SIN5013 - Análise de Algoritmos e Estruturas de Dados

Resolução de recorrências

Prof. Flávio Luiz Coutinho

Resolução de recorrências

No processo de análise de algoritmos recursivos, após aplicar o método a fim de determinar a função T, nos deparamos com uma função definida em função de si própria (ou seja, definida recursivamente) que chamamos de recorrência.

max1: $T(n) = k1$ [para $n = 1$]

$= T(n - 1) + k2$ [para $n > 1$]

max2: $T(n) = k1$ [para $n = 1$]

$= 2T(n / 2) + k2$ [para $n > 1$]

Resolução de recorrências

Para que possamos ter uma ideia melhor de qual é a cara da função (ou seja, determinar sua complexidade assintótica), precisamos **resolver a recorrência**, de modo a se obter uma “fórmula fechada”, ou seja, que é definida sem depender de si mesma.

Há **3 métodos** que podem ser aplicados para se resolver recorrências:

- Método iterativo
- Prova por indução
- Teorema Mestre

Resolução de recorrências (método iterativo)

Algoritmo max2 (divide um problema de tamanho n em dois subproblemas de tamanho n/2 que são resolvidos recursivamente):

$$T(n) = k1 \quad \text{(para } n = 1\text{)}$$

$$= 2T(n/2) + k2 \quad \text{(para } n > 1\text{)}$$

Como usar o método iterativo para resolver esta recorrência?

- Expansões sucessivas da fórmula usando a própria definição de T.

Resolução de recorrências

(0) $T(n)$

Resolução de recorrências

(0) $T(n)$

(1) $T(n) \quad = \quad 2T(n/2) + k2$

Resolução de recorrências

(0) $T(n)$

(1) $T(n) \quad = \quad 2T(n/2) + k2$

Resolução de recorrências

(0) $T(n)$

(1) $T(n) = 2T(n/2) + k2$

(2) $T(n) = 2\,[\,2T(n/4) + k2\,] + k2$

$T(n/2) = 2T(n/4) + k2$

Resolução de recorrências

(0) $T(n)$

(1) $T(n) = 2T(n/2) + k2$

(2) $T(n) = 2\ [\ 2T(n/4) + k2\ ] + k2$

(2) $T(n) = 4T(n/4) + 2k2 + k2$

$T(n/2) = 2T(n/4) + k2$

Resolução de recorrências

(0) $T(n)$

(1) $T(n) = 2T(n/2) + k2$

(2) $T(n) = 2\,[\,2T(n/4) + k2\,] + k2$ $\qquad T(n/2) = 2T(n/4) + k2$

(2) $T(n) = 4T(n/4) + 2k2 + k2$

Resolução de recorrências

(0) $T(n)$

(1) $T(n) = 2T(n/2) + k2$

(2) $T(n) = 2\,[\,2T(n/4) + k2\,] + k2$ $\qquad T(n/2) = 2T(n/4) + k2$

(2) $T(n) = 4T(n/4) + 2k2 + k2$

(3) $T(n) = 4\,[\,2T(n/8) + k2\,] + 2k2 + k2$ $\qquad T(n/4) = 2T(n/8) + k2$

Resolução de recorrências

(0) $T(n)$

(1) $T(n) = 2T(n/2) + k2$

(2) $T(n) = 2 [2T(n/4) + k2] + k2$ $\qquad T(n/2) = 2T(n/4) + k2$

(2) $T(n) = 4T(n/4) + 2k2 + k2$

(3) $T(n) = 4 [2T(n/8) + k2] + 2k2 + k2$ $\qquad T(n/4) = 2T(n/8) + k2$

(3) $T(n) = 8T(n/8) + 4k2 + 2k2 + k2$

Resolução de recorrências

(0)	$T(n)$			
(1)	$T(n)$	$=$	$2T(n/2) + k2$	
(2)	$T(n)$	$=$	$2\,[\,2T(n/4) + k2\,] + k2$	$T(n/2) = 2T(n/4) + k2$
(2)	$T(n)$	$=$	$4T(n/4) + 2k2 + k2$	
(3)	$T(n)$	$=$	$4\,[\,2T(n/8) + k2\,] + 2k2 + k2$	$T(n/4) = 2T(n/8) + k2$
(3)	$T(n)$	$=$	$8T(n/8) + 4k2 + 2k2 + k2$	

Resolução de recorrências

(0)	$T(n)$			
(1)	$T(n)$	=	$2T(n/2) + k2$	
(2)	$T(n)$	=	$2\,[\,2T(n/4) + k2\,] + k2$	$T(n/2) = 2T(n/4) + k2$
(2)	$T(n)$	=	$4T(n/4) + 2k2 + k2$	
(3)	$T(n)$	=	$4\,[\,2T(n/8) + k2\,] + 2k2 + k2$	$T(n/4) = 2T(n/8) + k2$
(3)	$T(n)$	=	$8T(n/8) + 4k2 + 2k2 + k2$	
(4)	$T(n)$	=	$8\,[\,2T(n/16) + k2\,] + 4k2 + 2k2 + k2$	$T(n/8) = 2T(n/16) + k2$
(4)	$T(n)$	=	$16T(n/16) + 8k2 + 4k2 + 2k2 + k2$	

Resolução de recorrências

(4) $T(n) = 16T(n/16) + 8k2 + 4k2 + 2k2 + k2$

(4) $T(n) = 16T(n/16) + k2(8 + 4 + 2 + 1)$

(4) $T(n) = 2^4\ T(n / 2^4) + k2 * \Sigma\ (2^j)$ [para j = 0 até 3]

Resolução de recorrências

(4) $T(n) = 16T(n/16) + 8k2 + 4k2 + 2k2 + k2$

(4) $T(n) = 16T(n/16) + k2(8 + 4 + 2 + 1)$

(4) $T(n) = 2^4 T(n / 2^4) + k2 * \Sigma (2^j)$ [para j = 0 até (4 - 1)]

Resolução de recorrências

(4) T(n) = 16T(n/16) + 8k2 + 4k2 + 2k2 + k2

(4) T(n) = 16T(n/16) + k2(8 + 4 + 2 + 1)

(4) T(n) = 2^4 T (n / 2^4) + k2 * Σ (2^j) [para j = 0 até (4 - 1)]

…

(i) T(n) = 2^i T (n / 2^i) + k2 * Σ (2^j) [para j = 0 até (i - 1)]

Resolução de recorrências

(4) $T(n) = 16T(n/16) + 8k2 + 4k2 + 2k2 + k2$

(4) $T(n) = 16T(n/16) + k2(8 + 4 + 2 + 1)$

(4) $T(n) = 2^4\, T(n / 2^4) + k2 * \Sigma (2^j)$ [para $j = 0$ até $(4 - 1)$]

…

(i) $T(n) = 2^i\, T(n / 2^i) + k2 * \Sigma (2^j)$ [para $j = 0$ até $(i - 1)$]

…

(?)

Resolução de recorrências

(4) $T(n) = 16T(n/16) + 8k2 + 4k2 + 2k2 + k2$

(4) $T(n) = 16T(n/16) + k2(8 + 4 + 2 + 1)$

(4) $T(n) = 2^4 T(n / 2^4) + k2 * \Sigma (2^j)$ [para j = 0 até (4 - 1)]

…

(i) $T(n) = 2^i T(n / 2^i) + k2 * \Sigma (2^j)$ [para j = 0 até (i - 1)]

…

(?)

Até quando fazer o processo de expansão (ou: quantas iterações são possíveis)?

Resolução de recorrências

(4) $T(n) = 16T(n/16) + 8k2 + 4k2 + 2k2 + k2$

(4) $T(n) = 16T(n/16) + k2(8 + 4 + 2 + 1)$

(4) $T(n) = 2^4\, T(n / 2^4) + k2 * \Sigma\, (2^j)$ [para j = 0 até (4 - 1)]

…

(i) $T(n) = 2^i\, T(n / 2^i) + k2 * \Sigma\, (2^j)$ [para j = 0 até (i - 1)]

…

(?)

Até quando fazer o processo de expansão (ou: quantas iterações são possíveis)?

Até chegar ao caso base!

Resolução de recorrências

(4) $T(n) = 16T(n/16) + 8k2 + 4k2 + 2k2 + k2$

(4) $T(n) = 16T(n/16) + k2(8 + 4 + 2 + 1)$

(4) $T(n) = 2^4\, T(n / 2^4) + k2 * \Sigma(2^j)$ [para j = 0 até (4 - 1)]

…

(i) $T(n) = 2^i\, T(n / 2^i) + k2 * \Sigma(2^j)$ [para j = 0 até (i - 1)]

…

(?)

Até quando fazer o processo de expansão (ou: quantas iterações são possíveis)?

Até chegar ao caso base! Que ocorre na iteração i_max quando:

$n / 2^{(i\_max)} = 1$ -----> $n = 2^{(i\_max)}$ -----> $i\_max = \log2(n)$

Resolução de recorrências

(i)	$T(n)$	=	$2^i T(n / 2^i) + k2 * \Sigma (2^j)$	[$j = 0$ até $(i - 1)$]
…				
$(\log_2(n))$	$T(n)$	=	$2^{\log_2(n)} T(1) + k2 * \Sigma (2^j)$	[$j = 0$ até $(\log_2(n) - 1)$]

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	

Resolução de recorrências

(i)	$T(n)$	=	$2^i T(n / 2^i) + k2 * \Sigma (2^j)$	[$j = 0$ até $(i - 1)$]
…				
$(\log2(n))$	$T(n)$	=	$2^{\log2(n)} T(1) + k2 * \Sigma (2^j)$	[$j = 0$ até $(\log2(n) - 1)$]
$(\log2(n))$	$T(n)$	=	$n\, T(1) + k2 * \Sigma (2^j)$	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * [2^(log2(n)) - 1]	(soma dos termos PG)

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * [2^(log2(n)) - 1]	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * [2^(log2(n)) - 1]	
(log2(n))	T(n)	=	nk1 + k2 * [n - 1]	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * [2^(log2(n)) - 1]	
(log2(n))	T(n)	=	nk1 + k2 * [n - 1]	
(log2(n))	T(n)	=	nk1 + (n-1)k2	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * [2^(log2(n)) - 1]	
(log2(n))	T(n)	=	nk1 + k2 * [n - 1]	
(log2(n))	T(n)	=	nk1 + (n-1)k2	
(log2(n))	T(n)	=	nk1 + nk2 - k2	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * [2^(log2(n)) - 1]	
(log2(n))	T(n)	=	nk1 + k2 * [n - 1]	
(log2(n))	T(n)	=	nk1 + (n-1)k2	
(log2(n))	T(n)	=	nk1 + nk2 - k2	
(log2(n))	T(n)	=	(k1 + k2)n - k2	

Resolução de recorrências

(i)	T(n)	=	2^i T (n / 2^i) + k2 * Σ (2^j)	[j = 0 até (i - 1)]
…				
(log2(n))	T(n)	=	2^log2(n) T(1) + k2 * Σ (2^j)	[j = 0 até (log2(n) - 1)]
(log2(n))	T(n)	=	n T(1) + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * Σ (2^j)	
(log2(n))	T(n)	=	nk1 + k2 * [2^(log2(n)) - 1]	
(log2(n))	T(n)	=	nk1 + k2 * [n - 1]	
(log2(n))	T(n)	=	nk1 + (n-1)k2	
(log2(n))	T(n)	=	nk1 + nk2 - k2	
(log2(n))	T(n)	=	(k1 + k2)n - k2	
(log2(n))	T(n)	=	Θ(n)	

Resolução de recorrências (prova por indução)

Algoritmo max1 (a partir de um problema de tamanho n resolve recursivamente um problema de tamanho n - 1):

$$T(n) = \begin{cases} k1 & \text{(para } n = 1\text{)} \\ T(n-1) + k2 & \text{(para } n > 1\text{)} \end{cases}$$

Como usar prova por indução para resolver esta recorrência?

- Dada uma solução candidata, demonstrar se ela, de fato, está correta.

Resolução de recorrências

Recorrência: $T(n) = k1$ (para $n = 1$)

$= T(n - 1) + k2$ (para $n > 1$)

Solução: $T(n) = (n - 1)k2 + k1$

Resolução de recorrências

Recorrência:	$T(n)$	$=$	$k1$	(para $n = 1$)
		$=$	$T(n - 1) + k2$	(para $n > 1$)

Solução: $T(n) = (n - 1)k2 + k1$

Demonstração (caso base):

Resolução de recorrências

Recorrência: $T(n) = k1$ (para $n = 1$)

$= T(n - 1) + k2$ (para $n > 1$)

Solução: $T(n) = (n - 1)k2 + k1$

Demonstração (caso base):

$T(1)$

Resolução de recorrências

Recorrência: $T(n) = k1$ (para $n = 1$)

$= T(n - 1) + k2$ (para $n > 1$)

Solução: $T(n) = (n - 1)k2 + k1$

Demonstração (caso base):

$T(1) = (1 - 1)k2 + k1$

Resolução de recorrências

Recorrência: $T(n) = k1$ (para $n = 1$)

$= T(n - 1) + k2$ (para $n > 1$)

Solução: $T(n) = (n - 1)k2 + k1$

Demonstração (caso base):

$T(1) = (1 - 1)k2 + k1$

$T(1) = k1$

Resolução de recorrências

Recorrência: $T(n) = k1$ (para $n = 1$)

$= T(n - 1) + k2$ (para $n > 1$)

Solução: $T(n) = (n - 1)k2 + k1$

Demonstração (caso base):

$T(1) = (1 - 1)k2 + k1$

$T(1) = k1$ [mesmo valor dado pela definição da recorrência]

Resolução de recorrências

Recorrência:	$T(n)$	=	$k1$	(para $n = 1$)
		=	$T(n - 1) + k2$	(para $n > 1$)

Solução: $T(n) = (n - 1)k2 + k1$

Demonstração (caso base):

$T(1) = (1 - 1)k2 + k1$

$T(1) = k1$ [mesmo valor dado pela definição da recorrência]

Caso base OK!

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$ [a partir da definição da recorrência]

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$ [a partir da definição da recorrência]

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$ [a partir da definição da recorrência]

$T(n) = (n - 2)k2 + k1 + k2$ [uso da hipótese]

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$ [a partir da definição da recorrência]

$T(n) = (n - 2)k2 + k1 + k2$ [uso da hipótese]

$T(n) = k2[(n - 2) + 1] + k1$

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$ [a partir da definição da recorrência]

$T(n) = (n - 2)k2 + k1 + k2$ [uso da hipótese]

$T(n) = k2[(n - 2) + 1] + k1$

$T(n) = (n - 1)k2 + k1$

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$ [a partir da definição da recorrência]

$T(n) = (n - 2)k2 + k1 + k2$ [uso da hipótese]

$T(n) = k2[(n - 2) + 1] + k1$

$T(n) = (n - 1)k2 + k1$ [mesma solução que queremos provar]

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese (solução vale para n - 1): $T(n - 1) = (n - 2)k2 + k1$

Verificação do passo indutivo:

$T(n) = T(n - 1) + k2$ [a partir da definição da recorrência]

$T(n) = (n - 2)k2 + k1 + k2$ [uso da hipótese]

$T(n) = k2[(n - 2) + 1] + k1$

$T(n) = (n - 1)k2 + k1$ [mesma solução que queremos provar]

Passo indutivo OK!

Resolução de recorrências

- Prova por indução pode não parecer muito útil, pois “apenas” serve para demonstrar se uma dada solução candidata está ou não correta.
- Como obter uma solução candidata em primeiro lugar???

Resolução de recorrências

- Prova por indução pode não parecer muito útil, pois “apenas” serve para demonstrar se uma dada solução candidata está ou não correta.
- Como obter uma solução candidata em primeiro lugar???
- Prova por indução pode ser boa para provar limites!

Resolução de recorrências

- Ainda considerando o exemplo que acabamos de ver, seria possível usar prova por indução para mostrar que T(n) é limitado superiormente por uma função linear. Ou seja, demonstrar que:

  T(n) = T(n - 1) + k2 -----> T(n) = O(n)

- O que equivaleria a demonstrar que:

  T(n) <= cn

Resolução de recorrências

Demonstração (caso base):

T(1) <= c * 1

T(1) <= c

k1 <= c ?

Resolução de recorrências

Demonstração (caso base):

$T(1) <= c * 1$

$T(1) <= c$

$k1 <= c$? possível, desde que se escolha $c >= k1$.

Resolução de recorrências

Demonstração (caso base):

$T(1) <= c * 1$

$T(1) <= c$

$k1 <= c$? possível, desde que se escolha $c >= k1$.

Logo, caso base OK!

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese: $T(n - 1) <= c(n - 1)$

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese: $T(n - 1) <= c(n - 1)$

Verificação: $T(n) = T(n - 1) + k2$ [por definição]

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese: $T(n - 1) <= c(n - 1)$

Verificação: $T(n) = T(n - 1) + k2$ [por definição]

$T(n) = T(n - 1) + k2 <= c(n - 1) + k2$ [uso da hipótese]

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese:	$T(n - 1) <= c(n - 1)$	
Verificação:	$T(n) = T(n - 1) + k2$	[por definição]
	$T(n) = T(n - 1) + k2 <= c(n - 1) + k2$	[uso da hipótese]
	$T(n) <= cn - c + k2$	

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese:	$T(n - 1) <= c(n - 1)$	
Verificação:	$T(n) = T(n - 1) + k2$	[por definição]
	$T(n) = T(n - 1) + k2 <= c(n - 1) + k2$	[uso da hipótese]
	$T(n) <= cn - c + k2$	

Se $cn - c + k2 <= cn$ então, por transitividade, $T(n) <= cn$

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese: $T(n - 1) <= c(n - 1)$

Verificação: $T(n) = T(n - 1) + k2$ [por definição]

$T(n) = T(n - 1) + k2 <= c(n - 1) + k2$ [uso da hipótese]

$T(n) <= cn - c + k2$

Se $cn - c + k2 <= cn$ então, por transitividade, $T(n) <= cn$

É possível escolher c, tal que $cn - c + k2 <= cn$?

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese: $T(n - 1) <= c(n - 1)$

Verificação: $T(n) = T(n - 1) + k2$ [por definição]

$T(n) = T(n - 1) + k2 <= c(n - 1) + k2$ [uso da hipótese]

$T(n) <= cn - c + k2$

Se $cn - c + k2 <= cn$ então, por transitividade, $T(n) <= cn$

É possível escolher c, tal que $cn - c + k2 <= cn$?

Sim, basta tomar $c >= k2$.

Resolução de recorrências

Demonstração (passo indutivo):

Hipótese: $T(n - 1) <= c(n - 1)$

Verificação: $T(n) = T(n - 1) + k2$ [por definição]

$T(n) = T(n - 1) + k2 <= c(n - 1) + k2$ [uso da hipótese]

$T(n) <= cn - c + k2$

Se $cn - c + k2 <= cn$ então, por transitividade, $T(n) <= cn$

É possível escolher c, tal que $cn - c + k2 <= cn$?

Sim, basta tomar $c >= k2$. Passo indutivo OK!

Resolução de recorrências

Assim, para c >= k1 e c >= k2, a prova por indução nos mostra que

$$T(n) \le cn$$

e portanto

$$T(n) = O(n)$$

Resolução de recorrências (Teorema Mestre)

Outra maneira de se resolver recorrências é pela aplicação do Teorema Mestre, uma solução “genérica” para recorrências da forma:

$$T(n) = aT(n / b) + f(n)$$

onde

a:	quantidade de subproblemas resolvidos recursivamente
b:	fator de redução de um problema em novos subproblemas
$f(n)$:	custo das operações executadas na chamada do método recursivo que resolve um problema de tamanho n, sem incluir o custo das a chamadas recursivas em si.

Resolução de recorrências

"Essência" do teorema (explicação informal):

Compara-se a função $f(n)$ com a função $n^{(\log_b a)}$

Se ambas forem equivalentes, então $T(n) = \Theta(f(n) * \log_2(n))$

Caso contrário, a função dominante determina a complexidade de $T(n)$

Resolução de recorrências

“Essência” do teorema (explicação informal):

Compara-se a função $f(n)$ com a função $n^{\wedge}(\log_b a)$

Se ambas forem equivalentes, então $T(n) = \Theta(f(n) * \log_2(n))$

Caso contrário, a função dominante determina a complexidade de $T(n)$

E de onde surge este $n^{\wedge}(\log_b a)$???

- árvore de recursão com $\Theta(\log_b n)$ níveis e $\Theta(a^{\wedge}(\log_b n))$ [$= \Theta(n^{\wedge}(\log_b a))$] nós.

Resolução de recorrências

Para aplicação do Teorema Mestre, devemos verificar 3 possíveis casos:

1) $f(n) = O(\ n\text{^}(\log_b a - \epsilon)\ )$ [deve haver uma “folga” no limite superior]

   Neste caso, $T(n) = \Theta(n\text{^}\log_b a)$

2) $f(n) = \Theta(n\text{^}\log_b a)$ [$f(n)$ e $n\text{^}(\log_b a)$ são equivalentes]

   Neste caso, $T(n) = \Theta(f(n) * \log_2(n))$

Resolução de recorrências

Para aplicação do Teorema Mestre, devemos verificar 3 possíveis casos:

3) $f(n) = \Omega(n \hat{} (\log_b a + \epsilon))$ [deve haver uma “folga” no limite inferior]

E, além disso, $a * f(n / b) <= k * f(n)$, para uma constante $k < 1$

Neste caso, $T(n) = \Theta(f(n))$

Resolução de recorrências

Para aplicação do Teorema Mestre, devemos verificar 3 possíveis casos:

3) $f(n) = \Omega(n^{\wedge}(\log_b a + \epsilon))$ [deve haver uma “folga” no limite inferior]

E, além disso, a * f(n / b) <= k * f(n), para uma constante k < 1

Neste caso, $T(n) = \Theta(f(n))$

Não necessariamente um dos três casos será atendido. Se isto acontecer, não é possível determinar a complexidade de T(n) pela aplicação do teorema.

Resolução de recorrências

Exemplo 1: $T(n) = 2T(n / 2) + k2$ (algoritmo max2)

$a = 2, b = 2 \Longrightarrow n^\wedge\log_b a = n^\wedge\log_2 2 = n^1$

$f(n) = k2 = O(n^{1 - \epsilon})$ [$\epsilon = 0.1$ por exemplo] $\Longrightarrow$ caso 1 do teorema!

Logo, pelo caso 1 do Teorema Mestre:

$T(n) = \Theta(n^\wedge\log_b a) \Longrightarrow T(n) = \Theta(n)$

Resolução de recorrências

Exemplo 2: $T(n) = 3T(n / 4) + n$

$a = 3, b = 4 \Longrightarrow n\text{^}\log_b a = n\text{^}\log_4 3 = n^{0.79}$

$f(n) = n^1 = \Omega(n^{0.79 + \epsilon})$ [$\epsilon = 0.2$ por exemplo] $\Longrightarrow$ possível caso 3.

$a * f(n/b) <= k * f(n)$, para $k < 1$? $\Longrightarrow$ $3n/4 <= kn$, para algum $k < 1$? Sim!

Logo, pelo caso 3 do Teorema Mestre: $T(n) = \Theta(f(n)) \Longrightarrow T(n) = \Theta(n)$

Resolução de recorrências

Exemplo 3: $T(n) = 4T(n / 4) + n$

$a = 4,\ b = 4 \Rightarrow n\hat{}\log_b a = n\hat{}\log_4 4 = n^1$

$f(n) = n^1 = \Theta(n^1) = \Theta(n\hat{}\log_b a) \Rightarrow$ caso 2 do teorema!

Logo, pelo caso 2 do Teorema Mestre:

$T(n) = \Theta(\ (n\hat{}\log_b a) * \log_2 n\ ) \Rightarrow T(n) = \Theta(n\log_2 n)$

Resolução de recorrências

Exercício: suponha uma nova variante do método max2 que, ao invés de representar os subproblemas (subvetores) através dos índices ini e fim, crie dois novos vetores (cada um com metade do tamanho do vetor recebido pela chamada atual) e copie neles os valores presentes no vetor recebido na chamada atual. A recorrência, neste cenário, passaria a ser:

$T(n) = 2T(n / 2) + k2 + k3n$ [k3n refere-se à cópia dos n valores]

$a = 2, b = 2 \Longrightarrow n^{\wedge}\log_b a = n^1$

$f(n) = k3n + k2 = \Theta(n^1) = \Theta(n^{\wedge}\log_b a) \Longrightarrow$ caso 2 do teorema!

Logo pelo caso 2 do Teorema Mestre: $T(n) = \Theta(n * \log_2 n)$