INF 112: Programação II

Aula 07

- **Algoritmos de ordenação**

Fábio R. Cerqueira, UFV, DPI, frcerqueira@gmail.com

Algoritmos de ordenação

- *Ordenar* significa rearranjar um conjunto de objetos em ordem ascendente ou descendente. O principal objetivo é, posteriormente, facilitar a busca por algum desses objetos.

- Os objetos podem possuir vários atributos, mas somente um deles, denominado de *chave*, é usado na ordenação.

- Neste capítulo estudaremos diversos algoritmos de ordenação. Começaremos estudando os algoritmos elementares, que são adequados para sequências pequenas de elementos. Depois estudaremos alguns algoritmos mais sofisticados, que são mais adequados para grande volumes de dados.

Algoritmos de ordenação

- Existem varias formas de se classificar os algoritmos de ordenação. A tabela a seguir mostra um tipo de classificação (extraída de *Sedgewick*):

	Externos	Internos
Elementares		Ordenação por Seleção, Ordenação por Inserção, Método da bolha, shellsort
Complexos	Sort-merge, Balanced Multiway Merging	Quicksort, Radixsort, Mergesort, Heapsort

- Os algoritmos ***internos*** são aqueles que ordenam os elementos na memória principal do computador. Já os algoritmos ***externos*** ordenam os elementos armazenados em um dispositivo externo de armazenamento, como disco ou fita. Neste último caso, o acesso aos elementos é caro em termos de recursos computacionais e deve ser levado em consideração na elaboração do algoritmo.

Algoritmos de ordenação

- Outra classificação divide os algoritmos em *estáveis* e *instáveis*. Um algoritmo é dito estável se a ordem relativa dos elementos que possuem a mesma chave é mantida durante a ordenação.

- Há ainda a divisão dos métodos de ordenação em:
  - *Métodos simples*: de fácil implementação, com complexidade $O(n^2)$.
  - *Métodos eficientes*: de complexidade $O(n \log n)$, com estratégia e implementação mais complexas.

- Uma observação importante é que os algoritmos de ordenação são sempre analisados pelo *número de comparações e movimentações* que realizam, pois estas são as operações primordiais nesta classe de algoritmos.

Algoritmos de ordenação – Método da bolha

- Provavelmente o algoritmo de ordenação mais fácil de se implementar (e o mais ineficiente) é o algoritmo conhecido como *método da bolha*.

- O algoritmo é denominado desta forma devido ao comportamento que o mesmo impõe aos elementos da sequência, que se assemelham a bolhas subindo gradualmente para a superfície.

- Neste algoritmo, elementos adjacentes são testados e trocados caso estejam fora de ordem. Fazendo isto repetidamente, resulta em colocar o maior elemento na última posição. Realizando um segundo atravessamento da lista, sob o mesmo raciocínio, o segundo maior elemento acabará na penúltima posição. Se aplicarmos esta ideia por n-1 atravessamentos na lista, a mesma estará ordenada.

Algoritmos de ordenação – Método da bolha

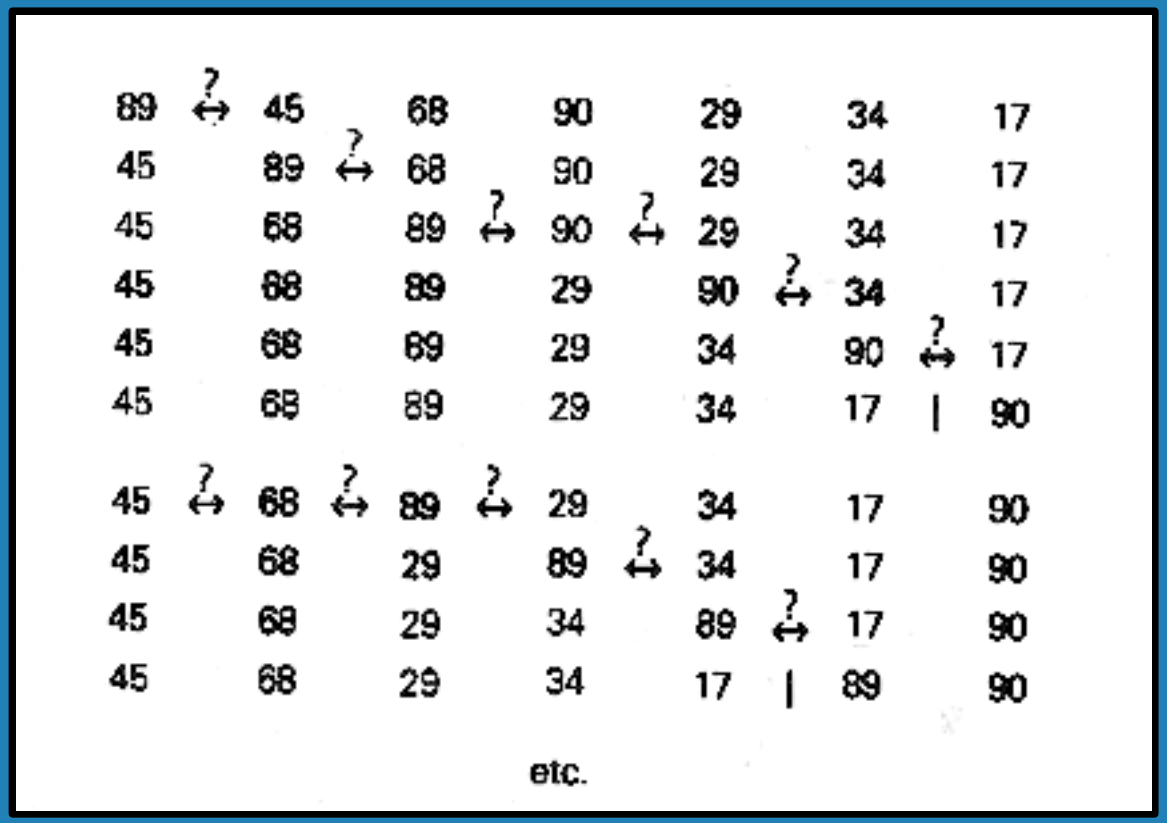
- Segue o algoritmo da bolha na forma de pseudocódigo:

ALGORITHM *BubbleSort*$(A[0..n-1])$

//Sorts a given array by bubble sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in ascending order
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow 0$ **to** $n-2-i$ **do**
        if $A[j+1] < A[j]$ swap $A[j]$ and $A[j+1]$

Algoritmos de ordenação – Método da bolha

- Exemplo:



- Este algoritmo é estável?

Método da bolha – análise de eficiência

- Tamanho da entrada: número de elementos (n) do arranjo.
- Operação básica: vamos considerar a comparação.
- Seja $C(n)$ o número de vezes que a comparação é executada:

$$C(n)=\sum_{i=0}^{n-2}\sum_{j=0}^{n-2-i}1=\sum_{i=0}^{n-2}(n-2-i-0+1)=\sum_{i=0}^{n-2}(n-1-i)=\frac{(n-1)n}{2}.$$

- Complexidade assintótica:

$$\frac{n^2}{2}-\frac{n}{2}\leqslant\frac{n^2}{2}+\frac{n^2}{2}(se\ n\geqslant 0)\leqslant n^2+n^2=2n^2$$

Assim, tomando-se c=2 e n_0=0:

$$\frac{n^2}{2}-\frac{n}{2}\leqslant 2n^2, n\geqslant 0$$

- Conclusão: o método da bolha, para o número de comparações que realiza, tem complexidade $O(n^2)$, ou seja, é quadrático com relação à entrada.

Algoritmos de ordenação – Seleção

- A ideia do algoritmo de *ordenação por seleção* é atravessar a lista para encontrar o menor elemento e trocá-lo com o primeiro. Após isto, iniciando-se do segundo elemento, aplica-se a mesma ideia, ou seja, seleciona-se o menor elemento dentre os que estão do segundo em diante e este é trocado com o segundo elemento. Em geral, na i-ésima passagem ($0 \leq i \leq n$-2), deve-se encontrar o menor elemento em A[i..n-1] e trocá-lo com $A[i]$.

Elementos em suas posições finais ***Os últimos n-i elementos***

$$A[0] \leq \ldots \leq A[i\text{-}1] \mid A[i], \ldots, min, \ldots, A[n\text{-}1]$$

- Seguindo com esta abordagem, os elementos estarão ordenados após n-1 atravessamentos.

Algoritmos de ordenação – Seleção

- Segue o algoritmo:

ALGORITHM *SelectionSort*$(A[0..n-1])$

```
//Sorts a given array by selection sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in ascending order
for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
        if A[j] < A[min]   min ← j
    swap A[i] and A[min]
```

Algoritmos de ordenação – Seleção

ALGORITHM *SelectionSort*$(A[0..n-1])$
//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in ascending order
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$

Exemplo:

| 89 | 45 | 68 | 90 | 29 | 34 | 17
17 | 45 | 68 | 90 | 29 | 34 | 89
17 | 29 | 68 | 90 | 45 | 34 | 89
17 | 29 | 34 | 90 | 45 | 68 | 89
17 | 29 | 34 | 45 | 90 | 68 | 89
17 | 29 | 34 | 45 | 68 | 90 | 89
17 | 29 | 34 | 45 | 68 | 89 | 90

➔ Este algoritmo é estável?

Ordenação por seleção - análise de eficiência

- Tamanho da entrada: o número de elementos n;
- Operação básica: a comparação `A[j] < A[min]`;
- O número de vezes que a operação é executada será dado pelo somatório:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1)-(i+1)+1] = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}.$$

→ Portanto, a ordenação por seleção é $O(n^2)$ para a operação de comparação.

- Note, no entanto, que o número de movimentações é apenas $O(n)$. Esta propriedade distingue este algoritmo de maneira bastante positiva para situações em que as movimentações possam ser custosas.

Algoritmos de ordenação – Inserção

- Para entender a estratégia do algoritmo de *ordenação por inserção*, basta imaginar uma situação em que se quer ordenar uma pilha de cartas. Pode-se fazer da seguinte forma. Dado um conjunto de cartas já ordenado, a próxima carta a ser inserida deve ser inserida no conjunto de maneira também ordenada. Assim, o conjunto cresce de uma unidade e se mantem ordenado. Segue-se fazendo desta forma, para cada carta que ainda não foi incluída no conjunto ordenado.

- No início, o conjunto ordenado está vazio, então a primeira carta selecionada pode ser simplesmente considerada o conjunto ordenado inicial. A partir da segunda carta, basta seguir a ideia acima.

Algoritmos de ordenação – Inserção

- Segue o algoritmo:

ALGORITHM *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

Algoritmos de ordenação – Inserção

- Exemplo: ordenando 6, 4, 1, 8, 5

6 | 4 1 8 5
4 6 | 1 8 5
1 4 6 | 8 5
1 4 6 8 | 5
1 4 5 6 8

- Este algoritmo é estável?

Ordenação por inserção - análise de eficiência

- Tamanho da entrada: o número de elementos n;
- Operação básica: vamos escolher a primeira atribuição do laço mais interno (que corresponde à movimentação de um elemento);
- O número de vezes que a movimentação é executada será dado pelo somatório:

$$M(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

→ Portanto, a ordenação por inserção é $O(n^2)$ para o número de movimentações que realiza.

- É considerado o melhor algoritmo para uso geral, dentre os métodos de ordenação mais elementares.

Exercícios

1) Invente um tipo composto, por exemplo `Aluno`, que tenha vários campos e adapte os algoritmos passados para ordenar elementos deste tipo. Note que neste caso você precisará eleger um campo para ser a chave.

2) Complete o exemplo dado para o algoritmo da Bolha.

3) Faça o mesmo que foi feito no exemplo do algoritmo da Bolha (ou seja, ilustrar os atravessamentos do algoritmo) para a lista de caracteres: E, X, A, M, P, L, E.

4) O algoritmo da Bolha é estável?

Exercícios

5) Com relação ao algoritmo da Bolha:

a) Prove que se nenhuma troca é realizada em um certo atravessamento, então a lista está ordenada e o algoritmo pode ser interrompido.

b) Escreva o novo pseudocódigo (ou mesmo a nova função C++) para incorporar este melhoramento.

c) Para este novo algoritmo, qual é a complexidade de pior caso?

Exercícios

6) Faça o mesmo que foi feito no exemplo do algoritmo por seleção (ou seja, ilustrar os atravessamentos do algoritmo) para a lista de caracteres: E, X, A, M, P, L, E.

7) O algoritmo por seleção é estável?

8) Faça o mesmo que foi feito no exemplo do algoritmo por inserção (ou seja, ilustrar os atravessamentos do algoritmo) para a lista de caracteres: E, X, A, M, P, L, E.

9) O algoritmo por inserção é estável?

Exercícios

10) Se a lista de entrada do algoritmo já estiver ordenada, o que você diria a respeito do número de comparações / movimentações realizado por cada um dos algoritmos mostrados?

11) Faça a mesma análise acima, mas imaginando uma lista de entrada com os elementos em ordem decrescente (assumindo-se que se quer ordenar crescentemente).