Soluciones de ejercicios estudiados en clase

Algoritmos y Programación II 75.04

Lucio Santi <lsanti at dc.uba.ar>
Facultad de Ingeniería - UBA

1. Enunciado

Dar un algoritmo que permita reconstruir un árbol binario a partir de sus recorridos preorder e inorder. Asumir que el árbol no contiene elementos repetidos.

2. Resolución

Sean $P[1 \dots n], I[1 \dots n]$ los arreglos que contienen respectivamente los recorridos preorder e inorder de un árbol binario arbitrario t. Observemos, en primera instancia, que, por definición de recorrido preorder, el primer elemento de P será la raíz de t. Llamemos r a este elemento. Dado que por hipótesis podemos asumir que en t no hay valores repetidos, r debe aparecer exactamente una vez en I. Sea entonces k la posición de r en I (i.e., $I[k] = r$). Por definición de recorrido inorder, sabemos ahora que todo elemento a la izquierda de r en I debe pertenecer al subárbol izquierdo de t, t_i, y que todo elemento a su derecha debe ubicarse en el subárbol derecho, t_d. Puesto de otra manera, esto significa que $I[1 \dots k-1]$ y $I[k+1 \dots n]$ representan los recorridos inorder de t_i y t_d.

En cuanto a P, el contenido del subárbol izquierdo aparecerá allí en forma contigua a r. De lo anterior podemos afirmar que este subárbol posee $k-1$ elementos, con lo cual se concluye que $P[2 \dots k]$ es precisamente el recorrido preorder de t_i. El contenido del resto del arreglo pertenece, entonces, a t_d. Es decir, $P[k+1 \dots n]$ es el recorrido preorder de este subárbol.

De este análisis podemos derivar naturalmente un algoritmo recursivo para tratar el problema presentado. Ilustramos los lineamientos del mismo en la figura 1.

Notar que, para proveer una formulación correcta, resta identificar el caso base de esta estrategia. El escenario más simple que puede darse ocurre cuando el árbol a reconstruir es vacío, de manera que tanto P como I también lo serán (i.e., $n = 0$).

2.1. Complejidad temporal

Veamos ahora qué tan bueno es el desempeño de nuestro algoritmo. Analizaremos, a este propósito, las respectivas complejidades temporales de peor y mejor caso.

2.1.1. Peor caso

Sea $T : \mathbb{N} \to \mathbb{N}$ la función que representa el tiempo de ejecución de peor caso de ReconstruirÁrbol. En lo que sigue, vamos a buscar una cota superior asintótica para T.

En el pseudocódigo de la figura 1 se observa que el tiempo de ejecución del algoritmo está dominado por un ciclo y por dos llamadas recursivas. Dado que el árbol binario subyacente a los arreglos es arbitrario, no está claro de antemano cuántas iteraciones realizará el ciclo ni tampoco lo están los tamaños exactos de cada invocación recursiva. Imaginemos por un momento una situación en la

ReconstruirÁrbol($P[1 \dots n]$, $I[1 \dots n]$):
Si $n = 0$: devolver un árbol binario vacío.
Sea $r := P[1]$.
Sea $k := 1$.
Mientras $I[k] \neq r$: $k := k + 1$.
Sea $t_i :=$ ReconstruirÁrbol($P[2 \dots k], I[1 \dots k-1]$).
Sea $t_d :=$ ReconstruirÁrbol($P[k+1 \dots n], I[k+1 \dots n]$).
Devolver un árbol binario t tal que:
- La raíz de t sea r,
- El subárbol izquierdo de t sea t_i, y
- El subárbol derecho de t sea t_d.

Figura 1: Pseudocódigo del algoritmo para reconstruir el árbol

que el ciclo maximice su cantidad de iteraciones. Esto es, dados P e I de tamaño n, el ciclo correrá exactamente n veces. Sea $M : \mathbb{N} \to \mathbb{N}$ la recurrencia que caracteriza el tiempo de ejecución de este escenario hipotético. Tenemos que

$$M(n) = M(k_1) + M(k_2) + n$$

con $k_1, k_2 \in \mathbb{N}$ tales que $k_1 + k_2 = n - 1$. Estos valores de k_1 y k_2 hacen referencia a los respectivos tamaños de los subárboles izquierdo y derecho subyacentes.

Es claro que $T(n) \in \mathrm{O}(M(n))$. Además, puede verse que $M(n) \in \mathrm{O}(n^2)$. Para ello, demostremos la siguiente propiedad.

Propiedad 1. $\exists\, c_0 \in \mathbb{N}_{>0} \,/\, M(n) \leq c_0 n^2 + 1 \;\forall\, n \in \mathbb{N}$

Demostración Utilizando el método de sustitución.
Si $n = 0$ y asumimos que $M(0) = 1$, el resultado es trivialmente cierto para cualquier elección de c_0.

Sea $n > 0$ y supongamos que $M(k) \leq c_0 k^2 + 1$ para cualquier natural $k < n$. Entonces,

$$\begin{aligned}
M(n) &= M(k_1) + M(k_2) + n \\
&\overset{\text{HI}}{\leq} (c_0 k_1^2 + 1) + (c_0 k_2^2 + 1) + n \\
&= c_0(k_1^2 + k_2^2) + n + 2 \\
&\overset{k_1, k_2 \geq 0}{\leq} c_0(k_1 + k_2)^2 + n + 2 \\
&= c_0(n-1)^2 + n + 2 \\
&= c_0(n^2 - 2n + 1) + n + 2 \\
&= c_0 n^2 + (1 - 2c_0)n + 2 + c_0 \qquad (1)
\end{aligned}$$

Podemos ver que

$$(1) \leq c_0 n^2 + 1 \Leftrightarrow 1 + c_0 \leq (2c_0 - 1)n$$

Si tomamos $c_0 \geq 2$, esta última desigualdad será cierta para cualquier natural $n \geq 1$.

De todo esto se concluye que es suficiente tomar, por ejemplo, $c_0 = 2$ para satisfacer la desigualdad deseada.

□

Como corolario, obtenemos por la transitividad de $O(\cdot)$ que $T(n) \in O(n^2)$. Sin embargo, aún no podemos afirmar que la peor ejecución del algoritmo insume tiempo cuadrático: nos falta identificar una familia de árboles binarios cuya reconstrucción ponga de manifiesto este comportamiento.

Puede comprobarse sin grandes dificultades que la familia de árboles degenerados a izquierda constituye, en efecto, el peor caso de este algoritmo. Para el n-ésimo miembro de esta familia, ilustrado en la figura 2, el tiempo de ejecución de ReconstruirÁrbol viene dado por la recurrencia

$$T_w(n) = T_w(n-1) + \Theta(n)$$

cuya solución asintótica es, claramente, $\Theta(n^2)$.

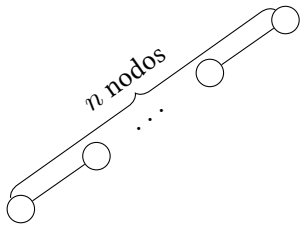

Figura 2: n-ésimo miembro de la familia de árboles degenerados a izquierda

Este escenario evidencia un costo asintótico que, como vimos, acota superiormente al tiempo de ejecución de nuestro algoritmo. En conclusión, la complejidad temporal de peor caso de ReconstruirÁrbol es $O(n^2)$.

2.1.2. Mejor caso

Siguiendo un razonamiento similar al de la sección previa, la familia de árboles binarios degenerados a derecha exhibe un comportamiento óptimo para este algoritmo. Esto puede observarse considerando que el tiempo de ejecución para el n-ésimo miembro de la familia queda caracterizado por la recurrencia

$$T_b(n) = T_b(n-1) + \Theta(1)$$

cuya solución asintótica es $\Theta(n)$.

Observemos, también, que un costo lineal es asintóticamente inmejorable dado que el algoritmo visita al menos una vez cada nodo del árbol. Esto permite concluir que, efectivamente, la complejidad temporal de mejor caso de ReconstruirÁrbol es $\Omega(n)$.