Solución del ejercicio 2 de la práctica 4

Cálculo analítico de la complejidad temporal

Se ha de calcular la complejidad en función del parámetro n (complejidad paramétrica) haciendo uso de las relaciones de recurrencia. La función no presenta caso mejor y peor dado que solo existe una instancia por cada valor de n. Por lo tanto, la expresión de complejidad que se obtenga corresponderá a un *coste exacto*.

Sea $T(n)$ una relación de recurrencia que expresa el número de pasos de programa que realiza el algoritmo. Para obtenerla, hemos de identificar el caso base y el caso general, que se define en función de sí misma.

El caso base se obtiene directamente del algoritmo, identificando cuándo termina sin llamada recursiva. En este algoritmo ocurre cuando $n \in \{0, 1\}$. Por otra parte, hemos de calcular los pasos que realiza la función en esta situación. En el caso base de la función algorítmica, se realiza solo 1 paso de programa. Así obtenemos:

$$T(n) = 1 \text{ si } n \in \{0, 1\} \tag{2.1}$$

Para obtener el caso general de la recurrencia, en primer lugar hay que determinar la cantidad de pasos de programa que realiza el algoritmo (también en su caso general) sin tener en cuenta las llamadas recursivas; se trata, por lo tanto, del coste del `for`, que realiza n iteraciones todas ellas de coste constante:

$$c_{\texttt{for}}(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n) \tag{2.2}$$

En segundo lugar, hay que añadir la contribución de las llamadas recursivas formulándolas de manera recurrente según $T(f(n))$, donde $f(n)$ expresa la forma en la que se reduce el problema en función del valor de entrada n. En este caso tenemos cuatro llamadas recursivas; todas ellas reducen n de la misma manera: mediante la división entera $n/2$, es decir, $f(n) = n/2$. Combinando esto con lo obtenido en (2.2) se obtiene el caso general:

$$T(n) = \Theta(n) + 4T(n/2) \text{ si } n > 1 \tag{2.3}$$

Notar que $\Theta(n)$ incluye otras instrucciones cuyo coste temporal está acotado por una constante: el `cout` que está fuera del bucle y las 4 iteraciones del segundo `for`.

Con (2.1) y (2.3) obtenemos la relación de recurrencia completa:

$$T(n) = \begin{cases} 1 & n \leq 1 \\ n + 4T(n/2) & n > 1 \end{cases}$$

Para evitar expresiones farragosas en la resolución de relación de recurrencia, se ha sustituido $\Theta(n)$ por la función más sencilla incluida en ese conjunto, es decir, n.

Ahora hemos de resolver la relación de recurrencia para obtener el orden de complejidad al que pertenece. Resolverla se refiere a expresar la misma función $T(n)$ sin hacer uso de la recursividad. Este tipo de recurrencias se resuelven utilizando el método *sustitución*. Se de trata comenzar en el caso general y realizar sustituciones sucesivas de las llamadas recursivas. Cada una de ellas se reemplaza por lo que resulta de aplicarle de nuevo la recurrencia. Así hasta poder expresar $Tn)$ en función de una profundidad recursiva cualquiera k:

$$\begin{aligned}
T(n) &\overset{1}{=} n + 4T(\frac{n}{2}) \\
&\overset{2}{=} n + 4(\frac{n}{2}) + 4^2T(\frac{n}{2^2}) \\
&\overset{3}{=} n + 4(\frac{n}{2}) + 4^2(\frac{n}{2^2}) + 4^3T(\frac{n}{2^3}) \\
&\overset{4}{=} n + 4(\frac{n}{2}) + 4^2(\frac{n}{2^2}) + 4^3(\frac{n}{2^3}) + 4^4T(\frac{n}{2^4}) \\
&\dots \\
&\overset{k}{=} n\sum_{i=0}^{k-1} 2^i + 4^kT(\frac{n}{2^k}) = n(2^k - 1) + 4^kT(\frac{n}{2^k})
\end{aligned}$$

Por lo tanto, $T(n)$ expresado en función de una profundidad de recursión cualquiera k, es:

$$T(n) = n(2^k - 1) + 4^kT(\frac{n}{2^k}) \tag{2.4}$$

Recordando que se pretende expresar $T(n)$ de una manera no recurrente, de todos los posibles valores que pueda tomar k, interesa el último pues es el que permite expresar $T(\frac{n}{2^k})$ sin hacer uso de la recursividad (puesto que corresponde al caso base). Así pues, tomamos un valor cualquiera de k tal que $\frac{n}{2^k} \leq 1$. $k = \lfloor \log_2 n \rfloor$ lo cumple.

Sustituimos el valor obtenido de k en la expresión (2.4):

$$T(n) = n(2^{\lfloor \log_2 n \rfloor} - 1) + 4^{\lfloor \log_2 n \rfloor}T(1)$$

Sabiendo, a partir de la recurrencia, que $T(1) = 1$ y simplificando la nueva expresión, obtenemos:

$$T(n) = n^2 - n + n^2$$

Ya tenemos expresado $T(n)$ sin hacer uso de la recursividad. Puesto que $T(n)$ expresa el coste temporal exacto del algoritmo, concluimos que este es:

$$c_e(n) \in \Theta(n^2)$$

Solución del ejercicio 3 de la práctica 3
Cálculo analítico de la complejidad temporal

Hemos de expresar la complejidad temporal en función del tamaño del problema, al que llamaremos n. En este algoritmo viene dado por el número de elementos de la cadena que se van a procesar, es decir $n = ult - pri + 1$.

Dado que el operador lógico `and`, del lenguaje `C/C++`, no resuelve el segundo operando si el primero resulta ser `False`, el algoritmo presenta caso mejor y caso peor.

Complejidad temporal en el mejor de los casos:

Para que una instancia del problema a resolver sea del caso mejor, la expresión lógica contenida en el `return` se debe evaluar con `False` en la primera llamada al algoritmo, es decir, no se debe realizar ninguna llamada recursiva. Para que esto ocurra se ha de cumplir `pal`[`pri`] $\neq$ `pal`[`ult`], donde `pri` y `ult` son los valores que inicialmente recibe el algoritmo. En este caso, se tiene:

$$c_i(n) = 1 \in \Omega(1)$$

Complejidad temporal en el peor de los casos:

En el peor de los casos están todas las instancias para las que se realiza el máximo número posible de llamadas recursivas, es decir, todas las cadenas `pal` que forman un palíndromo entre las posiciones `pri` y `ult` que inicialmente recibe el algoritmo.

El número de pasos de programa que realiza el algoritmo en el peor de los casos viene dado por la siguiente relación de recurrencia.

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 1 + T(n-2) & n > 1 \end{cases}$$

Notar que en cada llamada recursiva el tamaño del problema se reduce en 2 unidades; el coste del algoritmo sin tener en cuenta la única llamada recursiva que hay es 1.

Resolviendo mediante sustituciones sucesivas:

$$\begin{aligned} T(n) &\overset{1}{=} 1 + T(n-2) \\ &\overset{2}{=} 2 + T(n-4) \\ &\overset{3}{=} 3 + T(n-6) \\ &\dots \\ &\overset{k}{=} k + T(n-2k) \end{aligned}$$

Por lo tanto, $T(n)$ expresado en función de una profundidad de recursión cualquiera k, es:

$$T(n) = k + T(n-2k) \tag{3.1}$$

Tomando un valor de k que cumpla $n-2k <= 1$; por ejemplo, $k = (n-1)/2$; sabiendo que $T(1) = 1$ y sustituyendo en (3.1), tenemos:

$$T(n) = \frac{n-1}{2} + 1$$

Conclusión:

$$c_s(n) \in O(n)$$

La complejidad temporal del algoritmo está entre $\Omega(1)$ y $O(n)$.