MAC5711 – Análise de Algoritmos

Departamento de Ciência da Computação – IME/USP

Segundo Semestre de 2005

1. Recorrências

Uma recorrência é uma equação ou desigualdade que descreve uma função por meio de seus valores para entradas menores.

Em análise de algoritmos, é comum deduzirmos recorrências envolvendo notação assintótica a partir de algoritmos recursivos. Por exemplo, a recorrência do tempo de pior caso do algoritmo MERGESORT é a seguinte:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n). \tag{1}$$

Nestes casos, o que queremos determinar é apenas a ordem de grandeza da função $T(n)$.

Há vários métodos para se deduzir a ordem de grandeza de uma função que satisfaz uma recorrência deste tipo. O primeiro passo nessa dedução é nos concentrarmos numa recorrência tradicional (ou seja, sem a notação assintótica e definida completamente) derivada da recorrência original. No caso do nosso exemplo, derivamos de (1) a recorrência:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{se } n = 2, 3, 4, \ldots. \end{cases}$$

Observe a relação entre esta recorrência e (1). Primeiramente estabelecemos o caso base como sendo 1 (o mais simples possível) e decidimos arbitrariamente que o valor de $T(1) = 1$. Além disso, substituímos $\Theta(n)$ por n, um representante significativo da classe $\Theta(n)$ o mais simples possível. Na verdade, em vez de $T(n)$ nessa recorrência completa, deveríamos ter escrito o nome de uma outra função, pois esta não é de fato a nossa $T(n)$ inicial. Ela é apenas uma função auxiliar cujo comportamento assintótico determinamos para deduzir o comportamento assintótico da nossa $T(n)$ original. No entanto, abusamos da notação e escrevemos a recorrência com a própria $T(n)$.

Usando o exemplo acima, indicaremos a maneira de se resolver uma recorrência como esta.

Determinar a solução exata de uma recorrência deste tipo pode ser uma tarefa difícil. Devemos lembrar então que na verdade não estamos interessados na solução *exata* da recorrência, mas na ordem de grandeza da sua solução. Determinar a ordem de grandeza da solução desta recorrência é uma tarefa mais simples do que resolvê-la exatamente. Ela consiste em duas etapas, das quais a primeira essencialmente resolve o problema.

A primeira etapa consiste em nos restringirmos a valores de n que são potências de 2. Para tais valores, a recorrência se reduz a

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2\,T(n/2) + n & \text{se } n = 2, 4, 8, 16, \ldots. \end{cases}$$

Vamos mostrar como resolver uma recorrência deste tipo usando o método da substituição. Para isso, precisamos primeiramente encontrar um "chute" de solução da recorrência. Um jeito de determinar tal chute é fazendo a expansão da recorrência ou de sua árvore de recursão. Aqui mostraremos como fazer a expansão. Suponha que n

é uma potência de 2 grande. Neste caso, temos que

$$\begin{aligned} T(n) &= 2\,T(n/2) + n \\ &= 2\left(2\,T(n/2^2) + n/2\right) + n \;=\; 2^2\,T(n/2^2) + n + n \\ &= 2^2\left(2\,T(n/2^3) + n/2^2\right) + n + n \;=\; 2^3\,T(n/2^3) + n + n + n \\ &= \cdots \;=\; 2^k\,T(n/2^k) + kn, \end{aligned}$$

para um k qualquer não maior que $\lg n$.

Tomando-se $k = \lg n$, concluímos que $T(n) = nT(1) + n\lg n = n + n\lg n$. Esse é o nosso chute de solução para a recorrência para n potência de 2. Para verificar que este chute está correto, basta fazer uma prova por indução em k, onde $n = 2^k$. É esta prova por indução que é chamada de *método da substituição*.

Prova por indução em k que $T(2^k) = 2^k + k\,2^k$.
Base: $k = 0$.

Neste caso, $T(2^k) = T(1) = 1$ e $2^0 + 0.2^0 = 1$, ou seja, a fórmula vale.
Passo: $k > 0$.

Neste caso,

$$\begin{aligned} T(2^k) &= 2\,T(2^{k-1}) + 2^k \\ &= 2\left(2^{k-1} + (k-1)\,2^{k-1}\right) + 2^k && \text{por indução} \\ &= 2^k + (k-1)\,2^k + 2^k \\ &= 2^k + k\,2^k, \end{aligned}$$

ou seja, a fórmula vale, completando a prova por indução.

Dessa primeira etapa, deduzimos uma fórmula fechada para a solução da recorrência para valores de n que são potências de 2. Na segunda etapa, vamos mostrar que a ordem de grandeza de $T(n)$ para potências de 2 se aplica a $T(n)$ em geral, desde que $T(n)$ satisfaça uma restrição muito natural, que em particular vale para as funções que usualmente estudamos em análise de algoritmos.

Vamos mostrar que, se $T(n)$ é uma função não-decrescente e $T(n) = n + n\lg n$ para n potência de 2, então $T(n) = \Theta(n\lg n)$. Começaremos mostrando que $T(n) = O(n\lg n)$. Para isso, seja n um inteiro positivo arbitrário e seja m o inteiro tal que $2^{m-1} < n \leq 2^m$. Temos que, pelas hipóteses,

$$\begin{aligned} T(n) &\leq T(2^m) \\ &= 2^m + m2^m \\ &\leq 2n + (\lg n + 1)2n \\ &= 4n + 2n\,\lg n \\ &\leq 4n\,\lg n, && \text{para todo } n \geq 4. \end{aligned}$$

Disso concluímos que $T(n) = O(n\lg n)$. De maneira análoga, podemos demonstrar que $T(n) = \Omega(n\lg n)$. Ou seja, de fato $T(n)$ tem a ordem de grandeza determinada pela solução da recorrência restrita a valores de n que são potências de 2.

Geralmente, quando resolvemos uma recorrência deste tipo em análise de algoritmos, omitimos a segunda etapa da prova. Ou seja, provamos a solução da recorrência para potências de 2 e deduzimos, diretamente dali, a ordem de grandeza da solução da recorrência original.