Universidade Federal de Goiás – Instituto de Informática

Análise e Projeto de Algoritmos
Recursividade e Solução de Recorrências

Diogo Stelle

diogostelle@inf.ufg.br / diogo.stelle@gmail.com

2019

Algoritmos Recursivos

- Definição
  - Um algoritmo pode ser composto por funções, que, por sua vez, podem invocar outras funções.

    Quando uma função invoca a si própria, a denominamos **função recursiva**.

    É um conceito poderoso, pois define sucintamente conjuntos infinitos de instruções finitas.

    A ideia é aproveitar a solução de um ou mais subproblemas com estrutura semelhante para resolver o problema original.

Algoritmos Recursivos

- Exemplo
  - Um exemplo clássico de algoritmo recursivo é o cálculo de fatorial:
    - 0! = 1! = 1
    - n! = n × (n − 1)!

```
long fatorial(int n) {
    if( n <= 1)
        return 1;
    return n * fatorial(n - 1);
}
```

Algoritmos Recursivos

- Projeto
  - Um algoritmo recursivo é composto, em sua forma mais simples, de uma **condição de parada** e de um **passo recursivo**.
- Passo Recursivo
  - Realiza as chamadas recursivas e processa os diferentes valores de retorno, quando adequado. A ideia é associar um parâmetro n e realizar o passo recursivo sobre $n - 1$ (ou outra fração de n).
- Condição de Parada
  - Garante que a recursividade é finita, geralmente, definida sobre um **caso base**. Por exemplo, a condição $n \leq 1$ do exemplo anterior garante a parada com n positivo (considerando o decremento unitário).

Algoritmos Recursivos

- Análise de Complexidade do Fatorial Recursivo
  - Seja $T(n)$ a complexidade de tempo do fatorial recursivo:
    - Caso Base: $T(1) = a$;
    - Passo Recursivo: $T(n) = T(n - 1) + b$, $n > 1$.

Algoritmos Recursivos

- Análise de Complexidade do Fatorial Recursivo
  - Calculando...
    - $T(2) = T(1) + b = a + b$
    - $T(3) = T(2) + b = a + 2b$
    - $T(4) = T(3) + b = a + 3b$
    - Generalizando, temos que a **forma fechada** para esta recorrência é $T(n) = a + (n - 1)b$.
    - Logo, sendo a e b constantes, $T(n) = O(n)$.

Algoritmos Recursivos

- Observações
    - Todo algoritmo recursivo possui uma versão iterativa equivalente, bastando utilizar uma pilha explícita.
    - Se um problema é definido em termos recursivos, a implementação via algoritmos recursivos é facilitada.
    - Entretanto, isto não quer dizer que esta será a melhor solução.
    - É necessário estar atento ao fator de **ramificação da recursão**, ou seja, quantas chamadas recursivas serão feitas por vez.
    - Erros de implementação fatalmente geram loop infinito ou estouro de pilha

Algoritmos Recursivos

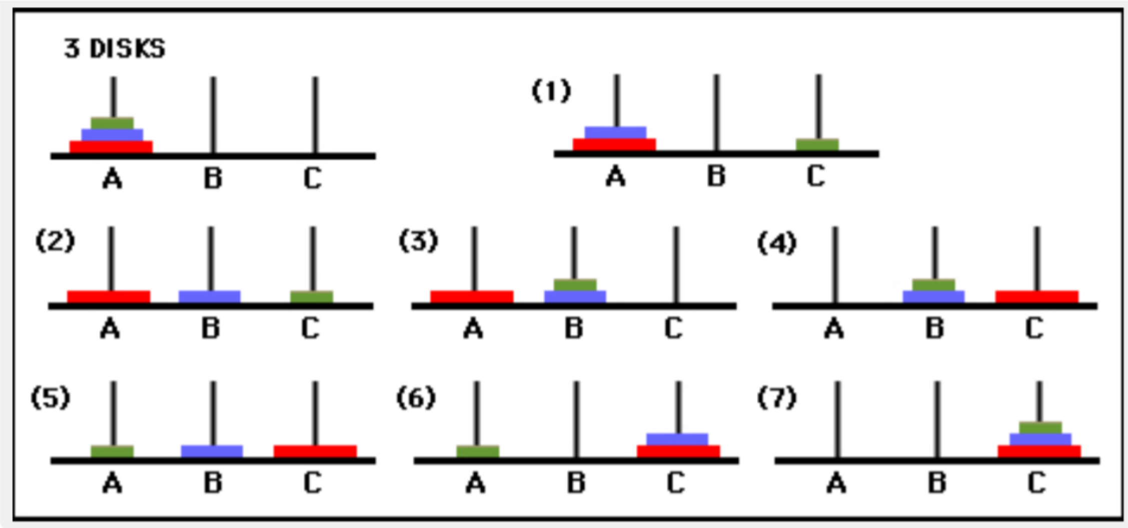
- Torres de Hanói
    - Na versão clássica, é necessário mover n discos de diâmetros diferentes, um de cada vez, de uma torre de origem para a torre de destino, usando ainda uma torre auxiliar. Não é permitido posicionar um disco maior sobre outro menor.

Algoritmos Recursivos

- Método de Solução
  - Seja $T(n)$ o número mínimo de movimentos necessários para mover todos os n discos para a torre de destino de acordo com o enunciado do problema.
  - Por inspeção, temos que:
    - $T(0) = 0$
    - $T(1) = 1$
    - $T(2) = 3$
    - $T(3) = 7$

Algoritmos Recursivos

- Método de Solução
  - Método de Solução n = 3, destino = C



Algoritmos Recursivos

- Método de Solução Generalizado
  - Para n discos:
    - Mova os n-1 discos do topo da origem para auxiliar (T(n – 1) movimentos);
    - Mova o maior disco da origem para o destino (1 movimento);
    - Mova os n-1 discos de auxiliar para o destino (T(n – 1) movimentos).
- Insight
  - Divida o problema original sucessivamente em subproblemas de tamanho n-1 e resolva recursivamente para tamanhos de n crescentes.

Algoritmos Recursivos

```
void hanoi(int n, int origem, int destino, int aux) {
    if(n==1)
        printf("\nMova de %d para %d", origem, destino);
    hanoi(n-1, origem, aux, destino);
    printf("\nMova de %d para %d", origem, destino);
    hanoi(n-1, aux, destino, origem);
}
```

- Complexidade de Tempo
  - Caso Base: $T(1) = 1$
  - Passo Recursivo: $T(n) = 2T(n - 1) + 1, n > 1$

Algoritmos Recursivos

- Calculando $T(n) = 2T(n - 1) + 1, n \geq 1$
  - $T(0) = 0$
  - $T(1) = 1;$
  - $T(2) = 2 + 1 = 3$
  - $T(3) = 6 + 1 = 7$
  - $T(4) = 14 + 1 = 15$

  Generalizando, temos a forma fechada $T(n) = 2^n - 1$, ou seja, $T(n) = O(2^n)$

Algoritmos Recursivos

- Provando a forma fechada por Indução Matemática
  - Caso base: Temos que $T(0) = 2^0 - 1 = 0$ e $T(1) = 2^1 - 1 = 1$, conforme descrito na recorrência.
  - Indução: Faremos a indução em n. Supomos que a forma fechada seja válida para todos os valores até $n - 1$, ou seja, $T(n - 1) = 2^{n-1} - 1$. Provaremos que a forma fechada também é válida para T(n):

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

Algoritmos Recursivos

- Encontrando Formas Fechadas
  - Podemos encontrar a forma fechada para o valor de $T(n)$ normalmente em três etapas:
    - 1) Analisar os pequenos casos de $T(n)$, o que pode nos fornecer insights;
    - 2) Encontrar e provar uma recorrência para o valor de $T(n)$;
    - 3) Encontrar e provar uma forma fechada para a recorrência.

Algoritmos Recursivos e Recorrências

- Aplicação e Resolução
  - A complexidade de algoritmos recursivos pode ser frequentemente descrita através de recorrências.
  - Geralmente, recorremos ao **Teorema Mestre** para resolver estas recorrências. Em casos em que o Teorema Mestre não se aplica, a recorrência deve ser resolvida de outras maneiras.
  - Resolver uma recorrência significa eliminar as referências que ela faz a si mesma.
  - Três dos métodos mais comuns para resolução de recorrências são o **método de substituição**, o **método de árvore de recursão** (ou expansão) e o **teorema mestre**.

Algoritmos Recursivos e Recorrências

- “Truques” de Resolução
  - Existem alguns “truques” para a resolução de recorrências:
    - Procurar por algum padrão ao expandir uma recorrência, como alguma recorrência básica.
    - Realizar manipulações algébricas, como troca de variáveis ou divisão da recorrência, que favoreçam a resolução.

Para tanto, é necessário ter conhecimento algébrico, de recorrências básicas e uma dose de “maldade”.

Algoritmos Recursivos e Recorrências

- Alguns Somatórios Úteis

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad (1)$$

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1 \quad (2)$$

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k} \quad (3)$$

$$\sum_{i=0}^{k} a^i = \frac{a^{k+1} - 1}{a - 1} \quad (a \neq 1) \quad (4)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \quad (5)$$

Algoritmos Recursivos e Recorrências

- Exemplo
  - T(1)
  - T(n) = 2 * T(n/2) + 2,   para n ≥ 2, 4, 8, 16, ..., 2^i , ...

n	1	2	4	8	16	...
T(n)	1	4	10	22	46	

3n – 2

correto?

Método Iterativo

Método Iterativo

- Consiste em iterar a recorrência e escrevê-la como uma somatória de termos que dependem apenas de n

$$T(1) = 1$$

$$T(n) = 2 * T(n/2) + 2, \qquad \text{para } n \geq 2, 4, 8, 16, \ldots, 2^i, \ldots$$

  - $T(n) = 2 * T(n/2) + 2$ (iteração 1)

    $= 2 * \mathbf{[2 * T(n/4) + 2]} + 2 = 2^2 * T(n/2^2) + 6$ (iteração 2)

    $= 2 * \mathbf{[2^2 * T(n/2^3) + 6]} + 2 = 2^3 * T(n/ 2^3) + 14$ (iteração 3)

  - ...
  - $= 2^i * T(n / 2^i) + 2^{i+1} - 2$ (iteração **i**)

Método Iterativo

- Consiste em iterar a recorrência e escrevê-la como uma somatória de termos que dependem apenas de n

  $T(1) = 1$

  $T(n) = 2 * T(n/2) + 2,$ para $n \geq 2, 4, 8, 16, \ldots, 2^i, \ldots$

  - para **i-ésima** iteração?
    - $= 2^i * T(n / 2^i) + 2^{i+1} - 2$ (iteração **i**)
  - Quando parar?
    - $T(n / 2^i) = T(1)$
    - $n / 2^i = 1$ -> $n = 2^i$ -> $i = \log n$

Método Iterativo

- para **i-ésima** iteração?
  - $= 2^i * T(n / 2^i) + 2^{i+1} - 2$ (iteração **i**)
- Quando parar?
  - $T(n / 2^i) = T(1)$
  - $n / 2^i = 1$ -> $n = 2^i$ -> $i = \log n$
- substituindo na fórmula

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

Método da Substituição

Método da Substituição

- Ideia geral
  - “adivinhe” a forma da solução
  - use indução para encontrar as constantes e prove que a solução está correta

Método da Substituição

- Recorrência

  $T(1) = 1$

  $T(n) = 2 * T(n/2) + 2,$ para $n \geq 2, 4, 8, 16, ..., 2^i, ...$

  - Chute
    - $T(n) = 3n - 2$ para $n = 1, 2, 4, 8, 16, ...$
  - Utilizar indução para provar que o chute está correto....

Método da Substituição

- Utilizar indução para provar que o chute está correto

  Recorrência **$T(n) = 2 * T(n/2) + 2$ para $n \geq 2, 4, 8, 16, ...$** Chute **$3n - 2$**

  - $T(1) = 3*1 - 2 = 1$
  - $T(n) = 2 * T(k / 2) + 2$ para $n \geq 2, 4, 8, 16$
  - Assumimos que $T(n/2) = 3n - 2$

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

$$= 2 * (3n - 2) + 2 = 2 * (3(n/2) - 2) + 2$$
$$= 6(n/2) - 4 + 2 = 3n - 2$$

Método da Substituição

- Como fazer boas escolhas
    - não há uma forma geral de adivinhar soluções corretas para recorrências
    - adivinhar uma solução depende de experiência e criatividade
    - se a recorrência é similar a alguma que você viu antes, então chutar uma solução similar é razoável
    - uma outra forma é provar limitantes grandes, inferior e superior, sobre a recorrência e então ir reduzindo a faixa de incerteza

Método da Árvore de Recursão

Método da Árvore de Recursão

- Ideia Geral
  - no método da substituição, um bom chute pode ser difícil
  - desenhar uma árvore de recursão é uma boa forma de obter um bom chute
  - em uma **árvore de recursão** cada vértice representa o custo de um subproblema unitário, no conjunto de chamadas da função recursiva
  - somamos o custo em cada nível da árvore para obter os custos por nível e então somamos todos os custos por nível para determinar o custo total de todos os níveis da recursão

Método da Árvore de Recursão

- Ideia Geral
    - uma árvore de recursão é melhor usada para obter um bom chute, que pode então ser provado pelo método da substituição
    - quando usamos uma árvore de recursão para fazer um chute, podemos ignorar alguns detalhes
    - se, por outro lado, formos cuidadosos no desenho da árvore de recursão e na soma de seus custos, podemos usá-la como uma prova direta da solução de uma recorrência
    - em geral, optamos por usar a árvore de recursão para obter um bom chute da solução de uma recorrência e, em seguida, provamos que o chute está correto usando o método da substituição

Método da Árvore de Recursão

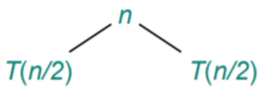
- Exemplo

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

Método da Árvore de Recursão

- Exemplo

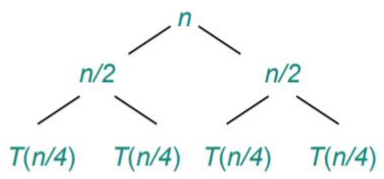
$$T(1) = 1$$
$$T(n) = 2*T(n/2) + n$$



Método da Árvore de Recursão

- Exemplo

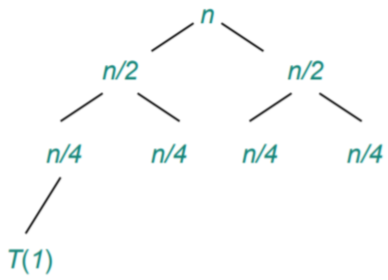
$$T(1) = 1$$
$$T(n) = 2*T(n/2) + n$$



Método da Árvore de Recursão

- Exemplo

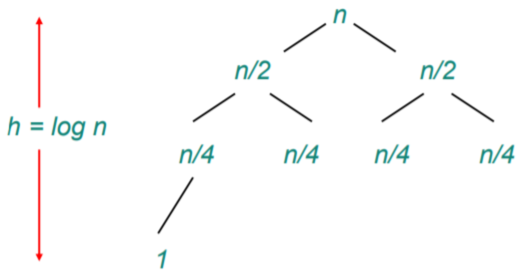
$$T(1) = 1$$
$$T(n) = 2*T(n/2) + n$$



Método da Árvore de Recursão

- Exemplo

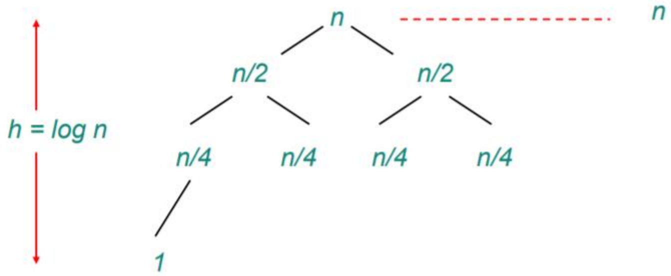
$$T(1) = 1$$
$$T(n) = 2*T(n/2) + n$$



Método da Árvore de Recursão

- Exemplo

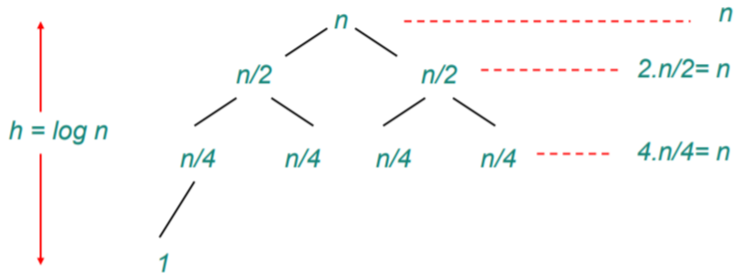
$$T(1) = 1$$
$$T(n) = 2*T(n/2) + n$$



Método da Árvore de Recursão

- Exemplo

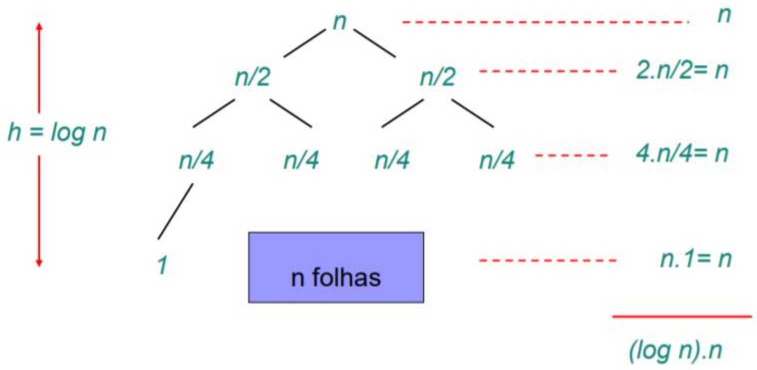
$$T(1) = 1$$
$$T(n) = 2*T(n/2) + n$$



Método da Árvore de Recursão

- Exemplo

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



Método Mestre

Método Mestre

- Ideia Geral
  - o método mestre é uma receita para solucionar recorrências da forma

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

  - onde $a \geq 1$ e $b > 1$ são constantes positivas e $f(n)$ é uma função assintoticamente positiva
  - A recorrência acima descreve a complexidade de tempo de um algoritmo que divide o problema original de tamanho n em a subproblemas de tamanho n/b.

Método Mestre

- Ideia Geral
  - Os a subproblemas são resolvidos em tempo T(n/b) cada um.
  - A função f(n) engloba o custo de dividir o problema original e, eventualmente, combinar os resultados dos subproblemas.
  - Note que n/b pode não ser inteiro, entretanto, o termo T(n/b) pode ser substituído por T($\lceil n/b \rceil$) ou T($\lfloor n/b \rfloor$) sem afetar o comportamento assintótico da recorrência.

Método Mestre

- Usando o Teorema Mestre
  - O Teorema Mestre enumera três casos em que se torna fácil resolver recorrências.
  - Note que não estamos interessados em obter a forma fechada para a recorrência, mas sim em seu comportamento assintótico, de maneira direta.
  - Isto é o contrário de quando empregamos o método de substituição, no qual encontramos a forma fechada e depois analisamos seu comportamento assintótico.

Método Mestre

- Teorema
  - Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função e $T(n)$ definida sobre inteiros não negativos pela recorrência

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

em que interpretamos $T(n/b)$ como $T(\lceil n/b \rceil)$ ou $T(\lfloor n/b \rfloor)$.

  - Então $T(n)$ possui os seguintes limites assintóticos:
    - **1** Se $f(n) \in O(n^{\log_b a - \varepsilon})$ para alguma constante $\varepsilon > 0$, então $T(n) \in \Theta(n^{\log_b a})$.
    - **2** Se $f(n) \in \Theta(n^{\log_b a})$, então $T(n) \in \Theta(n^{\log_b a} \log n)$.

Método Mestre

- Teorema
  - Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função e $T(n)$ definida sobre inteiros não negativos pela recorrência

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

em que interpretamos $T(n/b)$ como $T(\lceil n/b \rceil)$ ou $T(\lfloor n/b \rfloor)$.

  - Então $T(n)$ possui os seguintes limites assintóticos:
    - **3** Se $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ para alguma constante $\varepsilon > 0$ e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) \in \Theta(f(n))$.

Método Mestre

- Porquê três casos?
  - Nos 3 casos, comparamos uma função f(n) com a função $n^{\log_b a}$. A complexidade da recorrência é determinada pela maior das duas:
    - No primeiro caso, a função $n^{\log_b a}$ é maior, portanto, $T(n) = \Theta(n^{\log_b a})$.
    - No terceiro caso, a função f(n) é maior, portanto, $T(n) = \Theta(f(n))$.
    - No segundo caso, as duas funções têm o mesmo “tamanho”. A solução é multiplicada por um fator logarítmico, portanto, $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(f(n) \log n)$.

Método Mestre

- Exemplo 1

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

  - a = 9
  - b = 3
  - f(n) = n
  - Aplicando o caso 1 do teorema mestre:
    - Se $f(n) \in (n^{\log_b a - \varepsilon})$ para alguma constante $\varepsilon > 0$, então $T(n) \in \Theta(n^{\log_b a})$.

$$\boxed{\begin{array}{c} n \in O\left(n^{1,999}\right) \\ T(n) \in \Theta(n^2) \end{array}}$$

Método Mestre

- Exemplo 2

$$T(n) = 2T(n/4) + \sqrt{n}$$

  - a = 2
  - b = 4
  - f(n) = $\sqrt{n}$
  - Aplicando o caso 2 do teorema mestre:
    - Se $f(n) \in \Theta(n^{\log_b a})$ então $T(n) \in \Theta(n^{\log_b a} \log n)$

$$\boxed{\begin{array}{c} \sqrt{n} \in \Theta\left(n^{\log_4 2}\right) \in \Theta\left(n^{1/2}\right) \\ T(n) \in \Theta(\sqrt{n} \log n) \end{array}}$$

Método Mestre

- Exemplo 3

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

  - $a = 3$
  - $b = 4$
  - $f(n) = n \log n$
  - Aplicando o caso 3 do teorema mestre:
    - Se $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ para alguma constante $\varepsilon > 0$ e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) \in \Theta(f(n))$.

Método Mestre

- Exemplo 3

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

  - a = 3 b = 4 f(n) = $n \log n$
  - n log n ∈ $\Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{\log_4 3 + \varepsilon}) = \Omega(n^{0{,}7925})$

$$af(n/b) \le cf(n)$$
$$3f(n/4) \le cf(n)$$
$$3[(n/4)\log(n/4)] \le \frac{3}{4} n \log n$$
$$\frac{3n}{4}(\log n - \log 4) \le \frac{3}{4} n \log n$$
$$\frac{3}{4} n \log n - 2 \le cn \log n$$
$$para\ c = 3/4$$

T(n) ∈ Θ(n log n)

Exercícios

Exercícios

- Resolva os exercícios 4.3-1, 4.3-2, 4.3-3, 4.3-8, 4.4-1, 4.4-2 e 4.5-1 do livro do Cormen