Complexidade de Algoritmos (parte 3)

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados II (AE23CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Sumário

- Recorrência
- Resolução de Recorrências
  - Método da substituição
  - Método da iteração
  - Método da árvore de recursão
  - Método mestre

Introdução

- Na aula anterior foi abordada a análise de complexidade de algoritmos
  - No entanto, analisamos apenas algoritmos iterativos

- Como é feita a análise de complexidade de algoritmos recursivos?

```
sub_rotina fatorial(n:  numérico)
Início
  declare aux numérico;
  se n = 1
    então aux ← 1;
    senão aux ← n * fatorial(n - 1);
  fatorial ← aux;
Fim
```

Introdução

- O exemplo apresentado anteriormente é obviamente $O(n)$

- Se a recursão é um "disfarce" da repetição (geralmente uma má aplicação da recursão), basta analisá-la como tal

```
sub_rotina fatorial(n:  numérico)
Início
  declare aux numérico;
  declare i numérico;
  aux ← 1;
  para i ← 0 até n faça
    aux ← aux + i * aux;
  fatorial ← aux;
Fim
```

Introdução

- Em muitos casos (até para recursividade mal empregada), é difícil transformar a sub-rotina em iteração
  - Pode ser necessário a análise de recorrência para analisar o desempenho do algoritmo

Sumário

Recorrência

- Função matemática que descreve sequências/conjuntos em termos de seu valor em entradas anteriores
  - Comumente aplicada para a análise de algoritmos de divisão-e-conquista

- Análise de recorrência
  - Caso base (1): tempo de execução constante (têm casos em que não é necessário)
  - Caso indutivo (2): quando o teste para o caso base falha, ou seja, o algoritmo não é executado em tempo constante

- Uma recorrência pode ter mais de um caso base e/ou caso indutivo

Recorrência

- Exemplo de uso de recorrência:
  - Números de Fibonacci
    - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
    - fib(0) = 0, fib(1) = 1, fib(n) = fib(n - 1) + fib(n - 2)

```
sub_rotina fib(n:  numérico)
Início
  declare aux numérico;
  se n = 0
    então aux ← 0
    senão se n = 1
      então aux ← 1
      senão aux ← fib(n - 1) + fib(n - 2);
  fib ← aux
Fim
```

Recorrência

- Seja T(n) o tempo de execução da função
  - Caso 1 (base): o tempo de execução é constante
    - se $n = 0$ o custo é para comparar o valor n + atribuir o valor 1 para aux + atribuir o valor de aux ao nome da função (retorno), ou seja, $T(0) = 3$
    - $n = 1$, o custo é para falhar na comparação do valor n com zero + comparar n com 1 + atribuir o valor 1 para aux + atribuir o valor de aux ao nome da função (retorno), ou seja, $T(1) = 4$
  - Caso 2 (indução): se $n > 1$, quando falha a segunda comparação (custo 2 até o momento), é feita uma atribuição + uma soma + duas chamadas recursivas + duas subtrações + uma atribuição o valor de aux ao nome da função, o que resulta na recorrência $T(n) = T(n-1) + T(n-2) + 7$

Recorrência

- O problema recursivo apresentado pode ser representado na seguinte forma:

$$T(n) = \begin{cases} 3, & se\ n = 0 \\ 4, & se\ n = 1 \\ T(n-1) + T(n-2) + 7, & se\ n > 1 \end{cases}$$

Sumário

Resolução de Recorrências

- Diversas técnicas podem ser aplicadas para a resolução de recorrências
  - Método da substituição
  - Método da iteração
  - Método mestre
  - Método da árvore de recursão

Resolução de Recorrências

Método da substituição

- Propõe chutar uma fórmula para recorrência
- Após, chute deve ser verificado se funciona para o caso base (substituir o valor de n para o valor que acarreta no caso base)
- Em seguida, é demonstrado que a fórmula está correta através da indução matemática
- Formas de fazer bons chutes
  - Ter "boa base" matemática (experiência)
  - Usar árvores de recursão (veremos mais adiante)
  - Utilizar o resultado de outras formas de análise de recorrência (*e.g.* método da iteração)

- Exemplo

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ T(n-1) + 3n + 2, & \text{se } n > 1 \end{cases}$$

  - Chute: $T(n) = \frac{3n^2+7n}{2} - 4$

- Exemplo

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ T(n-1) + 3n + 2, & \text{se } n > 1 \end{cases}$$

  - Chute: $T(n) = \frac{3n^2+7n}{2} - 4$
  - Resolução:

Caso base ($n = 1$): $T(1) = \frac{3*1^2+7*1}{2} - 4 = 5 - 4 = 1$

Indução: suponha que o chute vale para $n - 1$
$T(n-1) = \frac{3(n-1)^2+7(n-1)}{2} - 4$

Temos:
$T(n) = T(n-1) + 3n + 2 = \frac{3(n-1)^2+7(n-1)}{2} - 4 + 3n + 2$
$= \frac{3n^2-6n+3+7n-7}{2} + 3n - 2 = \frac{3n^2+n-4}{2} + 3n - 2$
$= \frac{3n^2+7n}{2} - 4$, ou seja, a complexidade é $O(n^2)$

- É uma forma do método da substituição onde a recorrência é expandida (iterada) e expressada como um somatório em termos de n

- Exemplo:

$$T(n) = \begin{cases} 1, & se\ n = 0 \\ T(n-1) + n, & se\ n > 0 \end{cases}$$

- É uma forma do método da substituição onde a recorrência é expandida (iterada) e expressada como um somatório em termos de n

- Exemplo:

$$T(n) = \begin{cases} 1, & se\ n = 0 \\ T(n-1) + n, & se\ n > 0 \end{cases}$$

Resolução:
Expandindo a recorrência, temos $T(n) = T(n-1) + n$
$= T(n-1-1) + n + (n-1)$
$= T(n-2-1) + n + (n-1) + (n-2)$
$= T(n-3-1) + n + (n-1) + (n-2) + (n-3)$
$= T(n-k) + \sum_{i=0}^{k-1} (n-i)$

Substituindo k por n, temos:

$T(n) = T(0) + \sum_{i=0}^{n-1} (n-i)$

$T(n) = 1 + \frac{n(n+1)}{2}$, ou seja, a complexidade é $O(n^2)$

- Exemplo 2:

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

-> Observe que a recorrência acima não possui um valor inicial

-> Neste caso, podemos definir o valor inicial de acordo com a nossa preferência

-> Para este tipo de recorrência, geralmente, o interesse é a obtenção do limite assintótico

- Exemplo 2:

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

Resolução

$T(n) = T(n/2) + 1$
$= T(n/4) + 1 + 1$
$= T(n/8) + 1 + 1 + 1$
$= T(n/16) + 1 + 1 + 1 + 1$
...
$= T(n/2^k) + k$

-> Considerando que n seja potência de 2 e que a iteração segue até $T(1)$, temos $n = 2^k$, ou seja, $k = \log n$

-> Assumindo que $T(1) = 0$, temos:
$T(n) = T(n/2^{\log n}) + \log n = T(n/n) + \log n = T(1) + \log n = 0 + \log n$
$T(n) = \log n$, ou seja, a complexidade é $O(\log n)$

- Exemplo 2:

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

Resolução (2)

$T(n) = T(n/2) + 1$
$= T(n/4) + 1 + 1$
$= T(n/8) + 1 + 1 + 1$
$= T(n/16) + 1 + 1 + 1 + 1$
...
$= T(n/k) + \log k$

-> Considerando que $k = n$ e que a iteração segue até $T(1) = 0$ temos:
$T(n) = T(n/n) + \log n = T(1) + \log n = 0 + \log n$
$T(n) = \log n$, ou seja, a complexidade é $O(\log n)$

- Possibilita a definição de complexidades assintóticas para recorrências que apresentam as seguintes condições:
  - $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
  - $a \geq 1$ e $b > 1$
- Caso a recorrência atenda essas condições, as seguintes situações devem ser analisadas:
  1. Se $f(n) \in O(n^{log_b a - \epsilon})$ para a constante $\epsilon > 0$, então $T(n) \in \Theta(n^{\log_b a})$
  2. Se $f(n) \in \Theta(n^{log_b a})$, então $T(n) \in \Theta(n^{\log_b a} \log_2 n)$
  3. Se $f(n) \in \Omega(n^{\log_b a + \epsilon})$ para a constante $\epsilon > 0$, e se $af\left(\frac{n}{b}\right) \leq cf(n)$, para a constante $c < 1$ e n suficientemente grande, então $T(n) \in \Theta(f(n))$

- Exemplo 1:

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

- Exemplo 1:

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

Resolução:
$a = 9$
$b = 3$
$f(n) = n$
$\log_b a = \log_3 9 = 2$
$f(n) \in O(n^{\log_3 9 - \epsilon})$, para $\epsilon = 1$ (**situação 1**)
$n \in O(n)$

Então, $T(n) \in \Theta(n^{\log_b a})$, ou seja $\Theta(n^{log_3 9}) = \Theta(n^2)$

Se o objetivo é obter a complexidade no pior caso, o termo acima pode ser representado em termos de *big-oh*: $O(n^2)$

- Exemplo 2:

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

- Exemplo 2:

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

Resolução:
$a = 1$
$b = 3/2$
$f(n) = 1$
$\log_b a = \log_{3/2} 1 = 0$
$f(n) \in \Theta(n^0)$ (**situação 2**)
$1 \in \Theta(1)$

Então, $T(n) \in \Theta(n^{\log_b a} \log_2 n)$, ou seja
$\Theta(n^{\log_{3/2} 1} \log_2 n) = \Theta(\log_2 n)$

- Exemplo 3:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

- Exemplo 3:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

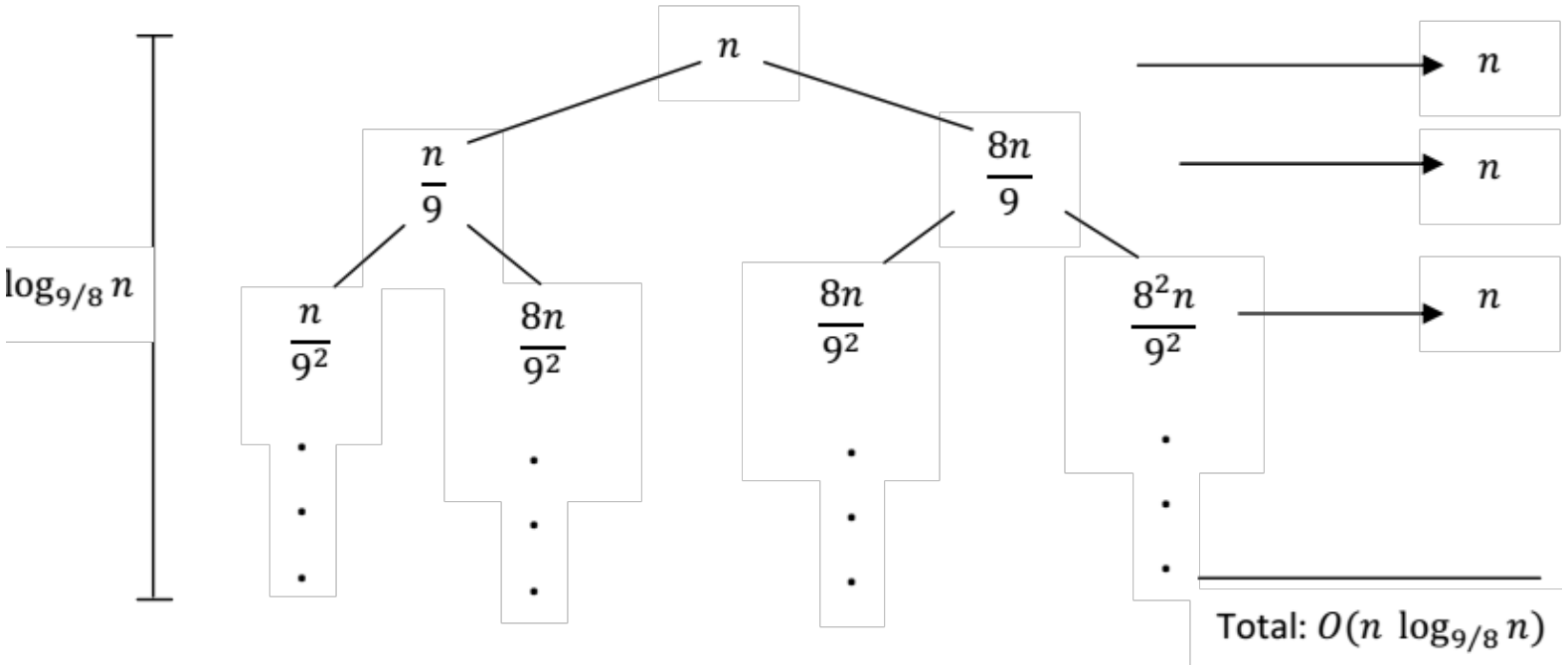
Resolução:
$a = 2$
$b = 2$
$f(n) = n^2$
$\log_b(a) = \log_2 2 = 1$
$f(n) \in \Omega(n^{\log_2 2+\epsilon})$, para $\epsilon = 1$ (**situação 3**)
$n^2 \in \Omega(n^2)$
$af\left(\frac{n}{b}\right) = 2\left(\frac{n}{2}\right)^2 = \frac{n^2}{2}$
$\frac{n^2}{2} \leq cf(n)$, para $c = \frac{1}{2}$

Então, $T(n) \in \Theta(f(n))$, ou seja $\Theta(n^2)$

- Similar ao método da iteração
- Permite visualizar melhor os termos da recorrência à medida que eles são expandidos (mais amigável)
- Nesse método, uma árvore é desenvolvida nível a nível, representando as recursões
- Em cada nível/nó da árvore são acumulados os tempos necessários para o processamento
- No final, a estimativa do tempo do problema é obtida

- Exemplo:

$$T(n) = T\left(\frac{n}{9}\right) + T\left(\frac{8n}{9}\right) + n$$



Considerações Finais

- Existem recorrências que não se conhecem métodos de resolução

- Apesar das recorrências serem um tema da disciplina "Matemática Discreta para Engenharia de Computação" (MD24CP), esse assunto é de grande importância para a análise de alguns algoritmos que serão vistos nas próximas aulas

Referências I


Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Clifford, S.
Algoritmos: teoria e prática.
Elsevier, 2012.


Horowitz, E., Sahni, S. Rajasekaran, S.
Computer Algorithms.
Computer Science Press, 1998.


Rosa, J. L. G.
Análise de Algoritmos - parte 1. SCC-201 – Introdução à Ciência da Computação II.
Slides. Ciência de Computação. ICMC/USP, 2016.


Ziviani, N.
Projeto de Algoritmos - com implementações em Java e C++.
Thomson, 2007.