Análise de algoritmos

Introdução à Ciência de Computação II

Baseados nos Slides do Prof. Dr. Thiago A. S. Pardo

Resolução de recorrências

- Método mestre
  - <u>Fornece limites</u> para recorrências da forma $T(n)=aT(n/b)+f(n)$, em que $a\geq 1$, $b>1$ e $f(n)$ é uma função dada
  - Envolve a memorização de alguns casos básicos que podem ser aplicados para muitas recorrências simples

Método mestre

- Seja
  $T(n) = aT(n/b) + f(n)$,
  em que $a \geq 1$, $b > 1$ e $f(n)$ é uma função dada.

- A equação acima descreve o tempo de cálculo de um problema de tamanho n dividido em
  - a subproblemas de tamanha n/b cada
  - Cada um dos a subproblemas são resolvidos recursivamente em tempo $T(n/b)$
  - $f(n)$ representa o custo de se dividir o problema e combinar os resultados dos subproblemas

Método mestre

- Observação em
  $T(n)=aT(n/b)+f(n)$

- Como tratar os casos em que n/b não gera um número inteiro
  - É possível mostrar que o arredondamento para cima ou para baixo não afeta a solução assintótica
  - Ver demonstração em **Cormen et al. 2009.**

Método mestre

- Seja a >= 1 e b > 1 constantes. Seja f(n) uma função. Considere que T(n) seja definida para n>0 (n é inteiro) como
  - $T(n) = a\, T(n/b) + f(n)$

- Então T(n) possui umas das seguintes formas assintóticas
  - Se $f(n) = O(n^{\log_b a - e})$ para algum e>0, então $T(n) = \Theta(n^{\log_b a})$.
  - Se $f(n) = \Theta(n^{\log_b a})$, então $T(n) = \Theta(n^{\log_b a} \lg n)$
  - Se $f(n) = \Omega(n^{\log_b a + e})$ para algum e>0, e se $a\,f(n/b) <= cf(n)$ para alguma constante c < 1 e todo n suficientemente grande, então $T(n) = \Theta(f(n))$

- Ver demonstração em **Cormen et al. 2009.**

Método mestre

Em todos os três casos, f(n) é comparada com $n^{\log_b a}$

Caso 1: Se $n^{\log_b a}$ é maior que f(n), a solução é $T(n) = \Theta(n^{\log_b a})$.

Caso 3: Se f(n) é maior que $n^{\log_b a}$ então $T(n) = \Theta(f(n))$

Caso 2: Se as funções tem o mesmo tamanho, multiplica-se por um fator logaritmo para obter $T(n) = \Theta(n^{\log_b a} \lg n)$

Em (1), f(n) tem que ser menor do que $n^{\log_b a}$ por um fator n^e para alguma constante $e > 0$

Em (3), f(n) tem que ser maior do que $n^{\log_b a}$ por um fator n^e para alguma constante $e > 0$. Além disso, a condição de regularidade $a f(n/b) <= c f(n)$ deve ser satisfeita

Método mestre

- Alguns *gaps* entre as 3 possibilidades possíveis
  - f(n) é assintoticamente menor que ($n^{\log_b a}$), mas não polinomialmente
  - f(n) é assintoticamente maior que ($n^{\log_b a}$), mas não polinomialmente
  - A condição de regularidade não é satisfeita

Método mestre: exemplos

- $T(n) = 9\ T(n/3) + n$

$a = 9;\ b = 3;\ f(n) = n;\ n^{\log_b a} = n^{\log_3 9} = n^2$

Caso 1

Fazendo $e = 1 \rightarrow f(n) = n = O(n^{2-1}) = O(n)$

$\rightarrow$ Caso 1: $T(n) = \Theta(n^{\log_b a})$

$\rightarrow T(n) = \Theta(n^2)$.

Método mestre: exemplos

- $T(n) = T(2n/3) + 1$

Método mestre: exemplo

- $T(n) = T(2n/3) + 1$

$a=1$, $b = 3/2$ e $f(n) = 1$

$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Caso 2: $f(n) = \Theta(n^{\log_b a})$

→ $T(n) = \Theta(1.\lg n)$

Método mestre: exemplo

- $T(n) = 3T(n/4) + n \lg n$

Método mestre: exemplo

- $T(n) = 3T(n/4) + n \lg n$

$a = 3; b = 4; f(n) = n \lg n$

$n^{\log_b a} = n^{\log_4 3} = n^{0.793}$

Caso 3: $f(n) = \Omega(n^{\log_b a + e})$, com $e = 0.2$

Regularidade:

$a\, f(n/b) = 3(n/4) \lg(n/4) <= (3/4)\, n \lg n = c\, f(n)$

$T(n) = \Theta(n \lg n)$

Método mestre

- $T(n) = 2\,T(n/2) + n \lg n$

Método mestre

- Usando o método mestre, calcule a complexidade da busca binária recursiva
  - T(n) = ?

Método mestre

- Complexidade da busca binária recursiva

$$T(n) = T(n/2) + \Theta(c)$$

$$n^{\log_b a} = n^{\log_2 1} = 1 = \Theta(c)$$

$$f(n) = \Theta(c)$$

Caso 2: $T(n) = \Theta\ (n^{\log_b a} \lg n) = \Theta(1.\lg n)$

Método mestre

- Resolvendo o problema do mergeSort
  - $T(n) = 2T(n/2) + \Theta(n)$

Método mestre

- Resolvendo o problema do mergeSort
  - $T(n) = 2T(n/2) + \Theta(n)$

$a=2; b=2; f(n) = \Theta(n)$

$n^{\log_a b} = n^{\log_a b} = n$

Caso 2: $f(n) = \Theta(n^{\log_a b}) = \Theta(n) \rightarrow$

$T(n) = n^{\log_a b} \lg n = n \lg n$

Método mestre

- Algoritmo recursivo para multiplicação de matrizes (não otimizado)
  - $T(n) = 8T(n/2) + \Theta(n^2)$

Método mestre

- Algoritmo recursivo para multiplicação de matrizes
- $T(n) = 8T(n/2) + \Theta(n^2)$

$n^{\log_b a} = n^{\log_2 8} = n^3$ -- $f(n) = O(n^2)$

Caso 1: $f(n) = O(n^{3-e})$, com e=1

Portanto, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

Método mestre

- Algoritmo *Strassen* para multiplicação de matrizes
  - $T(n) = 7T(n/2) + \Theta(n^2)$

Método mestre

- Algoritmo *Strassen* para multiplicação de matrizes

$T(n) = 7T(n/2) + \Theta(n^2)$

$n^{\log_b a} = n^{\log_2 7}$

Lembrando que $2.80 < \log_2 7 < 2.81$, temos

$f(n) = O(n^{\log_2 7 - e})$, para $e = 0.8$,

Caso 1: $T(n) =$ $T(n) = \Theta(n^{\log_b a}) = \Theta\ (n^{2.807})$