

Ordenação com Quick Sort

Mário Leite

...

A classificação de dados para gerar uma saída mais inteligível e eficaz é uma técnica fundamental na análise de dados, pois permite ao usuário final se inteirar melhor da informação gerada para a tomada de decisão. Essa classificação pode ser crescente ou decrescente; dependendo da maneira mais adequada ao interesse do usuário. Por exemplo, para analisar a lista dos alunos de uma turma essa classificação deve ser crescente, por nome; no entanto, para analisar a classificação dos alunos num teste vestibular pela nota obtida é mais interessante que a lista esteja em ordem decrescente, pois as maiores notas serão as primeiras. Na maioria das linguagens de programação existem comandos e funções internas para isto; em linguagens 4GL como SQL, por exemplo, para listar todos alunos em ordem decrescente de média: **SELECT * FROM Alunos ORDER BY media DEC**. A cláusula **DEC** se encarrega de fazer essa classificação em ordem decrescente.

Na prática, a técnica mais utilizada pelos programadores iniciantes é o “Método da Bolha” (*Bubble Sort*) por ser a mais conhecida e a mais fácil de empregar. Esta técnica se baseia na troca de posições dos elementos de uma lista, usando uma *variável* auxiliar. O algoritmo abaixo é a base desse tipo de ordenação em ordem crescente de valores, sendo **Vet[i]** e **Vet[j]** elementos genéricos da lista.

```
Para i De 1 Até (n-1) Faça
  Para j De (i+1) Até n Faça
    Se (Vet[i]>Vet[j]) Então
      Aux ← Vet[i]
      Vet[i] ← Vet[j]
      Vet[j] ← Aux
    FimSe
  FimPara
FimPara
```

Aqui ocorrem as trocas,
se forem necessárias.

Entretanto, a técnica do “Método da Bolha” não é eficiente; aliás, é a mais ineficiente entre todas as técnicas de classificação/ordenação. Por isto, para grandes quantidades de dados (da ordem de dezenas de milhares) outras técnicas devem ser empregadas; e uma dessas técnicas é a “Quick Sort”, que é uma das mais rápidas que existem. Esse método se baseia no princípio de “dividir para conquistar”, o que significa que divide o problema em subproblemas menores; resolve esses subproblemas e depois combina as suas soluções para obter a solução final em quatro etapas básicas, assim resumidas.

1) Escolha do pivô: O algoritmo seleciona um elemento da lista como *pivô*, em torno do qual os outros elementos serão rearranjados.

2) Particionamento: A lista é particionada em duas sublistas: uma contendo elementos menores que o *pivô* e outra contendo elementos maiores que o *pivô*.

3) Recursão: É aplicada recursividade às sublistas geradas pelo particionamento; ou seja, o algoritmo é aplicado às sublistas menores criadas até que toda a lista esteja ordenada.

Em resumo: Essencialmente, o “Quick Sort” divide o problema original em subproblemas menores; resolve cada subproblema independentemente e depois combina as soluções para obter toda a lista ordenada.

Algoritmo "OrdenaQuickSort"

//Programa modular contendo um procedimento recursivo que faz ordenação crescente
//de um vetor de inteiros pelo método "Quick Sort"

//Autor: Mário Leite

//E-mail: marleite@gmail.com

//-----

//Elementos globais

Const MaxTam=1000 *//limite o tamanho do vetor*

Var Vet: vetor[1..MaxTam] **de** inteiro
k, TamVet: inteiro

//-----

Procedimento **ProQuickSortRec**(inic,fim:inteiro)

//Faz a ordenação recursiva.

var i, j, pivot, aux: inteiro

Inicio

{Inicia o algoritmo Quick Sort}

i <- inic *//índice do primeiro elemento do vetor*

j <- fim *//índice do último elemento do vetor*

pivot <- (inic+fim) **div** 2 *//índice do elemento "pivot"*

Enquanto (i<j) **Faca**

Enquanto (Vet[i]<Vet[pivot]) **Faca**

 i <- i + 1

FimEnquanto

Enquanto (Vet[j]>Vet[pivot]) **Faca**

 j <- j - 1

FimEnquanto

Se (i<=j) **Entao**

 aux <- Vet[i]

 Vet[i] <- Vet[j]

 Vet[j] <- aux

 i <- i + 1

 j <- j - 1

FimSe

FimEnquanto

{Faz a chamada recursiva do procedimento}

Se (j>inic) **Entao**

ProQuickSortRec(inic,j)

FimSe

Se (i<fim) **Entao**

ProQuickSortRec(i,fim)

FimSe

FimProcedimento *//fim do procedimento "ProQuickSortRec"*

//=====

//Programa principal

Inicio

Repita

Escreva("Digite o número de elementos do vetor: ")

Leia(TamVet)

 TamVet <- **Int**(TamVet)

Ate ((TamVet>0) **e** (TamVet<=MaxTam))

Escreval("")

{Leitura dos elementos do vetor}

Para k **De** 1 **Ate** TamVet **Faca**

Escreva("Digite o elemento",k, " do vetor: ")

Leia(Vet[k])

 Vet[k] <- **Int**(Vet[k])

FimPara

LimpaTela

```
Escreval("Vetor original")
Para k De 1 Ate TamVet Faca
    Escreva(Vet[k], " ")
FimPara
ProQuickSortRec(1,TamVet)  //chama procedimento para fazer a ordenação
Escreval("")
Escreval("")
Escreval("Vetor ordenado")
Para k De 1 Ate TamVet Faca
    Escreva(Vet[k], " ")
FimPara
Escreval("")
FimAlgoritmo  //Fim do programa "OrdenaQuickSort" -----
```