Algoritmos e Estrutura de Dados

Aula 02 – Análise de Algoritmos, Notações e Funções Comuns

Prof. Tiago A. E. Ferreira

Roteiro da Aula

- Alguns Conceitos Básicos
- Análise de Algoritmos
  - Pior caso
  - Caso médio
  - Dividir e conquistar
- Funções
  - Notação Assintótica
  - Funções Comuns

Problema de Ordenação

- Entrada:
  - Uma seqüência de *n* números:
    - $< a_1, a_2, ..., a_n >$
- Saída:
  - Uma permutação dos números de entrada:
    - $< a'_1, a'_2, ..., a'_n >$, *tal que* $a'_1 \leq a'_2 \leq ... \leq a'_n$ *(ordenação crescente)*

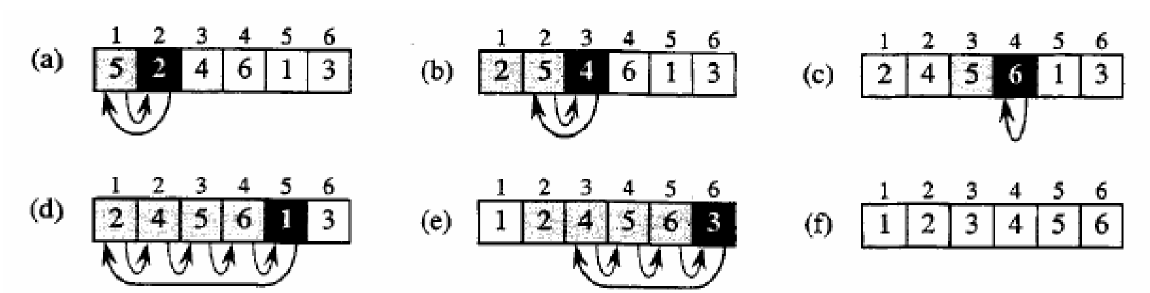
- *Obs.: os números que deseja-se ordenar serão chamados de* ***chaves***

Ordenamento por inserção

- O ordenamento por inserção segue uma idéia bastante intuitiva:
  - Jogo de cartas: arrumamos as carta em uma certa seqüência a medida que as pegamos

Procedimento

- Procedimento: *insertion-sort*
- Entrada:
  - Arranjo de números *A[1...n]*
- Saída:
  - Arranjo de números *A[1...n]* ordenados
    - *Os números de entrada são ordenados no local*
- Exemplo: Seja *A=<5 2 4 6 1 3>*



Pseudo-Código

```
INSERTION-SORT(A)
for j ← 2 to comprimento[A]
     do chave ← A[j]
        ▷ Inserir A[j] na seqüência ordenada A[1..j – 1].
        i ← j – 1
        while i > 0 e A[i] > chave
           do A[i + 1] ← A[i]
              i ← i – 1
        A[i + 1] ← chave
```

Loop invariante

Loop **for**

Loop **While**

O símbolo ▷ indica um comentário

Algoritmo é correto?

- Os *loops invariantes* são utilizados para ajudar o entendimento do por que um algoritmo é correto
  - Detalhes a serem mostrados:
    - **Inicialização**: ele é verdadeiro antes da primeira iteração
    - **Manutenção**: Se for verdadeiro antes de uma iteração do loop, continuará verdadeiro antes da próxima iteração
    - **Término**: Quando o loop termina, o invariante fornece uma propriedade útil que ajuda a mostrar que o algoritmo é correto
  - Quando as duas primeiras propriedades são válidas, o loop invariante é verdadeiro antes de toda a iteração do loop

Para o Exemplo de Ordenação

- **Inicialização:**
  - *j = 2,* o sub-arranjo *A[1..j-1]* consiste apenas no único elemento *A[1],* que é elemento de *A* e está ordendado
    - *Isto mostra que o loop invariante é válido antes da primeira iteração*
- **Manutenção:** (cada iteração mantém o loop invariante)
  - A medida que o **for** corre, desloca-se uma casa à direita em *A* (*A[j-1] A[j-2] A[j-3]...*) a procura da posição ideal para *A[j], mantendo o* sub-arranjo *A[1..j-1] ordenado*
- **Término:** (o que ocorre ao fim do loop)
  - *j = n+1,* sendo gerado o sub-arranjo *A[1..n]* que contem todos os elementos da seqüência de entrada e está ordenado
  - **Logo o algoritmo é correto!**

Análise de Algoritmos

- Analisar Algoritmo é...
  - Prever recursos necessários
    - Tempo de processamento
    - Memória necessária
    - Largura de banca para comunicações
    - Etc...
  - Descartar algoritmos inviáveis
  - Escolher algoritmo correto mais barato computacionalmente

Análise do algoritmo de ordenação

- O custo do algoritmo de ordenação dependerá:
  - Tamanho da entrada
    - Número de itens na entrada (***n***) para o problema
  - Grau do pré-ordenamento da entrada
- Tempo de execução de um algoritmo
  - Em uma determinada entrada, é o número de operações primitivas ou “etapas” executadas
    - Pode-se definir uma etapa por um passo no algoritmo que seja o mais independente possível da máquina
    - Considera-se que cada linha do pseudo-código leve um tempo constante para sua execução
      - *i-ésima* linha, tempo c_i

Custos por linha e total

INSERTION-SORT(A)	*custo*	*vezes*
for $j \leftarrow 2$ **to** *comprimento*$[A]$	c_1	n
do *chave* $\leftarrow A[j]$	c_2	$n-1$
▷ Inserir $A[j]$ na seqüência ordenada $A[1..j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ e $A[i] >$ *chave*	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow$ *chave*	c_8	$n-1$

Nº de vezes que o teste do *While* é executado

Custo Total:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) - c_8(n-1).$$

Melhor caso

- No melhor caso, a entrada já se encontra ordenada!
  - Custo:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8).
\end{aligned}
$$

    - Neste caso o teste do ***While*** é executado apenas uma vez para cada passo do ***for***.
    - Este custo pode ser escrito como ***an+b***, onde *a* e *b* são constantes, i.e., uma função linear em *n*

Pior Caso

- No pio caso, a entrada se encontra ordenada de forma decrescente! (ordem inversa de que se deseja ordenar)
  - Custo:
    - Para o pior caso, o teste do ***While* é repetido *j* vezes para cada passo do *for*.**
    - **Então:**

$$\sum_{j=2}^{n} j = \frac{n(n-1)}{2} - 1 \qquad \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

Pior Caso

- Portanto o custo total é dado por:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n-1)}{2} - 1\right)$$
$$+ c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$
$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n$$
$$- (c_2 + c_4 + c_5 + c_8).$$

- Esta é uma função do tipo: ***an^2+bn+c***
  - Um função quadrática de *n*

Notação O(·)

- As equações de custo são dependentes das constantes $c_1, c_2, \ldots, c_k$
  - Fica impraticável tentarmos estabelecer os valores exatos destas constantes
- Desta forma, é importante ter uma estimativa de como o custo cresce com o tamanho ***n*** da entrada
  - Considera-se o termo de maior importância da expressão de custo:
    - $an^2+bn+c \rightarrow O(n^2)$: *da ordem de* n^2
    - $an+b \rightarrow O(n)$: *da ordem de* n

Notação O

- Quando se considera o número de passo efetuados por um algoritmo,
  - Pode-se desprezar constantes aditivas e multiplicativas
    - Ex.: $3n$ será aproximado por n
  - São considerados apenas valores assintóticos, termos de menor grau podem ser desprezados
    - n^2+n+5 será aproximado por n^2

Notação O

- Formalizando:
  - Sejam f e h funções positivas e uma variável inteira n. Diz-se que f é $O(h)$, escrevendo-se $f=O(h)$, quando existir uma constante $c>0$ e um valor inteiro n_0, tal que

$$n>n_0 \Rightarrow f(n) \leq ch(n)$$

  - Ou seja, a função $h(n)$ é um limite superior para a função $f(n)$
- Propriedades, onde g e h são funções e k é uma constante:
  - $O(g+h) = O(g) + O(h)$
  - $O(k.g) = k.O(g) = O(g)$

Notação Θ

- A notação Θ é útil para exprimir limites superiores
  - Sejam *f* e *g* funções reais e positivas da variável inteira *n.* Diz que *f* é $\Theta(g)$, escrevendo-se $f = \Theta(g)$, quando ambas as condições $f = O(g)$ e $g = O(f)$ forem verificadas
  - A notação Θ exprime o fato que duas funções possuem a mesma ordem de grandeza assintótica.
  - Exs.:
    - $f=n^2-1$; $g=n^2$; $h=n^3$, *assim,* $f= \Theta(g)$ *mas* f *não é* $\Theta(h)$

Notação Ω

- A notação Ω é utilizada para limites assintóticos inferiores
  - Sejam *f e h* funções reais positivas da variável *n*. Diz-se que *f é* $\Omega(h)$, escrevendo-se $f= \Omega(h)$, quando existir uma constante $c > 0$ e um valor inteiro n_0, tal que

$$n>n_0 \Rightarrow f(n) \geq c.h(n)$$

  - *Exemplo, se* $f=n^2-1$*, então é válido afirmar:*
    - $f= \Omega(n^2)$
    - $f= \Omega(n)$
    - $f= \Omega(1)$
    - *Mas não é válido afirmar que* $f= \Omega(n^3)$

Funções Comuns

- Funções monótonas
  - Função monotonicamente crescente (ou monotonamente crescente)
    - Se $m \leq n \Rightarrow f(m) \leq f(n)$
  - Função monotonicamente decrescente (ou monotonamente decrescente)
    - Se $m \leq n \Rightarrow f(m) \leq f(n)$
- Funções estritas
  - Função estritamente crescente
    - Se $m \leq n \Rightarrow f(m) < f(n)$
  - Função estritamente decrescente
    - Se $m \leq n \Rightarrow f(m) \leq f(n)$

Funções Comuns

- Piso
  - Seja x um número real, seu piso é o maior inteiro menor ou igual a *x*
  - Notação: $\lfloor X \rfloor$
- Teto
  - Seja x um número real, seu teto é o menor inteiro maior que ou igual a *x*
  - Notação $\lceil x \rceil$
- Para todo real x:
  - $x-1 > \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$
- Para qualquer inteiro n
  - $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$
- Para qualquer real $n \geq 0$ e inteiros $a,b > 0$
  - $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$
  - $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$
  - $\lceil a/b \rceil \leq (a+(b-1))/b$
  - $\lfloor a/b \rfloor \leq (a-(b-1))/b$

Funções Comuns

- Aritmética Modular
  - Para qualquer inteiro *a* e qualquer inteiro positivo *n*,

$$a \bmod n = a - \lfloor a/n \rfloor n$$

  - *Se (a mod n) = (b mod n),* então *a* $\equiv$ *b (mod n) (**a** é equivalente a **b**, modulo **n**)*
- Polinômios
  - Dado um inteiro não negativo *d*, um *polinômio em n de grau d* é uma função *p(n)* da forma,

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

  - É dito que uma função *f(n)* é **polinomialmente limitada** se $f(n)=O(n^k)$ para alguma constante *k*.

Funções Comuns

- Função Exponencial
  - Para todo *n e a ≥ 1*, a função a^n é monotonicamente crescente em *n*.
  - Para *a>1,*

$$\lim_{n\to\infty}\frac{n^b}{a^n}=0 \quad \Rightarrow \quad n^b = O\left(a^n\right)$$

    - *Portanto, qualquer função exponencial com base escritamente maior que 1 cresce mais rapidamente que qualquer função polinomial*
- Função Logarítmica

$$\lim_{n\to\infty}\frac{\log_b n}{n^a}=0 \quad \Rightarrow \quad \log_b n = O\left(n^a\right)$$

  - Para qualquer constante *a>0.* Qualquer função polinomial positiva cresce mais rapidamente que qualquer função polilogarítmica.

Funções Comuns

- Função Fatorial

$$n! = \begin{cases} 1 & se\ n = 0 \\ n \cdot (n-1)! & se\ n > 0 \end{cases}$$

  - Note que $n! \leq n^n$, logo $n! = O(n^n)$

- Iteração Funcional

$$f^{(i)} = \begin{cases} n & se\ i = 0 \\ f\left(f^{(i-1)}(n)\right) & se\ i > 0 \end{cases}$$

  - Se $f(n)=2n$, então $f^{(i)}(n)=2^i n$

Números de Fibonacci

- Os números de Fibonacci são definidos como:
  - $F_0=0$
  - $F_1=1$
  - $F_i=F_{i-1}+F_{i-2}$, para $i \geq 2$
- Seqüência: 0,1,1,2,3,5,8,13,21,34,55,89,...
- Uma forma alternativa e definir os números de Fibonacci é a partir da **razão áurea** ϕ e seu conjugado $\hat{\phi}$

$$\left.\begin{aligned} \phi &= \frac{1+\sqrt{5}}{2} = 1{,}61803\ldots \\ \hat{\phi} &= \frac{1-\sqrt{5}}{2} = 0{,}61803\ldots \end{aligned}\right\} F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$