Noções de Classificação

Objetivos e Caracterizações

O acesso a um conjunto de dados é facilitado se o mesmo está armazenado conforme uma certa ordem, baseada num critério conhecido.

O objetivo básico da classificação ou ordenação é *facilitar o acesso aos dados*, propiciando a localização rápida de uma entrada, por um algoritmo. O interesse na ordenação se justifica por sua aplicação na adequação das tabelas (vetores de registros) à consulta.

UNIVASF

Objetivos e Caracterizações

Além disso, o desenvolvimento de métodos e processos de ordenação também se constitui em campo de pesquisa na aplicação de técnicas básicas de construção de algoritmos.

Talvez não haja outro problema de programação que, para uma mesma formulação (no caso: "ordenar uma ***lista***"), tenha tantas soluções diferentes, cada uma com qualidades e defeitos, relacionados com desempenho (complexidade em tempo e uso da memória), facilidade de programação, tipo e condição inicial dos dados, etc. .

Obs.: O termo ***lista*** será utilizado, deste ponto em diante, com o sentido de generalizar o termo ***vetor***.

A operação de classificação (ordenação) pode ser entendida como um rearranjo (ou *permutação*) de uma lista, por exemplo, do menor para o maior, aplicado sobre um dos campos (a chave da ordenação).

Os processos de classificação diferem quanto à área de trabalho necessária. Enquanto uns métodos trabalham sobre o próprio vetor a ordenar (métodos *in situ*), outros exigem um segundo vetor para armazenar o resultado, e até mesmo outras áreas auxiliares.

É interessante notar que nem sempre o uso de áreas auxiliares aumenta a eficiência dos processos, pois elas podem servir apenas para simplificar as operações, aumentando, no entanto, o número de operações a realizar.

Objetivos e Caracterizações

A eficiência dos processos pode ser avaliada pelo número C de operações de comparação entre chaves conjuntamente com o número M de movimentos de dados (transposições) necessários.

Tanto C como M são funções de n, o número de chaves.

Daí, têm-se os processos ditos ***diretos,*** assim chamados por serem de formulação relativamente simples, de manipulação ***in situ***.

Nesses processos o normal é acontecer um grande número de operações de comparação e de troca de valores.

Objetivos e Caracterizações

Existem também os processos *avançados*, que são aqueles de formulação baseada em expedientes não tão óbvios, com algoritmos, portanto, mais elaborado.

Nesses métodos as operações tendem a ser mais complexas, porém há menos trocas, o que explica seu melhor desempenho.

Por outro lado, os diversos processos de ordenação partem de alguma proposta básica, a qual caracteriza uma *família* de métodos.

Os diversos processos se diferenciam numa família conforme os meios e truques que usam para realizar a proposta básica.

UNIVASF

Objetivos e Caracterizações

Assim, os métodos diretos *in situ*, particularmente, pertencem às famílias de processos de ordenação por *troca*, por *inserção* e por *seleção*.

Os métodos avançados podem ser refinamentos ou combinações de processos diretos ou podem implementar outras propostas, identificando-se famílias como da ordenação por *intercalação,* por *particionamento* e por *distribuição de chaves*.

UNIVASF

Classificação

Exercício:

Construa um módulo que receba, como parâmetros, um vetor de inteiros, com no máximo 100 elementos, e o número de elementos neste vetor. O módulo em questão deve ordenar o vetor colocando seus elementos em ordem crescente.

UNIVASF

Classificação por Troca

Toda ordenação está baseada na permutação dos elementos do vetor; Logo, sempre dependerá de trocas.

São, no entanto, ditos processos por troca aqueles em que a operação de troca é dominante.

Analisaremos agora um método de classificação por troca, conhecido como classificação por troca simples, classificação por bolha ou bubble sort.

UNIVASF

Classificação por Troca - bubble sort

A ideia básica por trás do bubble sort é percorrer a lista sequencialmente várias vezes.

Cada passagem consistem em comparar cada elemento na lista com seu sucessor e trocar os dois elementos se eles não estiverem na ordem correta.

Para uma melhor compreensão examinaremos o seguinte exemplo:

25 57 48 37 12 92 86 33

UNIVASF

Classificação por Troca - bubble sort

x = { 25, 57, 48, 37, 12, 92, 86, 33 }

As seguintes comparações são feitas na primeira passagem:

x[0] com x[1] (25 com 57) nenhuma troca

x[1] com x[2] (57 com 48) troca

x[2] com x[3] (57 com 37) troca

x[3] com x[4] (57 com 12) troca

x[4] com x[5] (57 com 92) nenhuma troca

x[5] com x[6] (92 com 86) troca

x[6] com x[7] (92 com 33) troca

O que vocês percebem que ocorreu nesta passagem?

UNIVASF

Classificação por Troca - bubble sort

Conjunto completo de iterações:

iteração 0 (lista original)	25 57 48 37 12 92 86 33
iteração 1	25 48 37 12 57 86 33 92
iteração 2	25 37 12 48 57 33 86 92
iteração 3	25 12 37 48 33 57 86 92
iteração 4	12 25 37 33 48 57 86 92
iteração 5	12 25 33 37 48 57 86 92
iteração 6	12 25 33 37 48 57 86 92
iteração 7	12 25 33 37 48 57 86 92

Exercício 53:

Com base no que foi visto, construa um módulo que recebe, como parâmetros, um vetor de inteiros, com no máximo 100 elementos, e o número de elementos neste vetor. O módulo em questão deve ordenar o vetor por meio da aplicação do bubble sort.

```
procedimento bubble_sort (var v: vetor
[1..100] de inteiro; n: inteiro)
var i, j, temp: inteiro
inicio
   para i de 1 ate n-1 faca
      para j de 1 ate n-1 faca
         se (v[j]>v[j+1]) entao
            temp <- v[j]
            v[j] <- v[j+1]
            v[j+1] <- temp
         fimse
      fimpara
   fimpara
fimprocedimento
```

Classificação por Troca - bubble sort

Pode se fazer melhorias no algoritmo anterior?

Sim.

Quais?

- Não existe necessidade de verificar o sub vetor ordenado;

- e ao se verificar que o vetor está ordenado pode-se parar a ordenação.

UNIVASF

Classificação por Troca - bubble sort

Exercício 54:

Implemente o algoritmo com as melhorias discutidas.

```
procedimento bubble_sort (var v: vetor [1..100] de
 inteiro; n: inteiro)
var i, j, temp: inteiro
      nao_ocorreu_troca: logico
inicio
   para i de 1 ate n-1 faca
     nao_ocorreu_troca <- verdadeiro
     para j de 1 ate n-i faca
        se (v[j]>v[j+1]) entao
            temp <- v[j]
            v[j] <- v[j+1]
            v[j+1] <- temp
            nao_ocorreu_troca <- falso
        fimse
     fimpara
     se (nao_ocorreu_troca) entao
        interrompa
     fimse
   fimpara
fimprocedimento
```