Algoritmos y Estructuras de Datos – 2º Ing. Informática – Segundo Parcial

Respuestas del examen. 19 de junio de 2.006

1. (3 puntos)

Vamos a hacer el apartado a), que es la forma más adecuada de resolver el problema. Si nos fijamos un poco en las ecuaciones, vemos que todas son del tipo:

$$t(n)= a\, t(n/2) + n^p$$

Podemos aplicar el método de la ecuación característica, con el cambio de variable $n = 2^k$. Nos queda:

$$t'(k) = a\, t'(k-1) + 2^{pk}$$

Cuya ecuación característica es:

$$(x - a)(x - 2^p) = 0 \Rightarrow x= (a, 2^p)$$

Debemos distinguir dos casos: $a=2^p$, $a\neq 2^p$.

$$\text{Si } a=2^p \Rightarrow t'(k) = c_1\cdot 2^{pk} + c_2\cdot k\cdot 2^{pk} \Rightarrow t(n) = c_1\cdot n^p + c_2\cdot \log_2 n\cdot n^p \in O(\log n\cdot n^p)$$

$$\text{Si } a\neq 2^p \Rightarrow t'(k) = c_1\cdot 2^{pk} + c_2\cdot a^k \Rightarrow t(n) = c_1\cdot n^p + c_2\cdot a^{\log_2 n} = c_1\cdot n^p + c_2\cdot n^{\log_2 a} \in O(\max(n^p, n^{\log_2 a}))$$

$$\text{Luego, si } a<2^p \Rightarrow t(n) \in O(n^p)\text{; y si } a>2^p \Rightarrow t(n) \in O(n^{\log_2 a})$$

Aplicando el resultado a los ejemplos tenemos:

1. $a=1$; $b= 0$; $t(n) \in O(\log n)$; Ejemplo: búsqueda de un elemento en un árbol AVL.
2. $a=2$; $b= 0$; $t(n) \in O(n)$; Ejemplo: recorrido en inorden de un árbol AVL.
3. $a=2$; $b=1$; $t(n) \in O(n\cdot \log n)$; Ejemplo: algoritmo de ordenación por mezcla.
4. $a=3$; $b=1$; $t(n) \in O(n^{\log_2 3})$; Ejemplo: multiplicación rápida de enteros largos, de Karatsuba y Ofman.
5. $a=4$; $b=1$; $t(n) \in O(n^2)$; Ejemplo: multiplicación de enteros largos con divide y vencerás, método sencillo.
6. $a=7$; $b=2$; $t(n) \in O(n^{\log_2 7})$; Ejemplo: multiplicación rápida de matrices de Strassen.
7. $a=8$; $b=2$; $t(n) \in O(n^3)$; Ejemplo: multiplicación de matrices con divide y vencerás, método sencillo.

2. (2 puntos)

Por un lado, está claro que si todos los valores de las casillas fueran positivos la solución óptima sería sacar siempre 1 en el dado. Si hay algún valor negativo, deberíamos saltar hasta el primer número positivo que aparezca. La única dificultad del problema es decidir qué ocurre cuando hay 6 o más negativos consecutivos. Podemos usar varias estrategias: saltar al que reste menos (es decir, al mayor de los 6), o saltar siempre un 6 en el dado (para pasar rápido). Pero ninguno de estos criterios garantiza la solución óptima.

La implementación es sencilla. Supongamos que el resultado se almacena en el array Tiradas [1..n].

```
operación OcaVoraz (T: array [1..n] de entero, var puntuación: entero; var Tiradas: array [1..n] de entero)
    actual:= 1
    puntuación:= T[1]
    numTiradas:= 0
    mientras actual < n hacer
        imax:= 1;
        para i:= 1, ..., 6 hacer
            si (actual+i > n) O (T[actual] ≥ 0) entonces
                imax:= i
                break //salir del para
            sino si T[actual+i] ≥ T[imax] entonces
                imax:= i
            finsi
        finpara
        numTiradas:= numTiradas+1
        Tiradas[numTiradas]:= imax
        actual:= actual + imax
        puntuación:= puntuación + T[actual]
    finmientras
```

Observar que en caso de empate a valores negativos, nos vamos al de mayor valor. Pero esto tampoco garantiza la solución óptima. Por ejemplo, suponer el caso: **T**= (200,-100,-101,-102,-103,-104,-105,-106,-107,-108,-109,-110). Claramente, la solución óptima es (6, 6), con puntuación 95. Sin embargo, nuestro algoritmo devolvería (1, 1, 1, 1, 1, 1, 6), con puntuación -415.

3. (2,5 puntos)

Si planteamos el problema como una toma de decisiones, la decisión que tenemos analizar es cada una de las 6 posibles tiradas del dado. La solución óptima será la que dé mayor valor. Si definimos la función **Tiradas(k): entero**, como la puntuación óptima de llegar a la casilla **k**, la definición recurrente sería:

$$\text{Tiradas}(k) = T[k] + \max \{\text{Tiradas}(k-1), \text{Tiradas}(k-2), \ldots \text{Tiradas}(k-6)\} = T[k] + \max_{i=1..6} \{\text{Tiradas}(k-i)\}$$

Los casos base serían:

$$\text{Tiradas}(1) = T[1];\ \text{Tiradas}(<0) = -\infty$$

En este problema, la tabla del algoritmo es unidimensional. Podemos definir V: array [1..n] de entero, con V[i] = Tiradas(i), según la fórmula anterior. La implementación del algoritmo es trivial. La reconstrucción de la solución se haría empezando en la posición de la tabla V[n], y analizando qué valor dio el máximo. Esto nos da la última tirada en la solución óptima. Guardaríamos esa tirada y nos moveríamos a esa casilla. Y así hasta llegar a la casilla 1. La implementación es sencilla y se deja también como ejercicio.

4. (2,5 puntos)

El tipo nodo debe almacenar la tupla solución, la posición actual, la puntuación actual y los valores de la cotas. Igual que en avance rápido, la tupla solución sería un array s= (s_1, s_2, ..., s_m), con m≤n, y cada s_i indica la tirada i-ésima del dado. La representación del nodo podría ser:

```
tipo nodo = registro
        tupla: array [1..n] de entero
        numTiradas: entero
        actual: entero
        puntuación: entero
        CI, CS, BE: entero
finregistro
```

Para la cota inferior se puede usar el algoritmo voraz del ejercicio 2, sin más que cambiar la inicialización de actual, puntuación y numTiradas, por las del nodo correspondiente. La cota superior podría ser la puntuación actual más la suma de las casillas positivas desde s.actual+1 hasta n. Y como beneficio estimado se puede tomar la media de ambas cotas.

El nodo raíz sería:

```
raíz.tupla:= (0, 0, ..., 0)
raíz.numTiradas:= 0
raíz.actual:= 1
raíz.puntuación:= T[1]
raíz.CI:= OcaVoraz2 (T, raíz)
raíz.CS:= Σi=1..n, con T[i]>0 T[i]
raíz.BE:= (raíz.CI + raíz.CS)/2
```

Y la generación de los descendientes de un nodo **x**:

```
para i:= 1, ..., 6 hacer
        si i + x.actual > n entonces
                break // salir del para
        finsi
        y.tupla:= x.tupla
        y.numTiradas:= x.numTiradas + 1
        y.tupla[y.numTiradas]:= i
        y.actual:= x.actual + i
        y.puntuación:= x.puntuación + T[y.actual]
        y.CI:= OcaVoraz2 (T, y)
        y.CS:= y.puntuación + Σi=y.actual..n, con T[i]>0 T[i]
        y.BE:= (y.CI + y.CS)/2
finpara
```

La función solución, para un nodo **x**, es simplemente comprobar si **x.actual** ≥ **n**. Evidentemente, el esquema necesario sería el de maximización. La variable de poda, **C**, tomaría el valor de la mayor CI o la mejor solución encontrada. Podamos un nodo **x** si **x.CS** ≤ C. Para la estrategia de ramificación podríamos tomar una MB-LIFO (mayor beneficio estimado, y en caso de empate seguir por el último nodo generado).