n)$ los tiempos de ejecución respectivos. Del problema extraemos las recurrencias.

Algoritmo A: Ordenar n elementos tiene costo $\Theta(n \log n)$. Recurrencia es $A(n) = 3A(n/3) + \Theta(n \log n)$. Es aplicable el teorema maestro, $a = b = 3$, $f(n) = \Theta(n \log n)$. El valor crítico es $\log_3 3 = 1$, tenemos que $f(n) = \Theta(n \log n)$, es el caso 4 del teorema maestro con $\alpha = 1$, resulta:

$$A(n) = \Theta(n \log^2 n)$$

Algoritmo B: Recurrencia es $B(n) = 2B(n-1) + \Theta(1)$. Definamos la función generatriz:

$$b(z) = \sum_{n \geq 0} B(n) z^n$$

Aplicando las propiedades a la recurrencia, aproximando $f(n) = c$ queda:

$$\frac{b(z) - B(0)}{z} = 2b(z) + \frac{c}{1-z}$$

Despejando:

$$b(z) = \frac{B(0)}{1-2z} + \frac{c}{(1-z)(1-2z)}$$

Por fracciones parciales sabemos que esto puede escribirse:

$$b(z) = \frac{\alpha}{1-2z} + \frac{\beta}{1-z}$$

Sabemos que de acá:

$$\begin{aligned} B(n) &= [z^n] b(z) \\ &= \alpha \cdot 2^n + \beta \\ &= \Theta(2^n) \end{aligned}$$

Algoritmo C: La recurrencia es $C(n) = 9C(n/4) + \Theta(n^2)$, Para el teorema maestro, $a = 9$, $b = 4$, $f(n) = \Theta(n^2)$. El valor crítico es $\log_4 9 < 2$, estamos en el caso 5 si se cumple la condición adicional con $c = 2$. Verificamos ésta:

$$9 \cdot \left(\frac{n}{4}\right)^2 = \frac{9}{16} n^2$$

O sea, se cumple por ejemplo con $k = 5/8 < 1$. Concluimos que:

$$C(n) = \Theta(n^2)$$

Para n grande, el mejor es el algoritmo A.

Puntajes

Total	35
Algoritmo A	10
Algoritmo B	10
Algoritmo C	10
Conclusión	5

3. Si el costo de n operaciones es $O(n \log n)$, el costo amortizado por operación es:

$$\frac{O(n \log n)}{n} = O(\log n)$$

El costo máximo de una operación es $O(n \log n)$ (por ejemplo, si las otras son todas $O(1)$). El costo por operación al menos es constante, el mínimo es $O(1)$.

Al análisis amortizado es aplicable cuando lo relevante es el costo total de una secuencia de operaciones. Si lo que importa es el costo de una operación individual (como en sistemas de tiempo real o sistemas interactivos), debe considerarse el costo máximo.

Puntajes

Total		25
Costo amortizado	7	
Costo peor caso de una operación	8	
Costo mejor caso de una operación	5	
Aplicabilidad	5	

4. La idea es tener arreglos $e[n]$ para el valor de la máxima suma alternante de largo par y $o[n]$ para la máxima suma alternante de largo impar de $a[0,\ldots,i]$. Valores iniciales son $e[0] = 0$ (no hay sumas alternantes de largo par de un solo elemento) y $o[0] = a[0]$ (la suma alternante de largo impar de un solo elemento).

   Enseguida, la máxima suma alternante de largo par hasta i es la máxima suma alternante de largo impar hasta $i-1$ o la máxima suma alternante de largo par hasta $i-1$, y similarmente para la máxima suma impar. Vale decir:

$$e[i+1] = \begin{cases} o[i] - a[i+1] & o[i] - a[i] > e[i] \\ e[i] & \text{caso contrario} \end{cases}$$

$$o[i+1] = \begin{cases} e[i] + a[i+1] & e[i] + a[i] > o[i] \\ o[i] & \text{caso contrario} \end{cases}$$

   Interesa el máximo de $e[n-1]$ y $o[n-1]$. Calculamos $e[i]$ y $o[i]$ desde $i = 0$ (los valores iniciales están dados arriba) hasta $i = n-1$.

   Es claro que el algoritmo planteado es $\Theta(n)$, y que es imposible mejorar esto (al menos se requiere examinar cada uno de los n elementos de a).

   En realidad, no necesitamos todos los valores, bastan los penúltimos para calcular el valor actual. Vea el programa C++ del listado 1 para detalles. En un ciclo lee una línea de valores, y luego lee los valores al arreglo a. Para cada arreglo leído calcula el máximo pedido.

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include <vector>

using namespace std;

int main()
{
    string input;

    while (getline(cin, input)) {
        istringstream line(input);

        vector<int> a;
        int value;

        while (line >> value)
            a.push_back(value);

        int e = 0, o = a[0];
        for (auto it = begin(a) + 1; it != end(a); ++it) {
            int tmp_e = e, tmp_o = o;
            if (e + *it > o)
                tmp_o = e + *it;
            if (o - *it > e)
                tmp_e = o - *it;
            e = tmp_e;
            o = tmp_o;
        }
        cout << (e > o ? e : o) << endl;
    }
}
```

Listado 1: Maximal Alternating Subset Sum

Puntajes

Total		40
Definir subproblemas (entrelazados)	4	
Recurrencias, valores iniciales	10	
Objetivo general	4	
Orden de operaciones	5	
Algoritmo	12	
Algoritmo es $\Theta(n)$	5	

HvB/LaTeX2ε