EDyAII - Análisis de costo para `tabulate`

y otras trampas

28 de mayo de 2020

Vamos a analizar dos implementaciones de `tabulate` para el tipo `BTree`.

```
data BTree a = Empty | Node Int (BTree a) a (BTree a)
-- Invariante: el entero guarda la longitud de la secuencia

-- Algunas funciones auxiliares
len :: BTree a -> Int
len Empty = 0
len (Node s _ _ _) = s

node :: BTree a -> a -> BTree a -> BTree a
node l x r = Node (len l + len r + 1) l x r
```

La primera "desliza" f usando composición para llamarse recursivamente

```
tabulate1 :: (Int -> a) -> Int -> BTree a
tabulate1 f 0 = Empty
tabulate1 f n =
    let s = div (n-1) 2
        (l, (x, r)) = tabulate1 f s ||| (f s ||| tabulate1 (f . (+(s+1))) (n-s-1))
     in node l x r
```

La segunda versión usa una función auxiliar para llevar explícitamente una "franja" en la cual tiene que llamar a f.

```
tabulate2 :: (Int -> a) -> Int -> BTree a
tabulate2 f n = tab 0 (n-1) where
  tab min max
    | min > max = Empty
    | otherwise = let mid = div (min + max) 2 in
                      (l, (x, r)) = tab min (mid-1) ||| (f mid ||| tab f (mid+1) max)
                  in node l x r
```

Las dos funciones cumplen la especificación (y de hecho devuelven exactamente el mismo resultado). Sin embargo, tienen una diferencia fundamental en el trabajo que realizan.

1. Primera versión

Empecemos analizando la primera versión. Consideramos el trabajo de cada división, suma, resta, y llamadas a `node` y `len` constantes. Podemos plantear el trabajo de la primera versión como:

$$
\begin{array}{lcll}
W(\texttt{tabulate}_1\ f\ 0) & = & O(1) & \\
W(\texttt{tabulate}_1\ f\ n) & = & O(1) & \text{(factores constantes de suma, div, etc)} \\
 & & + W(\texttt{tabulate}_1\ f\ s) & \text{(llamada izquierda)} \\
 & & + W(f\ s) & \text{(llamada a } f\text{)} \\
 & & + W(\texttt{tabulate}_1\ (f \circ (+(s+1)))\ (n-s-1)) & \text{(llamada derecha)}
\end{array}
$$

(El segundo caso sólo para $n \neq 0$. Arriba, s es $\lfloor \frac{n}{2} \rfloor$. Hacemos la simplificación usual de redondear y eliminar los pisos y techos, no hacen diferencia.)

La función $f' = f \circ (+(s+1))$ sólo hace un factor constante de trabajo antes de llamar a f, es decir:

$$W(f'\ n) = W(f\ (n+s+1)) + O(1) \tag{1}$$

esto significa, por ejemplo, que si f es constante, entonces f' también lo es. Dado esto, uno estaría tentado de ignorar este costo extra, y acotar la recurrencia con:

$$W(\texttt{tabulate}_1\ f\ n) \in O\left(\sum_{i=0}^{n-1} W(f\ i)\right)$$

"demostrándolo" por inducción así:

$$
\begin{aligned}
& W(\texttt{tabulate}_1\ f\ n) \\
= \quad & \langle \text{def. } W \rangle \\
& O(1) + W(\texttt{tabulate}_1\ f\ s) \\
& \qquad + W(f\ s) \\
& \qquad + W(\texttt{tabulate}_1\ (f \circ (+(s+1)))\ (n-s-1)) \\
= \quad & \langle \text{HI} \rangle \\
& O(1) + O\left(\sum_{i=0}^{s-1} W(f\ i)\right) \\
& \qquad + W(f\ s) \\
& \qquad + O\left(\sum_{i=0}^{(n-s-1)-1} W((f \circ (+(s+1)))\ i)\right) \\
= \quad & \langle \text{La composición sólo agrega un costo constante, podemos ignorarlo} \rangle \\
& O(1) + O\left(\sum_{i=0}^{s-1} W(f\ i)\right) \\
& \qquad + W(f\ s) \\
& \qquad + O\left(\sum_{i=0}^{(n-s-1)-1} W(f\ (i+(s+1)))\right) \\
= \quad & \langle \text{El } O(1) + W(f\ s) \text{ está acotado por } O(W(f\ s)) \rangle \\
& O\left(\sum_{i=0}^{s-1} W(f\ i)\right) + O(W(f\ s)) + O\left(\sum_{i=0}^{(n-s-1)-1} W(f\ (i+(s+1)))\right) \\
= \quad & \langle \text{Cambio de índices en sumatoria} \rangle \\
& O\left(\sum_{i=0}^{s-1} W(f\ i)\right) + O(W(f\ s)) + O\left(\sum_{i=s+1}^{n-1} W(f\ i)\right) \\
= \quad & \langle \text{Agrupar sumatorias} \rangle \\
& O\left(\sum_{i=0}^{n-1} W(f\ i)\right)
\end{aligned}
$$

Sin embargo, hay un error: en la tercera igualdad ignoramos un factor de $O(1)$ *para cada sumando de la segunda sumatoria*. Es decir, en realidad ignoramos un costo proporcional a $O(n-s-1) \approx O(\frac{n}{2}) \approx O(n)$. Haciendo los pasos más explícitamente, hicimos:

$$
\begin{aligned}
& O\left(\sum_{i=0}^{(n-s-1)-1} W((f \circ (+(s+1)))\ i)\right) \\
= \quad & \langle \text{Ecuación 1} \rangle \\
& O\left(\sum_{i=0}^{(n-s-1)-1} (W(f\ (i+(s+1))) + O(1))\right) \\
= \quad & \langle \text{Ignoramos } O(1) \rangle \\
& O\left(\sum_{i=0}^{(n-s-1)-1} W(f\ (i+(s+1)))\right)
\end{aligned}
$$

Pero no es correcto ignorar ese $O(1)$, porque esos costos, cada uno menor a una constante dada, se acumulan $n-s-1$ veces. Más correcto sería hacer:

$$
\begin{aligned}
& O\left(\sum_{i=0}^{(n-s-1)-1} W((f \circ (+(s+1)))\ i)\right) \\
= \quad & \langle \text{Ecuación 1} \rangle \\
& O\left(\sum_{i=0}^{(n-s-1)-1} (W(f\ (i+(s+1))) + O(1))\right) \\
= \quad & \langle \text{Distribuir} \rangle \\
& O\left(\sum_{i=0}^{(n-s-1)-1} W(f\ (i+(s+1)))\right) + O\left(\sum_{i=0}^{(n-s-1)-1} O(1)\right) \\
= \quad & \langle O(1) \times (n-s-1) = O(n-s-1) \rangle \\
& O\left(\sum_{i=0}^{(n-s-1)-1} W(f\ (i+(s+1)))\right) + O(n-s-1) \\
= \quad & \left\langle n-s-1 \text{ es } O(n) \text{ (recordar que } s \approx \frac{n}{2}) \right\rangle \\
& O\left(\sum_{i=0}^{(n-s-1)-1} W(f\ (i+(s+1)))\right) + O(n)
\end{aligned}
$$

Nota al margen: por estas cosas decimos que sumar y operar con notación O es un "abuso de notación", porque se presta a cometer estos errores sutiles. Si uno hiciera las pruebas con costos concretos y la definición de O, estos errores son evidentes, y difíciles de cometer.

Por el teorema maestro, este factor de $O(n)$ en cada llamada recursiva termina agregando un trabajo total en $O(n \lg n)$. La recurrencia se termina resolviendo a

$$W(\texttt{tabulate}_1\ f\ z) = O(n \lg n) + O\left(\sum_{i=0}^{n-1} W(f\ i)\right)$$

En el caso particular de usar $\texttt{tabulate}_1$ con una función de costo constante, tenemos un costo asintóticamente mayor al deseado: $O(n \lg n) > O(n)$.

Algunas otras formas de ver por qué las composiciones cuestan tanto

Pensando en árboles completos para simplificar (i.e. $n = 2^k - 1$ para algún k). (Esta parte sería más fácil si uno piensa en árboles que tiene valores en las hojas en vez de los nodos... puede ayudar tener eso en mente.)

— En cada punto de la recursión, cada llamada a f' del lado derecho va a costar una constante k extra por la composición y la suma, y tenemos $\frac{n}{2}$ de ellas, con lo cual vamos a tener un costo extra de $k \times \frac{n}{2} \in O(n)$.

— Pensando en el costo total, hay $\frac{n}{2}$ llamadas a f que ocurren en profundidad máxima $(\lg n)$. De ellas, al menos la mitad fue *las mismas o más* veces a la derecha que a la izquierda[1]. Entonces, estas $\frac{n}{4}$ llamadas hicieron al menos $\frac{\lg n}{2}$ trabajo cada una, lo cuál está en $\Omega(n \lg n)$.

— De nuevo mirando el costo total, cada vez que la función desciende por la rama derecha, agrega una composición más a f. Al ir k veces por la rama derecha, tenemos algo cómo

$$f \circ (+s_1) \circ (+s_2) \circ ... \circ (+s_k)$$

agregando un costo en $O(k)$ a la llamada. En total, tenemos n llamadas a las distintas f, y en promedio, cada llamada "fue" por la rama derecha aproximadamente $\frac{\lg n - 1}{2}$ veces (pista: la profundidad *promedio* en un árbol completo de profundidad n tiende a $n - 1$, y cada nodo tiene un "espejo" desde la raíz), y acumuló esa cantidad de composiciones. Entonces, tenemos un costo extra en $O(n * \frac{\lg n}{2}) = O(n \lg n)$.

— Pensemos el caso en que f es constante, y cuesta 1 unidad de trabajo evaluarla. Y supongamos que cada composición agrega otra unidad. Podemos plantear el trabajo de `tabulate f n` como $W(1, n)$, con:

$$\begin{aligned} W(k, 0) &= 0 \\ W(k, n) &= W(k, s) + k + W(k + 1, n - s - 1) \quad \text{(dónde } \texttt{s = div n 2}\text{)} \end{aligned}$$

donde k lleva el costo de llamar a la f compuesta. Demuestre que $W(k + 1, n) = W(k, n) + n$, y que $W(1, n) \in \Theta(n \lg n)$.

[1] Para convencerse, notar que cada "hoja" tiene su espejo que siguió el camino opuesto. O, pensar en los números binarios de n bits: ¿cuántos tienen más 1s que 0s?

2. Segunda versión

La segunda versión no tiene este problema. Podemos plantear el trabajo de `tab` como:

$$\begin{array}{lcll} W(\texttt{tab}\ f\ m\ M) & = & O(1) & (\text{cuándo } m > M) \\ W(\texttt{tab}\ f\ m\ M) & = & O(1) & (\text{factores constantes de suma, div, etc}) \\ & & + W(\texttt{tab}\ f\ m\ (s-1)) & (\text{llamada izquierda}) \\ & & + W(f\ s) & (\text{llamada a } f) \\ & & + W(\texttt{tab}\ f\ (s+1)\ M) & (\text{llamada derecha}) \end{array}$$

Acá, f no varía dentro de la recurrencia, y sí podemos demostrar por inducción que

$$W(\texttt{tab}\ f\ m\ M) \in O\left(\sum_{i=m}^{M} W(f\ i)\right)$$

La prueba es la que uno espera, y sale directa por inducción. Aquí hay un bosquejo del paso inductivo.

Queremos demostrar que $W(\texttt{tab}\ f\ m\ M) \in O(\sum_{i=m}^{M} W(f\ i))$, buscamos c y n_0 para satisfacer que

$$W(\texttt{tab}\ f\ m\ M) \leq c(\sum_{i=m}^{M} W(f\ i))$$

Prueba:

$$\begin{array}{ll} & W(\texttt{tab}\ f\ m\ M) \\ = & \langle \text{def. } W \rangle \\ & k_0 + W(\texttt{tab}\ f\ m\ (s-1)) \\ & \quad + W(f\ s) \\ & \quad + W(\texttt{tab}\ f\ (s+1)\ M) \\ \leq & \langle \text{HI} \rangle \\ & k_0 + c\sum_{i=0}^{s-1} W(f\ i) \\ & \quad + W(f\ s) \\ & \quad + c\sum_{i=s+1}^{M} W(f\ i) \\ \leq & \langle \text{Tomando } c \geq k_0 + 1 \rangle \\ & c\sum_{i=m}^{s-1} W(f\ i) + cW(f\ s) + c\sum_{i=s+1}^{M} W(f\ i) \\ = & \langle \text{Juntando las sumatorias} \rangle \\ & c\sum_{i=m}^{M} W(f\ i) \end{array}$$

O, siendo un poco más laxos y usando notación O (correctamente!)

$$
\begin{aligned}
& W(\texttt{tab}\ f\ m\ M) \\
= \quad & \langle \text{def. } W \rangle \\
& O(1) + W(\texttt{tab}\ f\ m\ (s-1)) \\
& \qquad + W(f\ s) \\
& \qquad + W(\texttt{tab}\ f\ (s+1)\ M) \\
= \quad & \langle \text{HI} \rangle \\
& O(1) + O\left(\sum_{i=0}^{s-1} W(f\ i)\right) \\
& \qquad + W(f\ s) \\
& \qquad + O\left(\sum_{i=s+1}^{M} W(f\ i)\right) \\
= \quad & \langle\, O(1) \text{ está acotado por } O(W(f\ s)) \,\rangle \\
& O\left(\sum_{i=0}^{s-1} W(f\ i)\right) + O(W(f\ s)) + O\left(\sum_{i=s+1}^{M} W(f\ i)\right) \\
= \quad & \langle \text{Agrupar} \rangle \\
& O\left(\sum_{i=m}^{M} W(f\ i)\right)
\end{aligned}
$$

¿Por qué es mejor?

De alguna manera, podemos pensar que el argumento `min` acumula todas las sumas que tenemos que hacer para "deslizar" f. En la versión anterior, esas sumas se *repetían para cada aplicación* de f, mientras que acá, hacemos la suma antes de descender en llamada recursiva, y cada una de las $\frac{n}{2}$ hojas *reusa* esta suma.

3. Sin embargo...

*¡Las profundidades **sí** son (asintóticamente) iguales para las dos versiones!*

Ejercicio: demostrarlo. ¿Por qué las composiciones aumentan el trabajo de manera asintótica, pero no afectan la profundidad...?

4. Otros problemas tramposos

0. Verdadero o Falso: para un círculo de radio 1, su superficie es proporcional a su diámetro.

1. Si f es constante, ¿cuál es el costo de aplicar $f \circ f$?

2. Si f es constante, ¿cuál es el costo de aplicar $f^k = \underbrace{f \circ f \circ ... \circ f}_{k \text{ veces } f}$?

3. Si f es constante, ¿cuál es el costo de aplicar Δ_f, dónde $\Delta_f(n) = (\underbrace{f \circ f \circ ... \circ f}_{n \text{ veces } f})(n)$?

4. Si f es lineal (i.e. $W(f(n)) \in O(n)$), ¿cuál es el costo de aplicar $f \circ f$?

5. Encuentre el error en el siguiente razonamiento:

Teorema 1 (Teorema trucho)**.** *La recurrencia:*

$$\begin{aligned} W(0) &= 0 \\ W(n+1) &= 1 + W(n) \end{aligned}$$

está en $O(1)$.

Demostración. Por inducción. Es claro que el caso base $W(0) \in O(1)$. Para el paso inductivo, asumimos que $W(n)$ está en $O(1)$, entonces:

$$\begin{aligned} & W(n+1) \\ = \quad & \langle \text{def. } W \rangle \\ & 1 + W(n) \\ = \quad & \langle \text{HI} \rangle \\ & 1 + O(1) \\ = \quad & \langle \text{prop. notación } O \rangle \\ & O(1) \end{aligned}$$

□

¿Qué hicimos mal? ¿La última prueba de la §2 es correcta? ¿Por qué?

6. ¿Cuál es la profundidad de la siguiente función, al aplicarla a una $f \in O(1)$?

```
tabulate3 :: (Int -> a) -> Int -> BTree a
tabulate3 f n = tab 0 (n-1) where
  tab min max
    | min > max = Empty
    | otherwise =
      let (x, r) = f min ||| tab (min+1) max
       in node Empty x r
```

7. Llamemos a un árbol binario k-*balanceado* cuando su altura es a lo sumo k veces la óptima, es decir, si cumple $h \leq k \lg n$, dónde h es la altura (máxima) del árbol y n su cantidad de nodos. Por ejemplo, todo red-black tree está 2-balanceado, pero un árbol degenerado en una lista con 5 (o más) nodos no. Ahora, llamemos a un árbol *más-o-menos-balanceado* cuando existe un k tal que el árbol está k-balanceado. Describa de otra manera el conjunto de los árboles más-o-menos-balanceados. ¿Qué tan útil es esta noción de balanceo?

8. Suponga que una función f toma un entero y genera una *lista* del tamaño de su argumento. ¿Puede decir algo del trabajo y profundidad de f?

9. Suponga que una función f toma un entero y genera un *árbol binario* del tamaño de su argumento. ¿Puede decir algo del trabajo y profundidad de f?