Algoritmos de Ordenação

ACH2002 - Introdução à Ciência da Computação II

Delano M. Beder

Escola de Artes, Ciências e Humanidades (EACH)
Universidade de São Paulo
dbeder@usp.br

10/2008

Material baseado em slides dos professores Cid de Souza e Cândida da Silva

O problema da Ordenação

Problema

Ordenar um conjunto de $n \geq 1$ inteiros.

- Podemos projetar por indução diversos algoritmos para o problema da ordenação.
- Na verdade, todos os algoritmos básicos de ordenação surgem de projetos por indução sutilmente diferentes.

Ordenação por indução: paradigma incremental

OrdenaçãoIncremental(A, n)

```
Entrada: Vetor A de n números inteiros.
Saída: Vetor A ordenado.
se n == 1 então
      retorne
se não
      <comandos iniciais>
      OrdenaçãoIncremental(A, n - 1)
      <comandos finais>
fim se
retorne
```


Ordenação por indução: Divisão e conquista

OrdenaçãoD&C(A, ini, fim)

```
Entrada: Vetor A de n números inteiros.
Saída: Vetor A ordenado.
n = fim - ini + 1
se n == 1 então
      retorne
se não
      <comandos iniciais: a divisão> (cálculo de q !)
      OrdenaçãoD&C(A, ini, q)
      OrdenaçãoD&C(A, q + 1, fim)
      <comandos finais: a conquista>
fim se
retorne
```

Projeto por Indução Simples

Hipótese de indução simples

Sabemos ordenar um conjunto de $n - 1 \geq 1$ inteiros.

- **Caso base:** n = 1. Um conjunto de um unico elemento está ordenado.
- **Passo da Indução (Primeira Alternativa):** Seja S um conjunto de $n \geq 2$ inteiros e x um elemento qualquer de S.
  Por hipótese de indução, sabemos ordenar o conjunto $S - x$, basta então inserir x na posição correta para obtermos S ordenado.
- Esta indução dá origem ao algoritmo incremental *Insertion Sort* (Inserção Direta).

Insertion Sort - Pseudo-código - Versão Recursiva

OrdenaçãoInserção(A, n)

Entrada: Vetor A de n números inteiros.
Saída: Vetor A ordenado.

```
se n ≥ 2 então
    OrdenaçãoInserção(A, n - 1)
    v = A[n-1]
    j = n - 1
    enquanto (j > 0) e (A[j-1] > v) faça
        A[j] = A[j-1]
        j = j - 1
    A[j] = v
```


Insertion Sort - Pseudo-código - Iterativa

OrdenaçãoInserção(A, n)

Entrada: Vetor A de n números inteiros.
Saída: Vetor A ordenado.

```
para i = 1 até n - 1 faça
    v = A[i]
    j = i - 1
    enquanto (j > 0) e (A[j-1] > v) faça
        A[j] = A[j-1]
        j = j - 1
    A[j] = v
```


Insertion Sort - Análise de Complexidade

- Quantas comparações e quantas trocas o algoritmo *Insertion Sort* executa no pior caso ?
- Tanto o número de comparações quanto o de trocas é dado pela recorrência:

$$T(n) = \begin{cases} 0 & \text{se} \quad n = 1 \\ T(n-1) + n & \text{caso contrário} \end{cases}$$

- Portanto, $\Theta(n^2)$ comparações e trocas são executadas no pior caso.

Projeto por Indução Simples

Hipótese de Indução Simples:

Sabemos ordenar um conjunto de $n - 1 \geq 1$ inteiros.

- **Caso base:** n = 1. Um conjunto de um unico elemento está ordenado.
- **Passo da Indução (Segunda Alternativa):** Seja S um conjunto de $n \geq 2$ inteiros e x o menor elemento de S. Então x certamente é o primeiro elemento da seqüência ordenada de S e basta ordenarmos os demais elementos de S.
  Por hipótese de indução, sabemos ordenar o conjunto $S - x$ e assim obtemos S ordenado.
- Esta indução dá origem ao algoritmo incremental *Selection Sort* (Seleção Direta)

Selection Sort - Pseudo-código - Versão Recursiva

OrdenaçãoSeleção(A, ini, fim)

Entrada: Vetor A de n números inteiros e os índices de início e término da seqüência a ser ordenada.
Saída: Vetor A ordenado.

```
se ini < fim então
    min = ini
    para j = ini + 1 até fim faça
        se A[j] < A[min] então
            min = j
    t = A[min]
    A[min] = A[ini]
    A[ini] = t
    OrdenaçãoSeleção(A, ini + 1, fim)
```


Selection Sort - Pseudo-código - Versão Iterativa

OrdenaçãoSeleção(A, n)

Entrada: Vetor A de n números inteiros.
Saída: Vetor A ordenado.

```
para i = 0 até n - 2 faça
    min = i
    para j = i + 1 até n - 1 faça
        se A[j] < A[min] então
            min = j
    t = A[min]
    A[min] = A[i]
    A[i] = t
```


Selection Sort - Análise de Complexidade

- Quantas comparações e quantas trocas o algoritmo *Insertion Sort* executa no pior caso ?
- O número de comparações é dado pela recorrência:

$$T(n) = \begin{cases} 0 & \text{se} \quad n = 1 \\ T(n-1) + n & \text{caso contrário} \end{cases}$$

- Portanto, $\Theta(n^2)$ comparações são executadas no pior caso.

Selection Sort - Análise de Complexidade

- Já o número de trocas é dado pela recorrência:

$$T(n) = \begin{cases} 0 & \text{se} \quad n = 1 \\ T(n-1) + 1 & \text{caso contrário} \end{cases}$$

- Portanto, $\Theta(n)$ trocas são executadas no pior caso.
- Apesar dos algoritmos *Insertion Sort* e *Selection Sort* terem a mesma complexidade assintótica, em situações onde a operação de troca é muito custosa, é preferível utilizar *Selection Sort*.

Projeto por Indução Simples

- Ainda há uma terceira alternativa para o passo da indução.
- **Passo da Indução (Terceira Alternativa):** Seja S um conjunto de $n \geq 2$ inteiros e x o maior elemento de S. Então x é certamente o último elemento da seqüência ordenada de S e basta ordenarmos os demais elementos de S. Por hipótese de indução, sabemos ordenar o conjunto $S - x$ e assim obtemos S ordenado.
- Em princípio, esta indução dá origem a uma variação do algoritmo *Selection Sort*.
- No entanto, se implementamos de uma forma diferente a seleção e o posicionamento do maior elemento, obteremos o algoritmo *Bubble Sort*.

Bubble Sort - Pseudo-código - Versão Iterativa

BubbleSort(A, n)

Entrada: Vetor A de n números inteiros.
Saída: Vetor A ordenado.

```
1. para i = n - 1 decrescendo até 1 faça
2.     para j = 1 até i faça
3.         se A[j-1] > A[j] então
4.             t = A[j-1]
5.             A[j-1] = A[j]
6.             A[j] = t
```


Bubble Sort - Análise de Complexidade

- Quantas comparações e quantas trocas o algoritmo *Bubble Sort* executa no pior caso ?
- Tanto o número de comparações quanto o de trocas é dado pela recorrência:

$$T(n) = \begin{cases} 0 & \text{se} \quad n = 1 \\ T(n-1) + n & \text{caso contrário} \end{cases}$$

- Portanto, $\Theta(n^2)$ comparações e trocas são executadas no pior caso.
- Ou seja, algoritmo *Bubble Sort* executa mais trocas que o algoritmo *Selection Sort*!