Análise de Algoritmos

Prof. Lilian Berton

São José dos Campos, 2018

Baseado no material de Antonio Alfredo Ferreira Loureiro
http://www.dcc.ufmg.br/~loureiro

Análise do tempo de execução

- **Procedimentos não recursivos:**
- Cada um deve ser computado separadamente, iniciando com os que não chamam outros procedimentos.
- Avalia-se então os que chamam os já avaliados (utilizando os tempos desses).
- O processo é repetido até chegar no programa principal.

- **Procedimentos recursivos:**
- É associada uma função de complexidade f(n) desconhecida, onde n mede o tamanho dos argumentos.

Algoritmos recursivos

- **Um objeto é recursivo quando é definido parcialmente em termos de si mesmo.**

- Exemplo: Função fatorial

  (a) $0! = 1$

  (b) se $n > 0$ então $n! = n \cdot (n - 1)!$

- Um problema recursivo P pode ser expresso como P ≡ P[Si , P], onde P é a composição de comandos Si e do próprio P.

- Importante: constantes e variáveis locais a P são duplicadas a cada chamada recursiva.

Problema de terminação

- **Definir uma condição de terminação.**

- Associar um parâmetro, por exemplo n, com P e chamar P recursivamente com n − 1 como parâmetro.
- A condição n > 0 garante a terminação.
  - Exemplo: P(n) ≡ if n > 0 then P[Si ; P(n − 1)].

- Importante: na prática é necessário mostrar que o nível de recursão é finito, e tem que ser mantido pequeno!

Razões para limitar a recursão

- Memória necessária para acomodar variáveis a cada chamada.
- O estado corrente da computação tem que ser armazenado para permitir a volta da chamada recursiva.

```
function F(i : integer)
begin
  if i > 0
  then F := i * F(i-1)
  else F := 1;
end;
```

F(4) →	1	4∗F(3)
	2	3∗F(2)
	3	2∗F(1)
	4	1∗F(0)
	1	

Quando não usar recursividade

- Algoritmos recursivos são apropriados quando o problema é definido em termos recursivos.

- Entretanto, uma definição recursiva não implica necessariamente que a implementação recursiva é a melhor solução!

- Casos onde evitar recursividade:
- P ≡ if condição then (Si ; P)
- Exemplo: P ≡ if i < n then (**i = i + 1; F = i ∗ F**; P)

Eliminando a recursão de cauda

```
function Fat : integer;
var F, i : integer;
begin
  i := 0; F := 1;
  while i < n do
  begin
    i := i+1;
    F := F*i;
  end;
  Fat := F;
end
```

- Logo, P ≡ if B then (S; P) deve ser transformado em:
- P ≡ (x = x0; while B do S)

Análise de algoritmos recursivos

- Comportamento é descrito por uma equação de recorrência.
- **Usar a própria recorrência para substituir para T(m), m < n até que todos os termos tenham sido substituídos por fórmulas envolvendo apenas T(0) ou o caso base.**

- Seja a seguinte equação de recorrência para a função fatorial:

$$T(n) = \begin{cases} d & n = 1 \\ c + T(n-1) & n > 1 \end{cases}$$

- Esta equação diz que quando n = 1 o custo para executar fat é igual a d.
- Para valores de n maiores que 1, o custo para executar fat é c mais o custo para executar T(n − 1).

Resolvendo equação de recorrência

$$
\begin{aligned}
T(n) &= c + T(n-1) \\
&= c + (c + T(n-2)) \\
&= c + c + (c + T(n-3)) \\
\vdots &= \vdots \\
&= c + c + \cdots + (c + T(1)) \\
&= \underbrace{c + c + \cdots + c}_{n-1} + d
\end{aligned}
$$

- Em cada passo, o valor do termo T é substituído pela sua definição (ou seja, esta recorrência está sendo resolvida pelo método da expansão).
- A última equação mostra que depois da expansão existem n − 1 c's, correspondentes aos valores de 2 até n.
- Desta forma, a recorrência pode ser expressa como: T(n) = c(n − 1) + d = O(n)

Exemplo 2 de recorrência

$$\begin{aligned} T(n) &= T(\tfrac{n}{2}) + 1 \quad (n \geq 2) \\ T(1) &= 0 \qquad\qquad\quad (n = 1) \end{aligned}$$

Vamos supor que:

$$n = 2^k \Rightarrow k = \log n$$

Resolvendo por expansão temos:

$$\begin{aligned} T(2^k) &= T(2^{k-1}) + 1 \\ &= (T(2^{k-2}) + 1) + 1 \\ &= (T(2^{k-3}) + 1) + 1 + 1 \\ &\vdots \quad\quad \vdots \\ &= (T(2) + 1) + 1 + \cdots + 1 \\ &= (T(1) + 1) + 1 + \cdots + 1 \\ &= 0 + \underbrace{1 + \cdots + 1}_{k} \\ &= k \end{aligned}$$

$$\begin{aligned} T(n) &= \log n \\ T(n) &= O(\log n) \end{aligned}$$

Indução matemática e algoritmos

- É útil para provar asserções sobre a correção e a eficiência de algoritmos.

- Consiste em inferir uma lei geral a partir de instâncias particulares.

- Seja T um teorema que tenha como parâmetro um número natural n.
- Para provar que T é válido para todos os valores de n, provamos que:
- **1. T é válido para n = 1;** **[PASSO BASE]**
- **2. Para todo n > 1,** **[PASSO INDUTIVO]**
  - **se T é válido para n,**
  - **então T é válido para n + 1.**

- Provar a condição 2 é geralmente mais fácil que provar o teorema diretamente (podemos usar a asserção de que T é válido para n).
- As condições 1 e 2 implicam T válido para n = 2, o que junto com a condição 2 implica T também válido para n = 3, e assim por diante.

Limite superior de equações de recorrência

- A solução de uma equação de recorrência pode ser difícil de ser obtida.
- Nesses casos, pode ser mais fácil tentar adivinhar a solução ou obter um limite superior para a ordem de complexidade.

- **Adivinhar a solução funciona bem quando estamos interessados apenas em um limite superior, ao invés da solução exata.**

- Mostrar que um certo limite existe é mais fácil do que obter o limite.
- Por exemplo:
- $T(2n) \leq 2T(n) + 2n - 1$,
- $T(2) = 1$,
- definida para valores de n que são potências de 2.

- **O objetivo é encontrar um limite superior na notação O, onde o lado direito da desigualdade representa o pior caso.**

Indução matemática para resolver equação de recorrência

$$\begin{aligned} T(2) &= 1, \\ T(2n) &\leq 2T(n) + 2n - 1, \end{aligned}$$

definida para valores de n que são potências de 2.

- Procuramos $f(n)$ tal que $T(n) = O(f(n))$, mas fazendo com que $f(n)$ seja o mais próximo possível da solução real para $T(n)$ (limite assintótico firme).
- Vamos considerar o palpite $f(n) = n^2$.
- Queremos provar que

$$T(n) \leq f(n) = O(f(n))$$

utilizando indução matemática em n.

Indução matemática para resolver equação de recorrência

Prove que $T(n) \leq f(n) = O(f(n))$, para $f(n) = n^2$, sendo

$$\begin{aligned} T(2) &= 1, \\ T(2n) &\leq 2T(n) + 2n - 1, \end{aligned}$$

definida para valores de n que são potências de 2.

Prova (por indução matemática):

1. Passo base:
   $T(n_0) = T(2)$: Para $n_0 = 2$, $T(2) = 1 \leq f(2) = 4$, e o passo base é V.

2. Passo indutivo: se a recorrência é verdadeira para n então deve ser verdadeira para $2n$, i.e., $T(n) \rightarrow T(2n)$ (lembre-se que n é uma potência de 2; conseqüentemente o "número seguinte" a n é $2n$).

   Reescrevendo o passo indutivo temos:

$$\begin{aligned} \text{Predicado}(n) &\rightarrow \text{Predicado}(2n) \\ (T(n) \leq f(n)) &\rightarrow (T(2n) \leq f(2n)) \end{aligned}$$

$$\begin{aligned} T(2n) &\leq 2T(n) + 2n - 1 && \text{[Definição da recorrência]} \\ &\leq 2n^2 + 2n - 1 && \text{[Pela hipótese indutiva podemos substituir } T(n)\text{]} \\ &\leq 2n^2 + 2n - 1 \overset{?}{<} (2n)^2 && \text{[A conclusão é verdadeira?]} \\ &\leq 2n^2 + 2n - 1 < 4n^2 && \text{[Sim!]} \end{aligned}$$

   Essa última inequação é o que queremos provar. Logo, $T(n) = O(n^2)$.

Indução matemática para resolver equação de recorrência

Prove que $T(n) \leq f(n) = O(f(n))$, para $f(n) = cn$, sendo

$$\begin{aligned} T(2) &= 1, \\ T(2n) &\leq 2T(n) + 2n - 1, \end{aligned}$$

definida para valores de n que são potências de 2.

Prova (por indução matemática):

1. Passo base:
   $T(n_0) = T(2)$: Para $n_0 = 2, T(2) = 1 \leq f(2) = 2c$, e o passo base é V.

2. Passo indutivo: se a recorrência é verdadeira para n então deve ser verdadeira para $2n$, i.e., $T(n) \rightarrow T(2n)$.

   Reescrevendo o passo indutivo temos:

$$\begin{aligned} \text{Predicado}(n) &\rightarrow \text{Predicado}(2n) \\ (T(n) \leq f(n)) &\rightarrow (T(2n) \leq f(2n)) \\ (T(n) \leq cn)) &\rightarrow (T(2n) \leq 2cn) \end{aligned}$$

$$\begin{aligned} T(2n) &\leq 2T(n) + 2n - 1 && \text{[Definição da recorrência]} \\ &\leq 2cn + 2n - 1 && \text{[Pela hipótese indutiva podemos substituir } T(n)\text{]} \\ &\leq 2cn + (2n - 1) \\ &\leq 2cn + 2n - 1 > 2cn && \text{[A conclusão } (T(2n) \leq 2cn) \text{ não é válida]} \end{aligned}$$

Indução matemática para resolver equação de recorrência

Prove que $T(n) \leq f(n) = O(f(n))$, para $f(n) = n \log n$, sendo

$$\begin{aligned} T(2) &= 1, \\ T(2n) &\leq 2T(n) + 2n - 1, \end{aligned}$$

definida para valores de n que são potências de 2.

Prova (por indução matemática):

1. Passo base:
   $T(n_0) = T(2)$: Para $n_0 = 2$, $T(2) = 1 \leq f(2) = 2 \log 2$, e o passo base é V.

2. Passo indutivo: se a recorrência é verdadeira para n então deve ser verdadeira para $2n$, i.e., $T(n) \rightarrow T(2n)$.

Reescrevendo o passo indutivo temos:

$$\begin{aligned} \text{Predicado}(n) &\rightarrow \text{Predicado}(2n) \\ (T(n) \leq f(n)) &\rightarrow (T(2n) \leq f(2n)) \\ (T(n) \leq n \log n)) &\rightarrow (T(2n) \leq 2n \log 2n) \end{aligned}$$

$$\begin{aligned} T(2n) &\leq 2T(n) + 2n - 1 && \text{[Definição da recorrência]} \\ &\leq 2n \log n + 2n - 1 && \text{[Podemos substituir } T(n)\text{]} \\ &\leq 2n \log n + 2n - 1 \overset{?}{<} 2n \log 2n && \text{[A conclusão é verdadeira?]} \\ &\leq 2n \log n + 2n - 1 < 2n \log n + 2n && \text{[Sim!]} \end{aligned}$$

Considerações sobre indução

- Indução é uma das técnicas mais poderosas da Matemática que pode ser aplicada para provar asserções sobre a correção e a eficiência de algoritmos.

- No caso de correção de algoritmos, é comum tentarmos identificar invariantes para laços.

- Indução pode ser usada para derivar um limite superior para uma equação de recorrência.

Teorema Mestre

- Recorrências da forma:

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

- onde $a \geq 1$ e $b > 1$ são constantes e f(n) é uma função assintoticamente positiva podem ser resolvidas usando o Teorema Mestre.

- Note que neste caso não estamos achando a forma fechada da recorrência mas sim seu comportamento assintótico.

Teorema Mestre

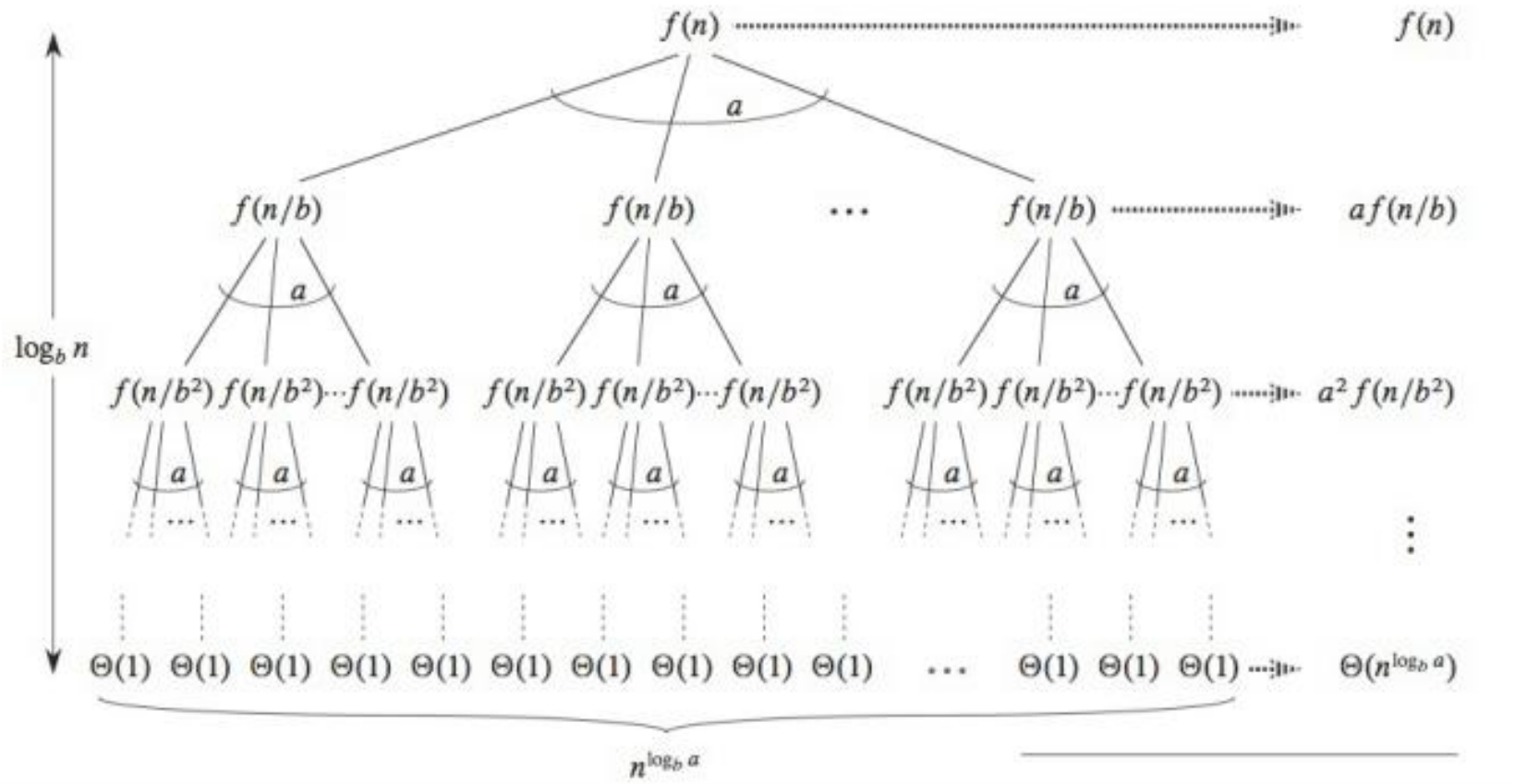
Sejam as constantes $a \geq 1$ e $b > 1$ e $f(n)$ uma função definida nos inteiros não-negativos pela recorrência:

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

onde a fração n/b pode significar $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$. A equação de recorrência $T(n)$ pode ser limitada assintoticamente da seguinte forma:

1. Se $f(n) = O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $\boxed{T(n) = \Theta(n^{\log_b a})}$.
2. Se $f(n) = \Theta(n^{\log_b a})$, então $\boxed{T(n) = \Theta(n^{\log_b a} \log n)}$.
3. Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguma constante $\epsilon > 0$ e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e para n suficientemente grande, então $\boxed{T(n) = \Theta(f(n))}$.

$n^{\log_b a}$ é o número de folhas da árvore de recursão a-ária gerada por T(n) = aT(n/b) + f(n).



Teorema Mestre

Nos três casos estamos comparando a função $f(n)$ com a função $n^{\log_b a}$. Intuitivamente, a solução da recorrência é determinada pela maior das duas funções.

Por exemplo:

- No primeiro caso a função $n^{\log_b a}$ é a maior e a solução para a recorrência é $T(n) = \Theta(n^{\log_b a})$.
- No terceiro caso, a função $f(n)$ é a maior e a solução para a recorrência é $T(n) = \Theta(f(n))$.
- No segundo caso, as duas funções são do mesmo "tamanho." Neste caso, a solução fica multiplicada por um fator logarítmico e fica da forma $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(f(n) \log n)$.

Teorema Mestre

- Teorema não cobre todas as possibilidades para f(n):
  - **Entre os casos 1 e 2 existem funções f(n) que são menores que $n^{\log_b a}$ mas não são polinomialmente menores.**
  - **Entre os casos 2 e 3 existem funções f(n) que são maiores que $n^{\log_b a}$ mas não são polinomialmente maiores.**

Uma função $f(n)$ é **polinomialmente maior** que outra função $g(n)$ se pudermos achar algum $\epsilon > 0$ tal que $\frac{f(n)}{g(n)} = n^{\epsilon}$.

Exemplo: n^2 é polinomialmente maior que n. No entanto, $nlogn$ e $2n$ não são polinomialmente maiores que n.

Uma função $f(n)$ é **polinomialmente menor** que outra função $g(n)$ se pudermos achar algum $\epsilon > 0$ tal que $\frac{g(n)}{f(n)} = n^{\epsilon}$.

- Se a função f(n) cai numa dessas condições ou a condição de regularidade do caso 3 é falsa, então não se pode aplicar este teorema para resolver a recorrência.

Exemplo 1 Teorema Mestre

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

Temos que,

$$a = 9, b = 3, f(n) = n$$

Desta forma,

$$n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$$

Como $f(n) = O(n^{\log_3 9 - \epsilon})$, onde $\epsilon = 1$, podemos aplicar o caso 1 do teorema e concluir que a solução da recorrência é

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

Exemplo 2 Teorema Mestre

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

Temos que,

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

Desta forma,

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

O caso 2 se aplica já que $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$. Temos, então, que a solução da recorrência é

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

Exemplo 3 Teorema Mestre

$$T(n) = 3T(n/4) + n \log n$$

Temos que,

$$a = 3, b = 4, f(n) = n \log n$$

Desta forma,

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

Como $f(n) = \Omega(n^{\log_4 3 + \epsilon})$, onde $\epsilon \approx 0.2$, o caso 3 se aplica se mostrarmos que a condição de regularidade é verdadeira para $f(n)$.

Para um valor suficientemente grande de n

$$af(n/b) = 3(n/4) \log(n/4) \leq (3/4)n \log n = cf(n)$$

para $c = 3/4$. Conseqüentemente, usando o caso 3, a solução para a recorrência é

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

Exemplo 4 Teorema Mestre

$$T(n) = 2T(n/2) + n \log n$$

Temos que,

$$a = 2, b = 2, f(n) = n \log n$$

Desta forma,

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

Aparentemente o caso 3 deveria se aplicar já que $f(n) = n \log n$ é assintoticamente maior que $n^{\log_b a} = n$. Mas no entanto, não é polinomialmente maior. A fração $f(n)/n^{\log_b a} = (n \log n)/n = \log n$ que é assintoticamente menor que n^{ϵ} para toda constante positiva ϵ. Conseqüentemente, a recorrência cai na situação entre os casos 2 e 3 onde o teorema não pode ser aplicado.

Exercícios

1. Aplique o teorema mestre nas seguintes equações de recorrência:

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

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

   $T(n) = 4T(n/2) + n^3$

2. O Teorema Mestre pode ser aplicado a recorrência $T(n) = 4T(n/2) + n^2 \log n$? Justifique a sua resposta.

3. Use indução para resolver a seguinte recorrência:

   $T(n) = 2 + T(n)$

   $T(1) = 1$

Pense sozinho, pesquise, investigue, leia...

Se você não o fizer, alguém o fará em seu lugar, ensinando o que pensar, dizer e até mesmo sentir. Lembre-se de que educar não é encher a mente, mas libertá-la de seus vínculos e que, muitas vezes, o aprendizado mais duradouro e profundo são aqueles que fazemos sozinhos.