Pauta de Corrección

Segundo Certamen
Algoritmos y Complejidad

18 de agosto de 2018

1. Registramos el número de repeticiones de a[k] en counts[k]. El programa es directo, considera para k dado el número máximo de veces que puede usarse a[k], y va descontando repeticiones hasta llegar a cero, y se llama recursivamente con la suma remanente para el siguiente valor de k. El listado 1 da el programa completo (lee la suma a conseguir y el arreglo a de una línea, y escribe las distintas opciones de obtener la suma).

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

using namespace std;

void comb_sums(int, int);

vector<int> a, counts;

int main()
{
    string input;

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

        int sum, value;

        line >> sum; a.clear();
        while (line >> value)
            a.push_back(value);
        counts.resize(a.size());
        for(auto it = counts.begin(); it != counts.end(); it++)
                *it = 0;
        cout << "Sum = " << sum << endl;
        cout << "a:";
```

```
        for(auto v : a)
                cout << " " << v;
        cout << endl << endl;

        comb_sums(sum, 0);

        cout << endl << endl;


    }
}

void comb_sums(int sum, int k)
{
    if(sum == 0) {
        cout << "Counts:";
        for(auto cnt : counts)
            cout << " " << cnt;
        cout << endl;
    }
    else if(k < a.size()) {
        for(int i = sum / a[k]; i; i--) {
            counts[k] = i;
            comb_sums(sum - i * a[k], k + 1);
        }
    }
}
```

Listado 1: Formas de conseguir la suma

Puntajes

Total	25
Estructura recursiva general	10
Cálculo de las cuentas	10
Condición de fin	5

2. Llamemos $A(n)$, $B(n)$ y $C(n)$ los tiempos de ejecución respectivos. Del problema extraemos las recurrencias. En la notación del problema maestro al reverso del enunciado:

Algoritmo A: La recurrencia da $a = 8$, $b = 2$, $f(n) = O(n^3)$. Tenemos $\alpha = \log_2 8 = 3$, por el cuarto caso del teorema maestro (tenemos $\beta = 0$):

$$A(n) = \Theta(n^3 \log n)$$

Algoritmo B: Vemos $a = 10$, $b = 3$, $f(n) = \Theta(n^2 \log n)$. Es $\alpha = \log_3 10 > 2$, es el quinto caso del teorema si se cumple la condición sobre f:

$$\begin{aligned} 10f(n/3) &= 10\left(\frac{n}{3}\right)^2 \log\left(\frac{n}{3}\right) \\ &= \frac{10}{9} n^2 \log n - \frac{10 \log 3}{9} n^2 \end{aligned}$$

Esto es menor a $10/9 f(n)$. O sea:

$$\begin{aligned} B(n) &= \Theta(f(n)) \\ &= \Theta(n^2 \log n) \end{aligned}$$

Algoritmo C: Para el teorema maestro, $a = 9$, $b = 3$, $f(n) = O(n^2 \log n)$. El valor crítico es $\log_3 9 = 2$, estamos en el cuarto caso con $\beta = 1$, resulta:

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

Para n grande, el mejor es el algoritmo C.

Puntajes

Total	15
Algoritmo A	4
Algoritmo B	4
Algoritmo C	4
Conclusión	3

3. El costo total de la secuencia de m operaciones, si no se ordena, es:

$$m \cdot \left(0,6 \cdot \frac{n}{2} + 0,4n\right) = 0,7mn$$

Si se ordena el costo total es:

$$2(n+1)\ln n + m \cdot 2\ln n = 2m\ln n + 2(n+1)\ln n$$

Estas son ecuaciones lineales en m, su intersección define el punto en que deja de ser ventajoso búsqueda lineal:

$$0,7m^* n = 2m^* \ln n + 2(n+1)\ln n$$
$$m^* = \frac{2(n+1)\ln n}{0,7n - 2\ln n}$$

Un gráfico (figura 1) muestra que m^* es bastante pequeño, nunca vale la pena la búsqueda lineal en este caso.

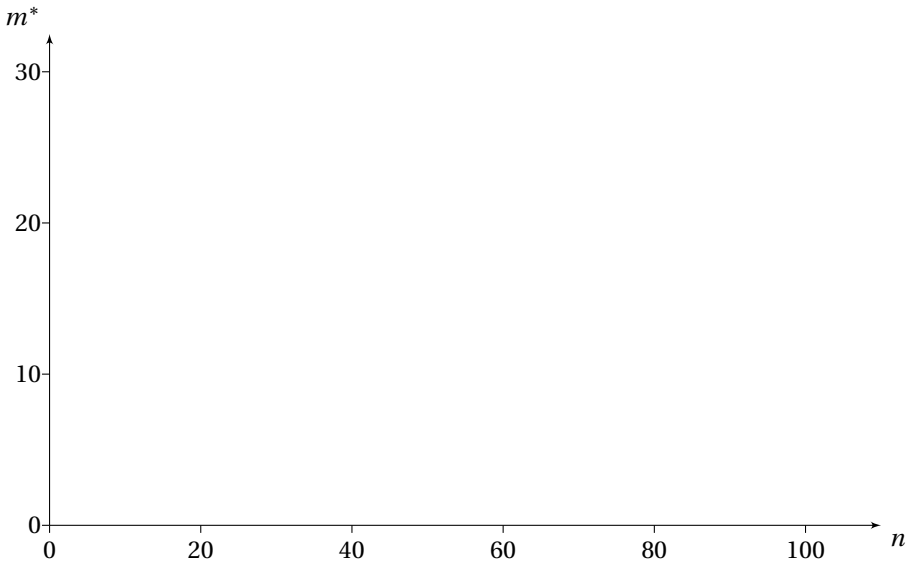

Figura 1: Gráfica de m^* contra n

Puntajes

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

4. Por turno.

   a) El algoritmo obvio es el 1. Si $T(n)$ es el tiempo requerido para computar

Algoritmo 1: Cálculo de a_n (algoritmo ingenuo)

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

   a_n por la recurrencia obvia, para constantes c (el costo de los cálulos y la llamada a función) y d (el costo de la llamada base):

$$T(n) = T(n-1) + T(n-2) + c, T(0) = T(1) = d$$

   Esto está acotado por abajo por la recurrencia de Fibonacci (vemos que $T(n) \geq dF_n$ de la recurrencia), cuya solución sabemos crece exponencialmente.
   Podemos resolver usando funciones generatrices. Definamos:

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

   con lo que de la recurrencia:

$$\frac{g(z) - dz - d}{z^2} = \frac{g(z) - d}{z} + g(z) + \frac{c}{1-z}$$

$$g(z) = \frac{d - dz + cz^2}{1 - 2z + z^3}$$

$$= \frac{c+d}{1-z-z^2} - \frac{c}{1-z}$$

   Vemos que la solución está dada por:

$$T(n) = (c+d)F_{n+1} - c$$

   que crece exponencialmente, como comentado.

   b) Una manera eficiente de calcular los valores es por programación dinámica: ir llenando un arreglo con los valores. Incluso, como la recurrencia involucra únicamente los últimos dos valores, bastan éstos. El tiempo claramente es $O(n)$. El algoritmo 2 da detalles.

Puntajes

Total	25
a) Planteo de la recurrencia, cota exponencial	15
b) Cálculo por programación dinámica	10

Algoritmo 2: Cálculo de a_n (programación dinámica)

```
function a(n)
    if n ≤ 1 then
        return 1
    end
    u, v ← 1, 1
    for i ← 2 to n do
        u, v ← v, (3v + 2u) mód 42
    end
    return v
end
```

5. El largo promedio de las colas no depende de la función de *hashing*, siempre es simplemente el número de objetos (palabras) dividido por el número de casilleros (tamaño de la tabla). El largo de la cola más larga es revelante, determina el peor caso de búsqueda en la tabla. En el peor caso (todos los objetos dan el mismo valor) es simplemente el número de objetos, en el mejor (distribución completamente uniforme) será el entero superior al largo promedio de las colas.

Puntajes

Total		15
Largo promedio no depende de la función	7	
Relevancia del largo máximo	8	