

BCC202 - Estruturas de Dados I

Aula 13: Ordenação: Bubble, Selection e Insertion Sort

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: silvap@ufop.edu.br

2021



Conteúdo

Introdução

- Conceitos Básicos de Ordenação
- Estável e Não Estável
- Interna e Externa

BubbleSort

SelectionSort

InsertionSort

Considerações Finais

Bibliografia

Conteúdo

Introdução

Conceitos Básicos de Ordenação
Estável e Não Estável
Interna e Externa

BubbleSort

SelectionSort

InsertionSort

Considerações Finais

Bibliografia

O que é um método de ordenação?

- ▶ **Ordenar**: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- ▶ A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
 - ▶ Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.
- ▶ As ordens mais usadas são a **numérica** e a **lexicográfica**.

Característica Estável e Não Estável

- ▶ Qualquer tipo de chave sobre o qual exista uma regra de ordenação bem-definida pode ser utilizado.
- ▶ Um método de ordenação é **estável** se a ordem relativa dos itens com chaves iguais não se altera durante a ordenação.
 - ▶ Alguns dos métodos mais eficientes não são estáveis.
 - ▶ A estabilidade pode ser forçada quando o método é não-estável.
 - Sedgewick (1988) sugere agregar um pequeno índice a cada chave antes de ordenar, ou então aumentar a chave de alguma outra forma.

Característica Interna e Externa

- ▶ Os métodos de ordenação podem ser **classificados** como:
 - ▶ Ordenação **Interna**: o arquivo a ser ordenado cabe todo na memória principal.
 - ▶ Qualquer registro pode ser imediatamente acessado.
 - ▶ Ordenação **Externa**: o arquivo a ser ordenado **não** cabe todo na memória principal.
 - ▶ Registros são acessados sequencialmente ou em blocos.

Características

- ▶ A maioria dos métodos de ordenação é baseada em **comparações** das chaves.
- ▶ Existem métodos de ordenação que utilizam o princípio da **distribuição**.
 - ▶ Exemplo: ordenar um baralho com 52 cartas pelo valor da carta e pelo naipe.
 1. Primeiro separe as cartas em treze montes (valores das cartas).
$$A < 2 < 3 < \dots < 10 < J < Q < K$$
 2. Em seguida, colete os montes na ordem desejada.
 3. Distribua cada monte em quatro montes (naipes das cartas).
$$\clubsuit < \diamondsuit < \spadesuit < \heartsuit$$
 4. Colete os montes na ordem desejada.
 - ▶ Qual é o custo deste algoritmo?

Critérios de análise

- ▶ Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:
 - ▶ Número de comparações $C(n)$ entre chaves.
 - ▶ Número de movimentações $M(n)$ de registros do arquivo.
- ▶ O uso econômico da memória disponível é um requisito primordial na ordenação interna.
- ▶ Métodos de ordenação **in situ** são os preferidos.
- ▶ Métodos que utilizam listas encadeadas não são muito utilizados.
- ▶ Métodos que fazem cópias dos itens a serem ordenados implicam num maior consumo de memória.

Ordenação Interna por Comparação

- ▶ Métodos **simples**:
 - ▶ Adequados para pequenos arquivos.
 - ▶ Requerem $O(n^2)$ comparações.
- ▶ Métodos **eficientes**:
 - ▶ Adequados para arquivos maiores.
 - ▶ Requerem $O(n \log n)$ comparações.
 - ▶ Implementações das comparações e trocas são mais complexas.

O melhor algoritmo

Atenção

- ▶ Não existe um método de ordenação considerado universalmente superior a todos os outros.
- ▶ É necessário analisar o problema e, com base nas características dos dados, decidir qual método melhor se aplica à ele.

O que estudaremos?

- ▶ Neste curso estudaremos apenas algoritmos de **ordenação interna** e que utilizam o princípio da **comparação**.
- ▶ Nesta aula:
 - ▶ **BubbleSort.**
 - ▶ **SelectionSort.**
 - ▶ **InsertionSort.**

O problema de ordenação

Ordenar dados, ordem crescente ou decrescente, ordenação numérica ou lexicográfica, considerando **chaves** dos registros.

- ▶ Ordenar alunos considerando a data em que ingressaram na universidade.
- ▶ Ordenar alunos considerando os CRs.

O problema da ordenação

Para facilitar, veremos os algoritmos para ordenar números inteiros.

Entrada

Uma sequência de n números $\langle a_1, a_2, \dots, a_n \rangle$.

Saída

Uma permutação (reordenação) $\langle (a')_1, (a')_2, \dots, (a')_n \rangle$ da sequência de entrada, tal que $(a')_1 \leq (a')_2 \leq \dots \leq (a')_n$.

Conteúdo

Introdução

Conceitos Básicos de Ordenação
Estável e Não Estável
Interna e Externa

BubbleSort

SelectionSort

InsertionSort

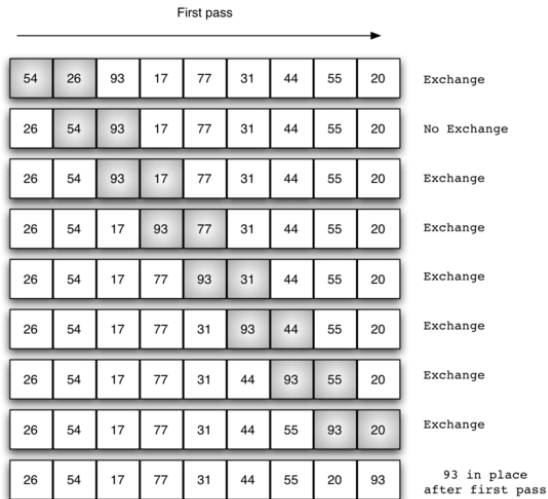
Considerações Finais

Bibliografia

Funcionamento

- ▶ Os elementos vão **"borbulhando"** a cada iteração do método até a posição correta para ordenação da lista.
- ▶ Como os elementos são trocados (borbulhados) frequentemente, há um alto custo com troca de elementos.
- ▶ O *bubblesort* é um algoritmo de ordenação popular, porém ineficiente.
 - ▶ Funciona permutando repetidamente elementos adjacentes que estão fora de ordem.

Ilustração: Primeira Iteração



Ilustração

A seguir, mais detalhes sobre o passo a passo do *bubblesort*:



Animação



Vídeo

Pseudocódigo

A seguir, o pseudocódigo do *BubbleSort*.

1 **Algorithm:** BUBBLESORT

Input: int* v, int n

2 **begin**

3 **for** $i \leftarrow 0$ **to** $i < n$ **do**

4 **for** $j \leftarrow 1$ **to** $j < n - i$ **do**

5 **if** $(v[j] < v[j - 1])$ **then**

6 trocar $v[j]$ com $v[j - 1]$

7 **end**

8 **end**

9 **end**

10 **end**

Tempo, Comparação e Espaço

BubbleSort é um algoritmo estável.

- ▶ Complexidade de tempo no pior caso
 - ▶ $O(n^2)$ comparações
 - ▶ $O(n^2)$ trocas
- ▶ Complexidade de tempo no melhor caso
 - ▶ $O(n^2)$ comparações
 - ▶ 0 trocas
 - ▶ $O(n^2) + 0 = O(n^2)$
- ▶ Complexidade de espaço/consumo de espaço
 - ▶ Extra: $O(1)$

Características

Vantagens

- ▶ Algoritmo **simples**.
- ▶ Algoritmo **estável**.

Desvantagens

- ▶ O fato de o arquivo já estar ordenado não ajuda a reduzir o número de comparações (o custo continua quadrático), porém o número de movimentações cai para zero.

1 Algorithm: BUBBLESORT-Modificado**Input:** int* v, int n**2 begin****3 for $i \leftarrow 0$ to $i < n$ do****4 troca = 0****5 for $j \leftarrow 1$ to $j < n - i$ do****6 if $(v[j] < v[j - 1])$ then****7 trocar $v[j]$ com $v[j-1]$** **8 troca \leftarrow troca + 1****9 end****10 end****11 if troca == 0 then****12 interrompe a execução do *bubblesort*****13 end****14 end****15 end**

Tempo, Comparação e Espaço

BubbleSort modificado é um algoritmo estável com as seguintes características.

- ▶ Complexidade de tempo no pior caso
 - ▶ $O(n^2)$ comparações
 - ▶ $O(n^2)$ trocas
- ▶ Complexidade de tempo no melhor caso
 - ▶ Vetor em ordem crescente:
 - ▶ $O(n)$ comparações
 - ▶ 0 trocas
 - ▶ $O(n) + 0 = O(n)$
- ▶ Complexidade de espaço/consumo de espaço
 - ▶ Extra: $O(1)$

Conteúdo

Introdução

Conceitos Básicos de Ordenação
Estável e Não Estável
Interna e Externa

BubbleSort

SelectionSort

InsertionSort

Considerações Finais

Bibliografia

Visão Geral

- ▶ Seleciona do n -ésimo menor (ou maior) elemento da lista.
- ▶ Troca do n -ésimo menor (ou maior) elemento com a n -ésima posição da lista.
- ▶ É realizada apenas uma única troca a cada iteração.

Visão Geral

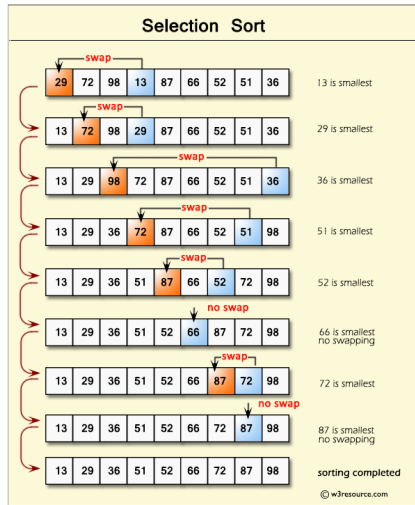
Mantemos um subvetor $A[0...i-1]$ tal que:

- ▶ $A[0...i-1]$ está ordenado
- ▶ $A[0...i-1] < A[i...n-1]$

Repetições

- ▶ A cada passo selecionamos o menor elemento em $A[i...n-1]$ e o colocamos em $A[i]$
- ▶ Repetimos o processo para $i = 0, \dots, n-1$ para ordenar o vetor A .
 - ▶ $a[k]$ é o menor valor de $a[i..n-1]$
 - ▶ trocar $a[i]$ e $a[k]$
 - ▶ $a[0...i]$ está ordenado

Ilustração



Ilustração

A seguir, mais detalhes sobre o passo a passo do *selection sort*.



Animação



Vídeo

Pseudocódigo

A seguir, o pseudocódigo do *SelectionSort*.

1 **Algorithm:** SELECTIONSORT

Input: int* v , int n

2 **begin**

3 **for** $i \leftarrow 0$ **to** $i < n$ **do**

4 $min \leftarrow i$

5 **for** $j \leftarrow i + 1$ **to** $j < n$ **do**

6 **if** $v[j] < v[min]$ **then**

7 $min \leftarrow j$

8 **end**

9 **end**

10 trocar $v[i]$ com $v[min]$

11 **end**

12 **end**

Tempo, Comparação e Espaço

Selection Sort é um algoritmo não estável com as seguintes características.

- ▶ **Complexidade de tempo no pior caso**
 - ▶ $O(n^2)$ comparações
 - ▶ $O(n)$ trocas
 - ▶ $O(n^2) + O(n) = O(n^2)$
- ▶ **Complexidade de tempo no melhor caso**
 - ▶ Vetor em ordem crescente:
 - ▶ $O(n^2)$ comparações
 - ▶ $O(n)$ trocas
 - ▶ $O(n^2) + O(n) = O(n^2)$
- ▶ **Complexidade de espaço/consumo de espaço**
 - ▶ Extra: $O(1)$

Características

Vantagens

- ▶ Custo linear no tamanho da entrada para o número de movimentos de registros.
- ▶ É o algoritmo a ser utilizado para arquivos com registros muito grandes (alto custo de movimentação).
- ▶ É muito interessante para arquivos pequenos.

Desvantagens

- ▶ O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo continua quadrático.
- ▶ O algoritmo **não é estável**.

Algoritmo Modificado

A seguir, uma melhoria implementada no *selectionsort*.

1 **Algorithm:** SELECTIONSORT_MODIFICADO

Input: int* v, int n

2 **begin**

3 **for** $i \leftarrow 0$ **to** $i < n$ **do**

4 $min \leftarrow i$

5 **for** $j \leftarrow i + 1$ **to** $j < n$ **do**

6 **if** $v[j] < v[min]$ **then**

7 $min \leftarrow j$

8 **end**

9 **end**

10 **if** $i \neq min$ **then**

11 trocar $v[i]$ com $v[min]$

12 **end**

13 **end**

14 **end**

Vantagem

Não realiza trocas quando $i == min$ (um elemento por ele próprio).

Tempo, Comparação e Espaço

SelectionSort é um algoritmo com as seguintes características.

- ▶ **Complexidade de tempo no pior caso**
 - ▶ $O(n^2)$ comparações
 - ▶ $O(n)$ trocas
 - ▶ $O(n^2) + O(n) = O(n^2)$
- ▶ **Complexidade de tempo no melhor caso**
 - ▶ $O(n^2)$ comparações
 - ▶ 0 trocas
 - ▶ $O(n^2) + 0 = O(n^2)$
- ▶ **Complexidade de espaço/consumo de espaço**
 - ▶ Extra: $O(1)$

Conteúdo

Introdução

Conceitos Básicos de Ordenação
Estável e Não Estável
Interna e Externa

BubbleSort

SelectionSort

InsertionSort

Considerações Finais

Bibliografia

Funcionamento: Analogia

- ▶ Algoritmo utilizado pelo jogador de cartas:
 - ▶ As cartas são ordenadas da esquerda para direita uma por uma.
 - ▶ O jogador escolhe a segunda carta e verifica se ela deve ficar antes da carta existente ou depois.
 - ▶ Depois a terceira carta é inserida, deslocando-a até sua posição correta.
 - ▶ O procedimento é repetido até ordenar todas as cartas.
- ▶ Considerando um vetor, alto custo para inserir um elemento na posição correta (*arrastar os elementos necessários*).

Ilustração

- ▶ A cada passo mantemos o subvetor $A[0...j-1]$ ordenado
- ▶ inserimos o elemento $A[j]$ neste subvetor
- ▶ repetimos o processo para $j = 1, \dots, n-1$
- ▶ ordenamos o vetor.

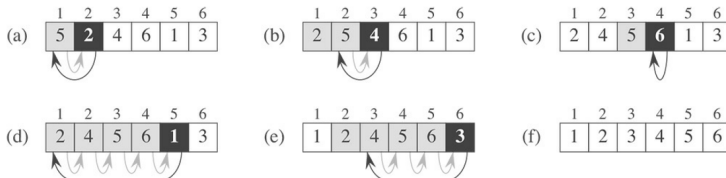


Figura: Figura extraída (Cormen et al., 2011)

Ilustração

A seguir, ilustrações do passo a passo do *insertionsort*.



Animação



Vídeo

Pseudocódigo

A seguir, o pseudocódigo do *InsertionSort*.

```
1  Algorithm: INSERTIONSORT
   Input: int* v, int n
2  begin
3      for  $i \leftarrow 1$  to  $i < n$  do
4           $aux \leftarrow v[i]$ 
5           $j \leftarrow i - 1$ 
6          while  $j \geq 0$  &  $aux < v[j]$  do
7               $v[j + 1] \leftarrow v[j]$ 
8               $j \leftarrow j - 1$ 
9          end
10          $v[j + 1] \leftarrow aux$ 
11     end
12 end
```

Considerações

O **InsertionSort** costuma ser uma escolha mais adequada nas seguintes situações [Souza et. al].

- ▶ Para vetores com no máximo 10 elementos
- ▶ Para vetores quase ordenados.

Algoritmos super eficientes assintoticamente tende a fazer muitas movimentações [Souza et. al].

Tempo, Comparação e Espaço

Insertion é um algoritmo estável com as seguintes características.

- ▶ **Complexidade de tempo no pior caso**
 - ▶ $O(n^2)$ comparações
 - ▶ $O(n^2)$ trocas
 - ▶ $O(n^2) + O(n^2) = O(n^2)$
- ▶ **Complexidade de tempo no melhor caso**
 - ▶ Vetor em ordem crescente:
 - ▶ $O(n)$ comparações
 - ▶ 0 trocas
 - ▶ $O(n) + 0 = O(n)$
- ▶ **Complexidade de espaço/consumo de espaço**
 - ▶ Extra: $O(1)$

Características

- ▶ O número mínimo de comparações e movimentos ocorre quando os itens estão originalmente em ordem.
- ▶ O número máximo ocorre quando os itens estão originalmente na ordem reversa.
- ▶ É o método a ser utilizado quando o arquivo está “quase” ordenado.
- ▶ É um bom método quando se deseja adicionar uns poucos itens a um arquivo ordenado, pois o custo é linear.
- ▶ O algoritmo de ordenação por inserção é **estável**.

Conteúdo

Introdução

Conceitos Básicos de Ordenação
Estável e Não Estável
Interna e Externa

BubbleSort

SelectionSort

InsertionSort

Considerações Finais

Bibliografia

Conclusão

- ▶ Nesta aula tivemos o primeiro contato com *algoritmos de ordenação*.
- ▶ Foram vistos os algoritmos **BubbleSort**, **SelectionSort** e **InsertionSort**.
- ▶ Cada algoritmo possui suas particularidades:
 - ▶ Não existe um método de ordenação considerado universalmente superior a todos os outros.
 - ▶ É necessário analisar o problema e, com base nas características dos dados, decidir qual método melhor se aplica à ele.

Conclusão

- Quadro comparativo dos métodos de ordenação:

Algoritmo	Comparações			Movimentações			Espaço	Estável	In situ
	Melhor	Médio	Pior	Melhor	Médio	Pior			
Bubble	$O(n^2)$			$O(n^2)$			$O(1)$	Sim	Sim
Selection	$O(n^2)$			$O(n)$			$O(1)$	Não*	Sim
Insertion	$O(n)$	$O(n^2)$		$O(1)$	$O(n^2)$		$O(1)$	Sim	Sim

* Existem versões estáveis.

Próxima Aula

Mergesort.

Conteúdo

Introdução

Conceitos Básicos de Ordenação
Estável e Não Estável
Interna e Externa

BubbleSort

SelectionSort

InsertionSort


Considerações Finais

Bibliografia

Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados de:

- ▶ **Slides MO417 - Complexidade de Algoritmos I**, elaborados por Cid Carvalho de Souza, Cândida Nunes da Silva e Orlando Lee e revisado por Zanoni Dias em agosto de 2011, <https://www.ic.unicamp.br/~zanoni/teaching/mo417/2011-2s/aulas/handout/04-ordenacao.pdf>. Acessado em 2021.

 Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford.

Introduction to Algorithms.

The MIT Press, 2011.

Exercício

- ▶ Dada a sequência de números: 3 4 9 2 5 1 8.
- ▶ Ordene em ordem **crescente** utilizando os três algoritmos aprendidos em sala (**BubbleSort**, **SelectionSort** e **InsertionSort**), apresentando a sequência dos números a cada passo (Teste de Mesa).