INSTITUTO FEDERAL DE MATO GROSSO
CAMPUS RONDONÓPOLIS
urso Superior de Análise e Desenvolvimento de Sistemas

Estrutura de Dados e Análise de Algoritmos

Prof. Daniel Domingos Alves

daniel.alves@ifmt.edu.br

Métodos de Ordenação: Bubble Sort
31/10/2024

Introdução

- Ordenar corresponde ao processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente
- O objetivo principal da ordenação é facilitar a recuperação posterior de itens do conjunto ordenado
  - Exemplo: catálogo telefônico, dicionários, índices de livros, tabelas, arquivos, etc.

Introdução

- O conceito de um conjunto ordenado de elementos tem **considerável impacto** sobre nossa vida cotidiana.

- Exemplos:
  - Localizar um número telefônico em catálogo;
  - Procurar um livro em biblioteca tradicional ou virtual;
  - Saber qual o próximo documento a ser impresso pela impressora em um departamento da empresa;
  - Saber qual o próximo processo a ser executado pelo processador de uma máquina qualquer; etc...

Classificação (Sorting)

- Processo de organizar itens em **ordem** (de)crescente, segundo algum critério.
- Também chamado de **ordenação**.
- Aplicações que utilizam-se de dados classificados
  - Preparação de dados para facilitar pesquisas futuras
    - Exemplo: dicionários e listas telefônicas
  - Agrupar itens que apresentam mesmos valores
    - Para eliminação dos elementos repetidos

Conceitos básicos

- Ordenação
  - Ato de colocar um conjunto de dados em uma determinada ordem predefinida
  - Fora de ordem
    - 5, 2, 1, 3, 4
  - Ordenado
    - 1, 2, 3, 4, 5 **OU** 5, 4, 3, 2, 1
- Algoritmo de ordenação
  - Coloca um conjunto de elementos em uma certa ordem

Conceitos básicos

- A ordenação permite que o acesso aos dados seja feito de forma mais eficiente
  - É parte de muitos métodos computacionais
    - Algoritmos de busca, intercalação/fusão, utilizam ordenação como parte do processo
    - Aplicações em geometria computacional, bancos de dados, entre outras necessitam de listas ordenadas para funcionar

Conceitos básicos

- A ordenação é baseada em uma chave
  - A chave de ordenação é o **campo** do item utilizado para comparação
    - Valor armazenado em um *array* de inteiros
    - Campo nome de uma *struct*
    - etc.
  - É por meio dela que sabemos se um determinado elemento está a frente ou não de outros no conjunto

Conceitos básicos

- Podemos usar qualquer tipo de chave
  - Deve existir uma regra de ordenação bem-definida
- Alguns tipos de ordenação
  - numérica
    - 1, 2, 3, 4, 5
  - lexicográfica (ordem alfabética)
    - Ana, André, Bianca, Ricardo

Conceitos básicos

- Independente do tipo, a ordenação pode ser
  - Crescente
    - 1, 2, 3, 4, 5
    - Ana, André, Bianca, Ricardo
  - Decrescente
    - 5, 4, 3, 2, 1
    - Ricardo, Bianca, André, Ana

Conceitos básicos

- Os algoritmos de ordenação podem ser classificados como de
  - **Ordenação interna**
    - O conjunto de dados a ser ordenado cabe todo na memória principal (RAM)
    - Qualquer elemento pode ser imediatamente acessado

Conceitos básicos

- Os algoritmos de ordenação podem ser classificados como de
  - **Ordenação externa**
    - O conjunto de dados a ser ordenado não cabe na memória principal
    - Os dados estão armazenados em memória secundária (por exemplo, um arquivo)
    - Os elementos são acessados sequencialmente ou em grandes blocos

Conceitos básicos

- Além disso, a ordenação pode ser estável ou não
  - Um algoritmo de ordenação é considerado **estável** se a ordem dos elementos com chaves iguais não muda durante a ordenação
  - O algoritmo preserva a **ordem relativa** original dos valores

Conceitos básicos

- Exemplo
  - Dados não ordenados
    - **5a**, 2, **5b**, 3, 4, 1
    - **5a** e **5b** são o mesmo número
  - Dados ordenados
    - 1, 2, 3, 4, 5a, 5b: ordenação **estável**
    - 1, 2, 3, 4, 5b, 5a: ordenação **não-estável**

Métodos de ordenação

- **Algoritmos de ordenação** nos trazem a oportunidade de ordenar dados, permitindo-nos responder as seguintes perguntas:
  - “quais os clientes que mais compraram esse ano?”;
  - “qual processo será enviado ao processador para ser processado?”;
  - “qual arquivo será impresso na impressora de um departamento qualquer da empresa?”;
  - etc...

Métodos de ordenação

- Os métodos de ordenação estudados podem ser divididos em
  - Básicos
    - Fácil implementação
    - Auxiliam o entendimento de algoritmos complexos
  - Sofisticados
    - Em geral, melhor desempenho

Análise de Desempenho

- A eficiência de tempo é calculada pelo **número de operações críticas efetuadas**.

- **Operações críticas**: (**1**) comparação entre chaves; (**2**) movimentação de registros ou de ponteiros para registros; (**3**) troca de dois registros.

Principais Categorias de Métodos

- Ordenação por Trocas
- Ordenação por Seleção
- Ordenação por Inserção
- Ordenação por Intercalação

Ordenação por Trocas

- Caracteriza-se pela comparação aos **pares de chaves**, trocando-as de posição caso estejam fora de ordem no par.

- Principais algoritmos
  - BubbleSort (Bolha)
  - QuickSort

Algoritmo Bubble Sort

- Também conhecido como ordenação por bolha
  - É um dos algoritmos de ordenação mais conhecidos que existe

Algoritmo Bubble Sort

Funcionamento

- Compara pares de valores adjacentes e os troca de lugar se estiverem na ordem errada
  - Trabalha de forma a movimentar, uma posição por vez, o maior valor existente na porção não ordenada de um *array* para a sua respectiva posição no *array* ordenado
- Esse processo se repete até que mais nenhuma troca seja necessária
  - Elementos já ordenados

BubbleSort (Método da Bolha)

- Compara todos os pares consecutivos (adjacentes no vetor) de chaves, realizando troca caso necessário.
- Realiza um certo número de varreduras sobre o vetor a ser ordenado.
- O procedimento **termina** quando, em uma dada varredura, **nenhuma troca de chaves ocorre** ou **após n – 1 varreduras (**sendo **n** o nº de elementos a ordenar**).**

Vídeo – Bubblesort

Exemplo – BubbleSort (1/3)

Suponha que se deseja classificar em ordem crescente o seguinte vetor de chaves [28, 26, 30, 24, 25].

Primeira Varredura

28	26	30	24	25	compara par (28, 26): **troca**
26	28	30	24	25	compara par (28, 30): não troca
26	28	30	24	25	compara par (30, 24): **troca**
26	28	24	30	25	compara par (30, 25): **troca**
26	28	24	25	**30**	Maior chave em sua posição definitiva

fim da primeira varredura

Exemplo – BubbleSort (2/3)

Vetor inicial de chaves [28, 26, 30, 24, 25].

Resultado do fim da primeira varredura
26 28 24 25 30

Segunda Varredura

26 28 24 25 30 compara par (26, 28) : não troca
26 28 24 25 30 compara par (28, 24) : **troca**
26 24 28 25 30 compara par (28, 25) : **troca**
26 24 25 **28** **30** (não precisa comparar)

fim da segunda varredura

Exemplo – BubbleSort (3/3)

Vetor de chaves [28, 26, 30, 24, 25] a ser ordenado.

Resultado do fim da segunda varredura
26 24 25 28 30

Terceira Varredura

26 24 25 28 30 compara par (26, 24) : **troca**
24 26 25 28 30 compara par (26, 25) : **troca**
24 25 **26** **28** **30** (não precisa comparar)
Fim da terceira varredura

Durante a quarta varredura, nenhuma troca ocorrerá e a execução do algoritmo terminará.

Exercícios

Considerando o seguinte vetor:

[25, 48, 37, 12, 57, 86, 33, 92]

1) Realize a ordenação do vetor utilizando o método **BubbleSort**. Quantas operações críticas (**comparações** + **trocas**) foram necessárias?
2) Quantas varreduras são necessárias para detectar que o vetor acima está classificado?

BubbleSort (1ª versão)

```
(1) procedimento BubbleSort (A : vetor, N: inteiro)
(2)       para i de 0 até N passo 1 faça
(3)             para j de 0 até N passo 1 faça
(4)                   se (A[j] > A[j+1]) então
(5)                         aux → A[j];
(6)                         A[j] → A[j+1];
(7)                         A[j+1] → aux;
(8)                   fimse;
(9)             fimpara;
(10)      fimpara;
```

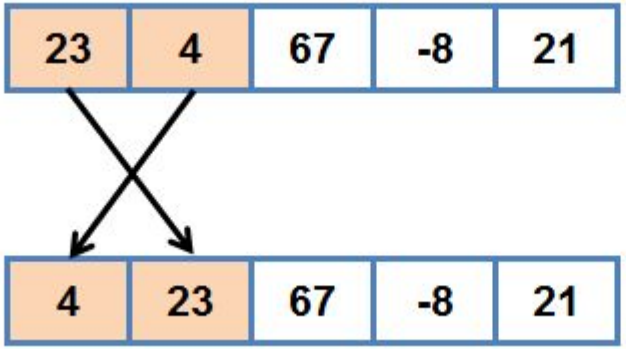
Obs.: O algoritmo continua realizando comparações com as chaves do vetor que estão parcialmente ordenados.
Como melhorar?

Algoritmo Bubble Sort

Algoritmo

```
41
42  void bubbleSort(int *V  , int N){
43      int i, continua, aux, fim = N;
44      do{
45          continua = 0;
46          for(i = 0; i < fim-1; i++){
47              if (V[i] > V[i+1]){
48                  aux = V[i];
49                  V[i] = V[i+1];
50                  V[i+1] = aux;
51                  continua = i;
52              }
53          }
54          fim--;
55      }while(continua != 0);
56  }
```

Troca dois valores consecutivos no vetor



Algoritmo Bubble Sort

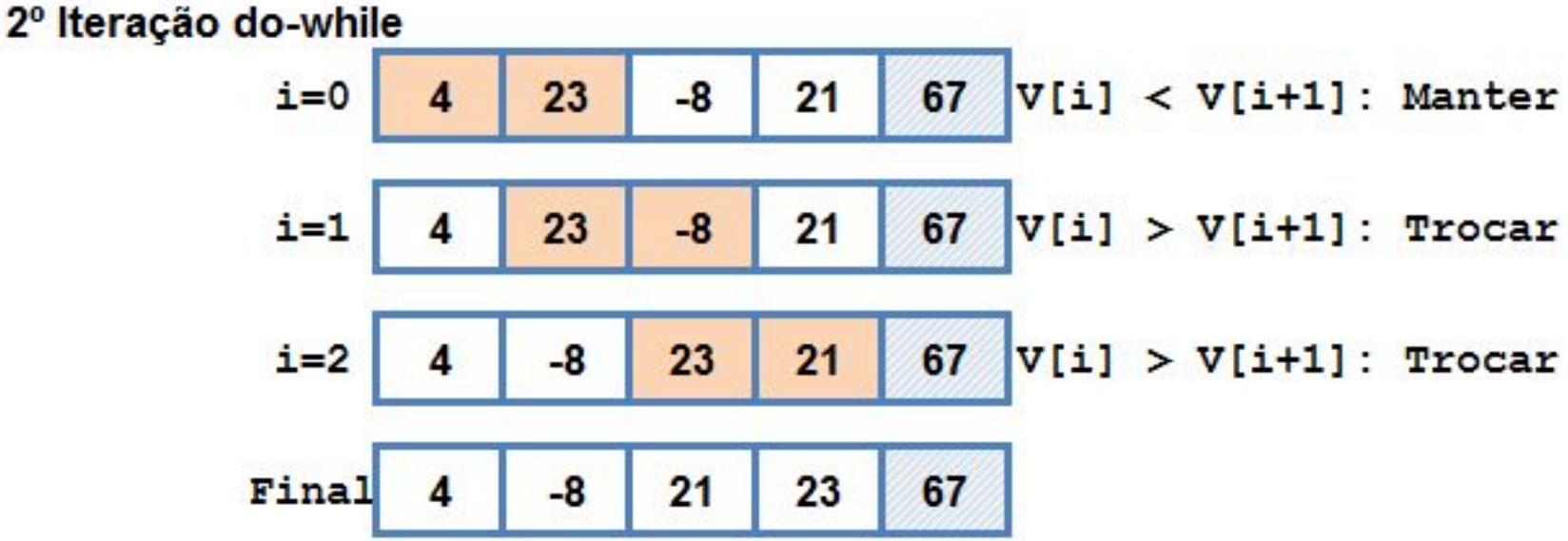
Passo a passo

1º iteração do-while: encontra o maior valor e o movimenta até a última posição

Algoritmo Bubble Sort

Passo a passo

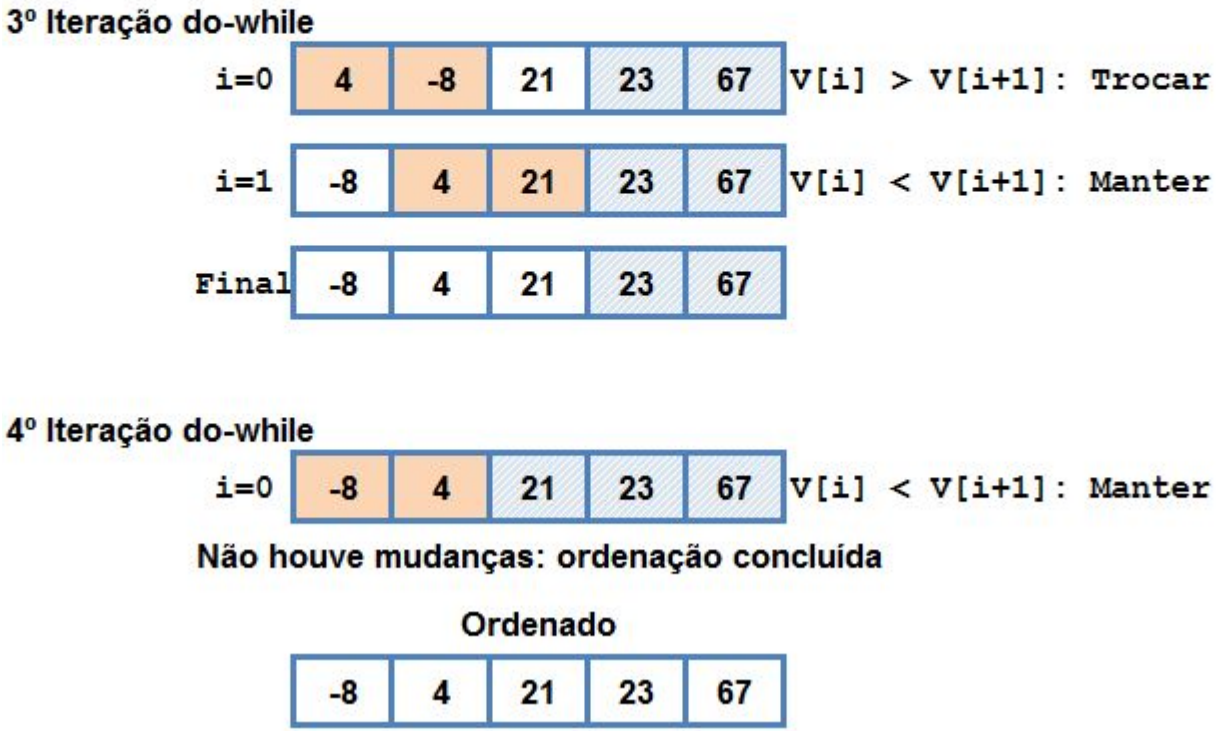
2º iteração do-while: encontra o segundo maior valor e o movimenta até a penúltima posição



Algoritmo Bubble Sort

Passo a passo

Processo continua até todo o array estar ordenado



Bubblesort - Análise de Desempenho (1/3)

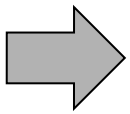
- **Melhor caso**
  - Quando o vetor já se encontra **ordenado** **nenhuma troca** ocorre na primeira varredura.
  - **Custo linear**: n - 1 comparações

- **Pior caso**
  - Quando o vetor se encontra na ordem **inversa a desejada**.
  - A cada varredura **apenas uma chave** será colocada em sua **posição definitiva**.

Bubblesort - Análise de Desempenho (2/3)

Pior caso

Qtde de Varreduras	Comparações efetuadas	Trocas efetuadas
1	n - 1	n - 1
2	n - 2	n - 2
3	n - 3	n - 3
...	...	...
n - 1	1	1
	$(n^2 - n)/2$	$(n^2 - n)/2$

Comparações = $(n^2 - n)/2$  $O(n^2)$

Bubblesort - Análise de Desempenho (3/3)

- Número de comparações entre chaves e movimentações de registros, pior caso:

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

- Método muito simples, mas custo alto
  - Adequado apenas se arquivo pequeno
  - Ruim se registros muito grandes
  - Número de operações não se altera se vetor já está (parcialmente) ordenado

Algoritmo Bubble Sort

- Vantagens
  - Simples e de fácil entendimento e implementação
  - Está entre os métodos de ordenação mais difundidos existentes
- Desvantagens
  - Não é um algoritmo eficiente
    - Sua eficiência diminui drasticamente à medida que o número de elementos no array aumenta
    - É estudado apenas para fins de desenvolvimento de raciocínio

Algoritmo Bubble Sort

- Complexidade
  - Considerando um array com **N** elementos, o tempo de execução é:
    - ***O(N)***, melhor caso: os elementos já estão ordenados.
    - ***O(N^2)***, pior caso: os elementos estão ordenados na ordem inversa.
    - ***O(N^2)***, caso médio.