Análisis de algoritmos recursivos

Jorge Mario Londoño Peláez & Varias AI

March 25, 2025

1 Análisis por medio de árbol de llamados recursivos

El análisis por medio de árbol de llamados recursivos es una técnica que permite visualizar y analizar el costo de un algoritmo recursivo. Se construye un árbol donde cada nodo representa una llamada a la función recursiva. La raíz del árbol representa la llamada inicial, y los hijos de cada nodo representan las llamadas recursivas realizadas por ese nodo.

Para analizar el costo del algoritmo, se contabiliza la operación seleccionada como modelo de costo para el algoritmo en cada nodo del árbol. El costo total del algoritmo es la suma de los costos de todos los nodos del árbol.

Ejemplo Árbol de llamados recursivos para el algoritmo mergesort

Consideremos el algoritmo mergesort para ordenar una lista de n elementos. El árbol de llamadas recursivas tendrá una estructura donde cada nivel representa una división del arreglo en dos mitades. En cada nodo, los modelos de costo dominante son la comparación de elementos y los accesos al arreglo en los llamados a la operación `merge`. En la figura 1 se ilustra el cálculo del número de comparaciones de `mergesort` usando el árbol de llamados recursivos.

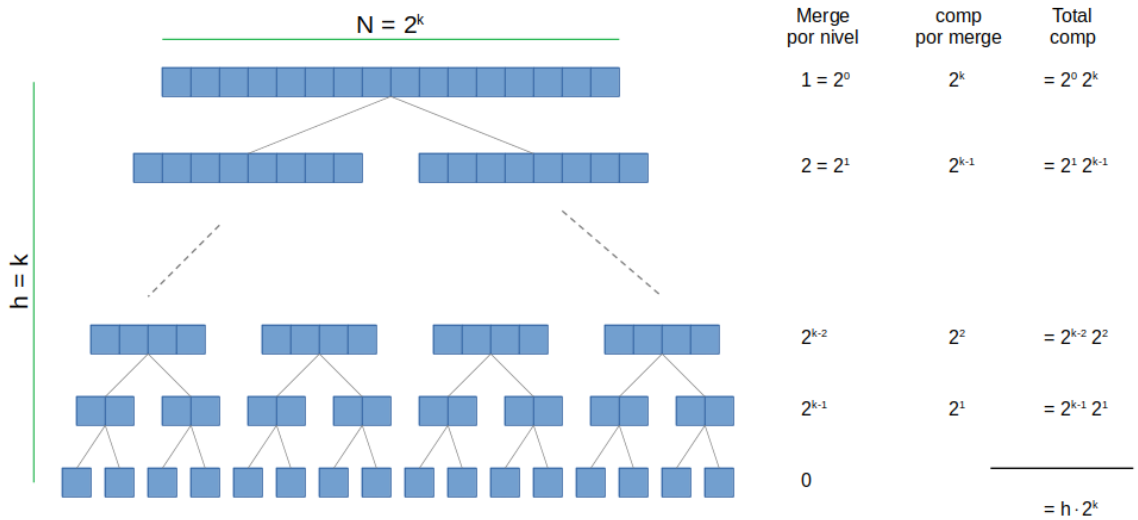

Figure 1: Árbol de invocaciones recursivas para `mergesort`

Limitaciones de esta metodología: Esta metodología puede ser compleja de aplicar si el árbol de llamadas no sigue un patrón regular, o si el costo de las operaciones en cada nodo varía sig-

nificativamente. La figura 2 ilustra una posible secuencia de llamadas recursivas para `quicksort`, mostrando las irregularidades del árbol.

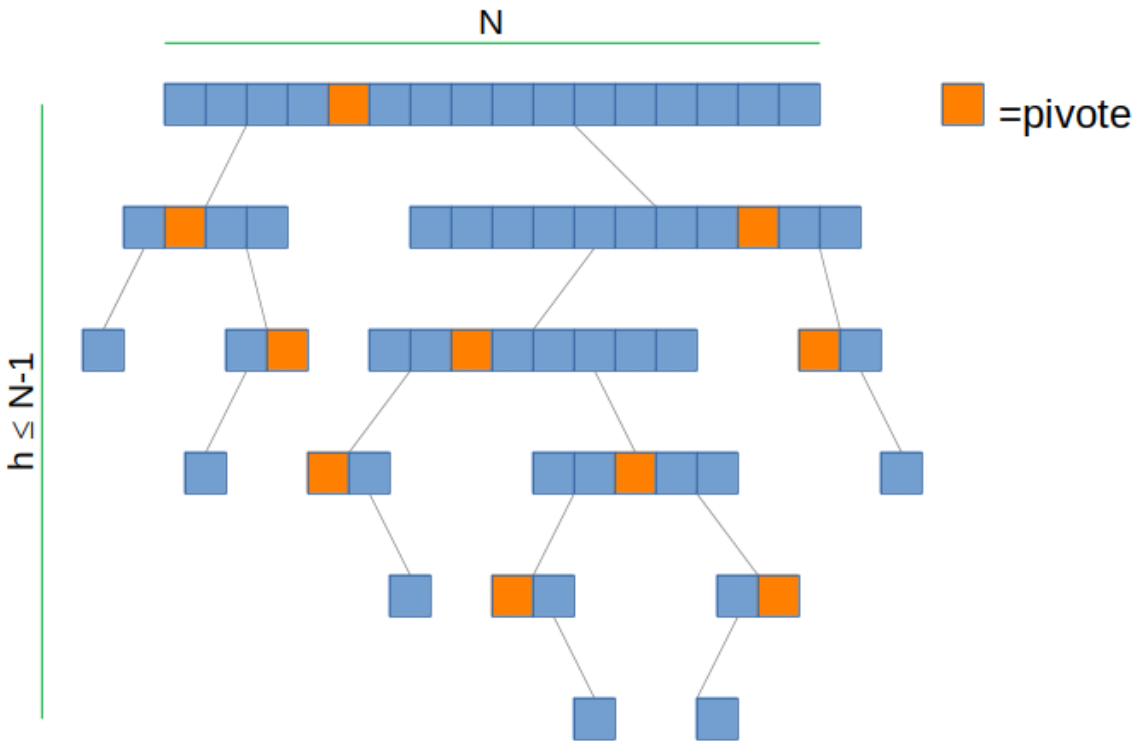

Figure 2: Árbol de invocaciones recursivas para `quicksort`

2 Análisis por medio de recurrencias

El análisis por medio de recurrencias es una técnica que permite describir el costo de un algoritmo recursivo en función del tamaño de la entrada. Una recurrencia es una ecuación que define una función en términos de sí misma, con uno o más casos base.

La recurrencia describe la cantidad de operaciones (modelo de costo) en función de una invocación del algoritmo más los llamados recursivos que se hacen.

Método de sustitución Este método permite "resolver" la recurrencia, es decir, encontrar una fórmula no recursiva para la misma función. El método de sustitución implica desarrollar la recurrencia hacia abajo hasta el caso base, y luego sustituir las respuestas del caso anterior hasta obtener la respuesta del llamado inicial.

Ejemplo Análisis por medio de recurrencias para el algoritmo `mergesort`

Se puede definir la funcion $C(N)$ como el número de comparaciones que hace `mergesort` en un intervalo de tamaño N. Se observa que al invocar el algoritmo en el intervalo de tamaño N este se subdivide en dos intervalos de tamaños $\lceil N/2 \rceil$ y $\lfloor N/2 \rfloor$, para los cuales se haran $C(\lceil N/2 \rceil)$ y $C(\lfloor N/2 \rfloor)$ comparaciones respectivamente. Considerando además que en el llamado a `merge` se hacen como máximo N comparaciones, se obtiene la recurrecia:

$$C(N) = N + C\left(\left\lceil \frac{N}{2} \right\rceil\right) + C\left(\left\lfloor \frac{N}{2} \right\rfloor\right)$$

Haciendo el cambio de variable $N = 2^H$ y agrupando se obtiene:

$$C(2^H) = 2^H + 2C\left(2^{H-1}\right)$$

Que se puede resolver aplicando el método de sustitución, y teniendo presente que en el caso base del algoritmo se hacen 0 comparaciones, es decir, $C(1) = 0$.

$$\begin{aligned}
C(2^H) &= 2^H + 2C\left(2^{H-1}\right) \\
C(2^{H-1}) &= 2^{H-1} + 2C\left(2^{H-2}\right) \\
&\vdots \\
C(2^1) &= 2^1 + 2C\left(2^{H-H}\right) = 2^1 \\
C(2^2) &= 2^2 + 2C\left(2^1\right) = 2^2 + 2(2^1) \\
C(2^3) &= 2^3 + 2C\left(2^2\right) = 2^3 + 2(2^2 + 2^2) \\
&\vdots \\
C(2^H) &= 2^H + 2(2^{H-1} + \ldots + 2^{H-1}) = \underbrace{2^H + \ldots + 2^H}_{\text{H veces}} \\
C(2^H) &= 2^H \cdot H \\
C(N) &= N\lceil \lg(N) \rceil
\end{aligned}$$

Ejemplo Análisis por medio de recurrencias para el algoritmo `quicksort`

El análisis de Quicksort por medio de recurrencias es más complejo debido a que el tamaño de los subproblemas depende de la elección del pivote. En el mejor de los casos, el pivote divide el arreglo en dos subarreglos de tamaño aproximadamente igual. En el peor de los casos, el pivote resulta ser el elemento más pequeño o el más grande, lo que lleva a un subarreglo de tamaño 0 y otro de tamaño $n - 1$.

Peor caso En el peor caso, la recurrencia para el número de comparaciones $C(n)$ es:

$$C(n) = C(n-1) + n$$

Esto se debe a que, en cada paso, se realiza una partición que requiere n comparaciones y se genera un subproblema de tamaño $n - 1$. Desarrollando la recurrencia:

$$\begin{aligned}
C(n) &= C(n-1) + n \\
&= C(n-2) + (n-1) + n \\
&= C(n-3) + (n-2) + (n-1) + n \\
&\vdots \\
&= C(1) + 2 + 3 + \ldots + n
\end{aligned}$$

Dado que $C(1) = 0$, la suma resultante es la suma de los primeros n enteros menos 1:

$$C(n) = \sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1 = \Theta(n^2)$$

Mejor caso En el mejor caso, cada partición divide el arreglo en dos subarreglos de tamaño aproximadamente $n/2$. La recurrencia para el número de comparaciones $C(n)$ es:

$$C(n) = 2C(n/2) + n$$

Esta recurrencia es similar a la de Mergesort. Aplicando el teorema maestro, con $a = 2$, $b = 2$, y $f(n) = n$, se tiene que $n^{\log_b a} = n^{\log_2 2} = n$. Como $f(n) = \Theta(n)$, estamos en el Caso 2 del teorema maestro, y la solución es $C(n) = \Theta(n \lg n)$.

Caso promedio El análisis del caso promedio es más intrincado, pero se puede demostrar que también tiene un costo de $\Theta(n \ln n)$. La recurrencia para el caso promedio es:

$$C(n) = n + \frac{1}{n} \sum_{i=1}^{n} (C(i-1) + C(n-i))$$

Esta recurrencia refleja que cada posible pivote tiene la misma probabilidad de ser elegido, y se suman los costos de las dos subllamadas recursivas resultantes. La solución de esta recurrencia es $C(n) = \Theta(n \ln n)$.

3 Teorema maestro

El teorema maestro proporciona una solución directa para recurrencias de la forma $T(n) = aT(n/b) + f(n)$, donde $a \geq 1$ y $b > 1$ son constantes, y $f(n)$ es una función asintóticamente positiva. Este teorema es útil para determinar el orden de crecimiento de algoritmos recursivos.

Expresión para el teorema maestro y distintos casos de aplicación:

El teorema maestro establece que, dependiendo de la relación entre $f(n)$ y $n^{\log_b a}$, se pueden determinar tres casos:

- Caso 1: Si $f(n) = O(n^{\log_b a - \epsilon})$ para alguna constante $\epsilon > 0$, entonces $T(n) = \Theta(n^{\log_b a})$.
- Caso 2: Si $f(n) = \Theta(n^{\log_b a})$, entonces $T(n) = \Theta(n^{\log_b a} \log n)$.
- Caso 3: Si $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguna constante $\epsilon > 0$, y si $af(n/b) \leq cf(n)$ para alguna constante $c < 1$ y n suficientemente grande, entonces $T(n) = \Theta(f(n))$.

Ejemplo: Aplicación del teorema maestro para el algoritmo `mergesort`

Para el algoritmo `mergesort`, la recurrencia es $C(n) = 2C(n/2) + O(n)$. Aquí, $a = 2$, $b = 2$, y $f(n) = O(n)$. Por lo tanto, $n^{\log_b a} = n^{\log_2 2} = n$. Como $f(n) = \Theta(n)$, estamos en el Caso 2 del teorema maestro, y la solución es $C(n) = \Theta(n \log n)$.

Limitaciones del teorema maestro: El teorema maestro no se puede aplicar a todas las recurrencias. Tiene limitaciones en cuanto a la forma de la recurrencia y las condiciones que deben cumplir la función $f(n)$ y las constantes a, b.