INF 02779 Análise de Algoritmos - 2002/2	20/01/03

Aula 9: Método da Árvore de Recursão

Instrutor: Berilhes Borges Garcia — Escriba: Rodrigo Pessoa

DRAFT

1 Método da Árvore de Recursão

Este método é uma forma conveniente de vizualizar o que ocorre quando a recursão é iterada. Desta forma este método ajuda a organizar a estrutura algébrica necessária para resolver uma recorrência. Adicionalmente este método ajuda na determinação de uma boa adivinhação para o método da substituição.
Example: Considere a relação de recorrência,

$$T(n) = \left\{ \begin{array}{ll} T(n) = 2T\left(\frac{n}{2}\right) + n^2 & se\ n > 1, \\ 1 & se\ n = 1. \end{array} \right\} \tag{1}$$

Nós agora desenhamos o começo da árvore de recursão para a recorrência acima. Por agora, assuma que n é uma potência de 2. Um diagrama árvore de recursão tem três partes. Do lado esquerdo, nós mantemos a trilha do tamanho do problema, no meio nós desenhamos a árvore, e do lado direito, nós montamos a trilha do trabalho já feito. Dessa forma no começo da árvore de recursão deste exemplo nós tomamos um problema de tamanho n, dividimos este em 2 problemas de tamanho $\frac{n}{2}$ e anotamos do lado direito que nós tivemos o custo de n^2 para fazer o trabalho neste nível da árvore. O fato que nós quebramos o problema em dois problemas é representado desenhando-se dois filhos do no raiz (nível 0). Isto resulta na seguinte figura:

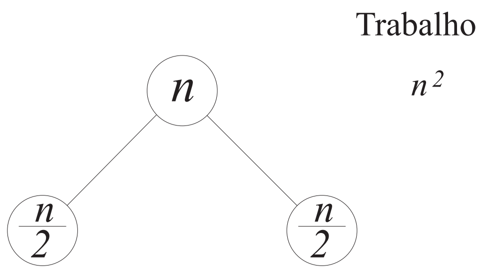

Figure 1:

Nós continuamos a desenhar a árvore desta maneira, até atingir a condição de fronteira, ou seja o caso base da recursão.
Dessa forma para desenhar uma árvore de recursão nós necessitamos determinar:

- o número de subproblemas;
- o tamanho de cada subproblema;

- o trabalho feito em cada nível da árvore;
- e a altura da árvore de recursão.

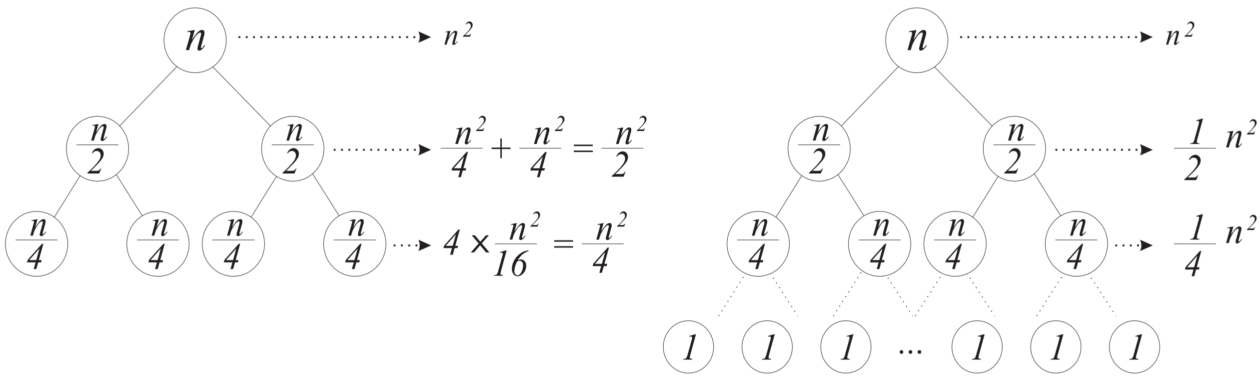

Figure 2:

Nós podemos ver que para este problema, no nível 2, nós temos 2^i subproblemas de tamanho $\frac{n}{2^i}$. dAdicionalmente, uma vez que um problema de tamanho $\frac{n}{2^i}$ requer $\left(\frac{n}{2^i}\right)^2$ unidades de trabalho para ser executado, há $\left(2^i\right)\left(\frac{n}{2^i}\right)^2 = \frac{n^2}{2^i}$ unidades de trabalho adicional por nível . De modo a determinar quantos níveis existem na árvore, nós observamos que em cada nível o problema é reduzido pela metade, e a árvore para quando o tamanho do problema é 1. Portanto existem $\lg n + 1$ níveis na árvore. E a altura da árvore é $\lg n$. Em cada nível da árvore, a exceção do último nível, nós temos que o montante de trabalho feito neste nível é igual a $\frac{n^2}{2^i} > 0 \leq i \leq \lg n - 1$.
O último nível é diferente dos demais níveis. Nos outros níveis, o custo é descrito pela parte recursiva da recorrência, neste caso $T(n) = 2T\left(\frac{n}{2}\right) + n^2$. No último nível, a carga de trabalho vem do caso base. Dessa forma nós devemos computar o número de problemas de tamanho 1, e então multiplicar este valor por $T(1)$.
A coisa importante é que nós sabemos exatamente quantos níveis existem, e a quantidade de trabalho feita em cada nível. Uma vez que nós conhecemos estes fatos, nós podemos somar a quantidade de trabalho feita em cada nível. Neste exemplo a quantidade de trabalho feita é:

$$T(n) = nT(1) + \sum_{i=0}^{\lg n-1} \frac{n^2}{2^i}, = \Theta(n^2) \tag{2}$$

O trabalho total feito pela árvore de recursão é a solução para nossa recorrência, porque a árvore simplesmente modela o processo de iteração da recorrência. O método da árvore de recorrência pode ser utilizado para dar uma boa indicação da aproximação que será utilizada no método da substituição.
Uma outra forma de observar que a recorrência (1) tem solução $\Theta(n^2)$ é observar que o primeiro nível tem um custo de n^2, o segundo nível tem um custo de $\left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^2 = \frac{n^2}{2}$ o terceiro nível tem custo $\left(\frac{n}{4}\right)^2 + \left(\frac{n}{4}\right)^2 + \left(\frac{n}{4}\right)^2 + \left(\frac{n}{4}\right)^2 = \frac{n^2}{4}$, e assum sucessivamente. Desa forma os custos decrescem geometricamente, portanto o custo total é no máximo um fator constante maior que o primeiro termo, e assim a solução é $\Theta(n^2)$. ⊠

Considere agora mais uma recorrência:

$$T\left(n\right)=\left\{\begin{array}{ll} T\left(\frac{n}{2}\right)+n & se\ n>1, \\ 1 & se\ n=1. \end{array}\right\} \tag{3}$$

Novamente, vamos assumir que n é uma potência de dois. Nós podemos interpretar esta recorrência da seguinte forma: para resolver um problema de tamanho n, nós devemos resolver um problema de tamanho $\frac{n}{2}$ e nós fazemos n unidades de trabalho adicional.
Vamos desenhar a árvore de recursão para este problema:

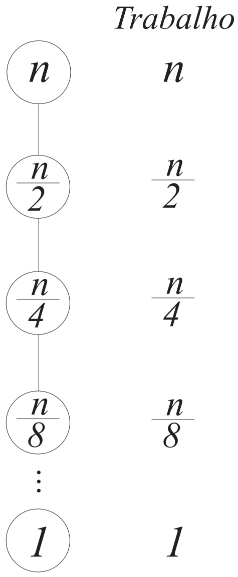

Figure 3:

Nós vemos neste exemplo que o tamanho dos problemas permanece o mesmo como na árvore anterior. Contudo, todo o resto é diferente. O número de subproblemas não duplica, este permanece o mesmo em cada nível. Consequentemente a quantidade de trabalho fica reduzida a metade em todo nível. Note que existe ainda $\ln n+1$ níveis. Dessa forma no nível i, nós temos 1 problema de tamanho $\frac{n}{2^i}$, e um montante de trabalho de $\frac{n}{2^i}$.
Nós agora desejamos calcular a quantidade total de trabalho feita, temos que o trabalho total é:

$$\begin{aligned} T(n) &= n+\frac{n}{2}+\frac{n}{4}+...+2+1 \\ &= \sum_{i=0}^{\lg n}\frac{n}{2^i} \\ &= n\sum_{i=0}^{\lg n}(\frac{1}{2})^i \leq n\sum_{i=0}^{\infty}(\frac{1}{2})^i = n \\ T(n) &= O(n). \end{aligned} \tag{4}$$

Mas $T(n)=\Omega(n)$, portanto, $T(n)=\Theta(n)$.

O que é importante para nós não é a forma "exata" da solução, mas um limite big O, ou melhor, um limite big Θ sob a solução.
Considere agora a seguinte recorrência:

$$T(n) = \left\{ \begin{array}{ll} T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n & se\ n > 1, \\ 1 & se\ n = 1. \end{array} \right\} \quad (5)$$

A árvore de recursão desta recorrência é mostrada abaixo:

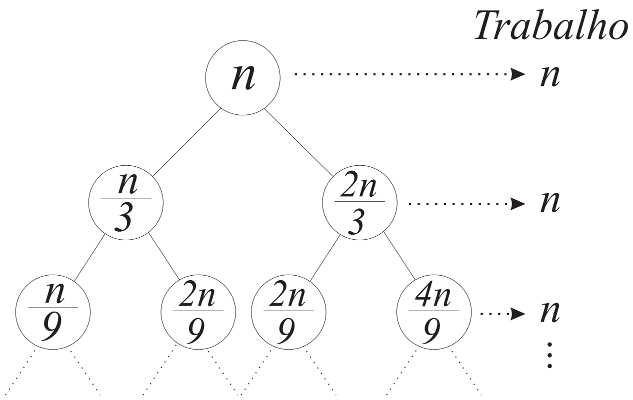

Figure 4:

Quando nós adicionamos os valores de cada nível, nós obtemos um valor de n. Mas será que o custo de cada nível é sempre n? O caminho mais longo da raiz para uma folha é $n \rightarrow (2/3)n \rightarrow (2/3)^2 n \rightarrow ... \rightarrow 1$. Uma vez que $(2/3)^k n = 1$ quando $k = \log_{(3/2)} n$, a altura da árvore é $\log_{\frac{3}{2}} n$. Assim a solução da recorrência é no máximo $n \log_{\frac{3}{2}} n = O(n \lg n)$.
Será que nós podemos fazer melhor que isto?