Estruturas de Dados e Básicas I - DIM0119

Selan R. dos Santos

DIMAp – Departamento de Informática e Matemática Aplicada
Sala 231, ramal 231, selan@dimap.ufrn.br
UFRN

2018.1

Análise de Alg. Recursivos — Conteúdo

Análise de Algoritmos Recursivos
Exemplo fatorial

- Vamos começar por um exemplo: o cálculo recursivo do fatorial de n.
- $n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$, para $n \geq 1$ e $0! = 1$.
- Definição recursiva de $n!$: $\begin{cases} F(n) = & F(n-1) \cdot n, \quad \text{para } n \geq 1 \\ F(0) = & 1 \end{cases}$

Fatorial recursivo de n

Entrada : Inteiro n não negativo.
Saída : O valor de $n!$.

```
função F(n: inteiro): inteiro
    se n = 0 então                # caso base
        retorna 1
    senão
        retorna F(n − 1)*n
```


Análise de Algoritmos Recursivos
Exemplo fatorial (cont.)

- A operação básica é a multiplicação.
- De acordo com a definição, o número de multiplicações deve satisfazer:

$$M(n) = \underbrace{M(n-1)}_{\text{Computar } F(n-1)} \quad \underbrace{+1}_{\text{Multiplicar } F(n-1) \text{ por } n} \qquad \text{para} \quad n > 0.$$

- Perceba que a relação acima não é expressa explicitamente em função de n, mas sim implicitamente como uma função de seu valor em outro ponto, $n-1$.
- Portanto, esta relação é chamada de relação de recorrência.
- Precisamos resolver a recorrência para exprimir $M(n)$ apenas em função de n.

Análise de Algoritmos Recursivos
Exemplo fatorial (cont.)

- Para uma solução única, precisamos de uma condição inicial, que faz a recursão parar, ou seja, **Se** `n=0` **então retorna** `1;`
- Portanto a condição inicial procurada é $M(0) = 0$ (ou seja, nenhuma multiplicação é realizada quando $n = 0$).
- Portanto encontramos a relação de recorrência e a condição inicial:

$$\begin{cases} M(n) & = & M(n-1) + 1 \quad \text{para } n > 0, \\ M(0) & = & 0. \end{cases}$$

- Perceba que esta função recursiva é diferente da função que calula o fatorial; $M(n)$ representa o número de multiplicações necessárias para calcular o fatorial de n considerando o algoritmo apresentado anteriormente.

Análise de Algoritmos Recursivos
Exemplo fatorial (cont.)

- Para resolver a recorrência vamos usar o método da substituição.

$$\begin{aligned} M(0) &= 0 \\ M(n) &= M(n-1) + 1 \quad \text{passo 1} \\ &= [M(n-2)+1] + 1 = M(n-2) + 2 \quad \text{passo 2} \\ &= [M(n-3)+1] + 2 = M(n-3) + 3 \quad \text{passo 3} \\ &\ldots \\ &= M(n-k) + k \quad k\text{-ésimo passo} \end{aligned}$$

- Como a relação é especificada para $M(0)$, então $n - k = 0$, ou teremos n passos, ou $k = n$.
- Logo, $M(n) = M(0) + n = 0 + n = n$, i.e. $O(n) = n$ ou linear (mesmo da versão iterativa).

Eficiência Temporal de Algoritmos Recursivos

TODO: fazer a análise de complexidade espacial para o cálculo do fatorial.

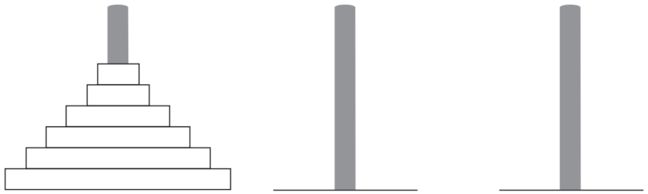
Eficiência Temporal de Algoritmos Recursivos

Plano Geral para Análise de Algoritmos Recursivos

- Decidir sobre o parâmetro n associado a entrada dos dados;
- Identificar a operação básica do algoritmo;
- Determinar as situações de entrada n que correspondem ao pior e melhor casos;
- Determinar a relação de recorrência com a condição inicial apropriada que expressa o número de vezes que a operação básia é executada;
- Resolver a relação de recorrência ou, pelo menos, determinar sua *ordem de crescimento*.

Exemplo #2: Torres de Hanoi

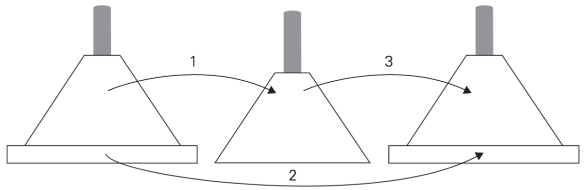
- O problema das torres de Hanoi requer que n discos de tamanhos diferentes sejam movidos da torre inicial para a final, usando uma segunda torre como auxiliar.
- Restrições: um disco maior não pode ficar sobre um menor e apenas um disco pode ser movido por vez.



Exemplo #2: Torres de Hanoi

Solução recursiva

- Considere que existe uma função recursiva `hanoi(`n`,`A`,`B`,`C`)` que consegue mover n discos de A para B usando C como auxiliar.
- Para mover $n > 1$ discos da 1ª para a 3ª torre (usando a 2ª como auxiliar) basta: (1) mover $n-1$ discos da 1ª para a 2ª torre (usando a 3ª como auxiliar), (2) mover o maior disco da 1ª para a 3ª torre e, novamente, (3) chamar a função recursiva para mover os $n-1$ discos da 2ª para a 3ª torre (usando a 1ª como auxiliar).



- O caso base acontece quando temos apenas um disco, ou $n = 1$; neste caso basta mover o disco diretamente da 1ª para a 3ª torre.

Exemplo #2: Torres de Hanoi

Análise temporal

- O parâmetro de entrada é o número n de discos.
- A operação básica é mover um disco.
- O número de movimentos $M(n)$ claramente depende de n então temos a seguinte equação de recorrência: $M(n) = M(n-1) + 1 + M(n-1)$, para $n > 1$.
- A condição inicial é $M(1) = 1$, logo temos a relação de recorrência para o número de movimentos $M(n)$:

$$\begin{cases} M(n) = 2M(n-1) + 1 & \text{para } n > 1, \\ M(1) = 1. \end{cases}$$

Exemplo #2: Torres de Hanoi

Análise temporal (cont.)

- Resolvendo com o método da substituição.

$$\begin{aligned} M(n) &= 2M(n-1) + 1 & M(n-1) = 2M(n-2) + 1 \\ &= 2[2M(n-2) + 1] + 1 = 2^2 M(n-2) + 2 + 1 & M(n-2) = 2M(n-3) + 1 \\ &= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3 M(n-3) + 2^2 + 2 + 1. & \end{aligned}$$

- A série sugere que a próxima substituição gera: $2^4 M(n-4) + 2^3 + 2^2 + 2 + 1$, de maneira que após a i-ésima substituição teríamos:

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^i M(n-i) + 2^i - 1.$$

- Como sabemos que $M(1) = 1$, o qual corresponde a $i = n - 1$ iteração, temos:

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

Exemplo #2: Torres de Hanoi

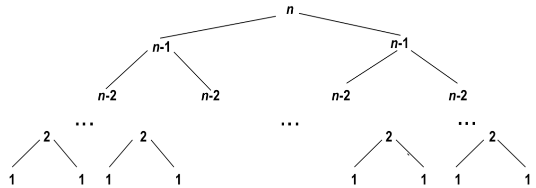
Considerações

- ▷ Note que a complexidade é exponencial, mas o algoritmos é o mais eficiente possível (o problema é complexo mesmo).
- ▷ Muitas vezes um algoritmo recursivo pode "esconder" sua ineficiência temporal devido a sua natureza sucinta.
- ▷ Quando algoritmos recursivos fazem mais de uma chamada a si mesmo, é interessante construir uma árvore de chamadas recursivas para nos ajudar a entender a complexidade.

Exemplo #2: Torres de Hanoi

Considerações (cont.)

- ▷ Os nós correspondem às chamadas recursivas, cujo rótulo corresponde aos parâmetros da chamada.



- ▷ Contando o número de nós na árvore temos o número de chamadas recursivas:

$$C(n) = \sum_{l=0}^{n-1} 2^l \quad \text{(onde } l \text{ é o nível da árvore)} \quad = 2^n - 1.$$

Princípios de Análise de Algoritmo

Exemplo 3: Soma de elementos em um vetor

Algoritmo para somar os elementos de uma lista

Entrada : Vetor L e o seu tamanho n.
Saída : A soma de todos os elementos em L.

```
função soma(L: arranjo de inteiro; n: inteiro): inteiro
    var resposta: inteiro                      # guarda resultado da soma
    se n = 0 então # c1
        resposta ← 0 # c2
    senão
        resposta ← L[0]+soma(L, n − 1) # c3
    retorna resposta # c4
```

- ▷ Cada chamada recursiva da soma decrementa o tamanho da lista. Exceção quando a lista é zero (caso base da recursão).
- ▷ Se n é o tamanho inicial, então o número total de chamadas será $n+1$.
- ▷ O custo de cada chamada é: $c_1 + c_2 + c_4$ (última chamada) e $(c_1 + c_3 + c_4)$ (demais chamadas).
- ▷ O tempo total $T = n \cdot (c_1 + c_3 + c_4) + c_1 + c_2 + c_4$, ou seja, $T \leq n \cdot c$, onde c é constante ⇒ a complexidade é dita **linear**.

Princípios de Análise de Algoritmo

Exemplo 3: Soma de elementos em um vetor

- ▷ Para avaliar a complexidade em relação a memória necessária, é preciso compreender como a memória de microprocessadores é organizada.
- ▷ Cada chamada de função ocupa um espaço na pilha de execução, onde é reservado espaço para variáveis locais e parâmetros da função chamada (assumindo passagem de parâmetro por valor).
- ▷ No caso do exemplo existem $n+1$ chamadas de função. Portanto o consumo de memória é o somatório do comprimento das listas

$$n + (n-1) + (n-2) + \cdots + 1 + 0 =$$
$$\textstyle\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \leq c \cdot n^2$$

- ▷ Portanto a **complexidade espacial** é uma função quadrática da entrada n.

Referências

J. Szwarcfiter and L. Markenzon
Estruturas de Dados e Seus Algoritmos, 2ª edição, **Cap. 1**.
Editora LTC, 1994.

R. Sedgewick
Algorithms in C, Parts 1-4, 3rd edition. **Cap. 2**
Addision Wesley, 2004.

A. Drozdeck
Data Structures and Algorithms in C++, 2nd edition. **Cap. 2**
Brooks/Cole, Thomsom Learning, 2001.

D. Deharbe
Slides de Aula. **aula 2**
DIMAp, UFRN, 2006.

M. Siqueira
Slides de Aula. **aula 1**
DIMAp, UFRN, 2009.