

Métodos 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 programa. O interesse na ordenação se justifica assim diretamente por sua aplicação na adequação das listas (tabelas) à consulta.

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. .

A operação de classificação ou 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 complexidade dos algoritmos serve igualmente para classificar os processos de ordenação. 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*, que apresentam, no entanto, complexidade $O(n^2)$.

Objetivos e Caracterizações

Nesses processos o normal é acontecer um grande número de operações de comparação e de troca de valores. Já os processos *avançados*, são aqueles de formulação baseada em expedientes não tão óbvios, com algoritmos, portanto, mais elaborados, que alcançam desempenho de ordem $O(n \log n)$. 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. 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.

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.

Classificação por Troca - bubble sort

A idéia básica por trás do bubble sort é percorrer a lista seqüencialmente 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

Classificação por Troca - bubble sort

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

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

Classificação por Troca - bubble sort

Com base no que foi visto, construa uma função, em C, que recebe um vetor de inteiros e o número de elementos neste vetor. Esta função deve ordenar o vetor implementando o bubble sort.

```
void bubble_sort (int v[ ], int n)  
{  
    int i, j;  
    for (i=0; i<n-1; i++)  
        for (j=0; j<n-1; j++)  
            if (v[j]>v[j+1])  
                {  
                    int temp;  
                    temp = v[j];  
                    v[j] = v[j+1];  
                    v[j+1] = temp;  
                } }
```

Classificação por Troca - bubble sort

Qual a complexidade do algoritmo anterior?

$O(n^2)$

Pode se fazer melhorias no algoritmo anterior? Caso possa, quais?

Sim. Não existe necessidade de verificar o sub vetor ordenado e ao se verificar que o vetor está ordenado pode-se parar a ordenação.

Classificação por Troca - bubble sort

Implemente o algoritmo com as melhorias discutidas. Depois analise a complexidade do mesmo.

```
void bubble_sort (int v[], int n)
{
    int i, j, ah_troca=1;
    for (i=0; i<n-1 && ah_troca; i++)
    {
        ah_troca = 0;
        for (j=0; j<n-i-1; j++)
            if (v[j]>v[j+1])
            {
                int temp;
                temp = v[j];
                v[j] = v[j+1];
                v[j+1] = temp;
                ah_troca = 1;
            }
    }
}
```

$O(n^2)$

Classificação por Troca - bubble sort

Existem outras formas de aprimorar o *bubble sort*. Uma delas é fazer com que as passagens sucessivas percorram o vetor em sentidos opostos, de modo que os elementos de menor magnitude se desloquem mais rapidamente para o início da lista da mesma forma que os de maior magnitude se desloquem para o final. Implementaremos agora, como exercício, uma função que implemente este método.

```

void shaker_sort (int v[ ], int n)
{
    int esq = 0, dir = n-1, i, trocou;
    do
    {
        trocou = 0;
        for (i=esq; i<dir; i++)
            if (v[i]>v[i+1])
            {
                int temp;
                temp = v[i];
                v[i] = v[i+1];
                v[i+1] = temp;
                trocou = 1;
            }
        dir--;
    }

```

```

if (trocou)
{
    trocou = 0;
    for (i=dir; i>esq; i--)
        if (v[i]<v[i-1])
            {
                int temp;
                temp = v[i];
                v[i] = v[i-1];
                v[i-1] = temp;
                trocou = 1;
            }
    }
    esq++;
}while(esq<dir && trocou);

```

400 }

Classificação por Troca - bubble sort

Esta variante é conhecida como troca alternada ou *shaker sort*, devido a propiciar, por assim dizer, acomodação por “agitação” das chaves. O ganho obtido se deve a que: a) vão se formando dois subvetores ordenados que podem sair do processo de comparação e trocas; b) para vetores pouco desordenados, também as chaves menores são logo posicionadas, detectando-se em seguida o fim do processo.

Classificação por Troca - bubble sort

De qualquer forma, o ganho é apenas em relação ao número de comparações: as trocas a serem efetuadas são sempre lado a lado, e todos os elementos terão que se movimentar no mesmo número de casas que no método *bubble sort*. Como as comparações são menos onerosas que as trocas, estas continuam a ditar o desempenho: o método *shaker sort*, no melhor caso (vetor ordenado) e no pior caso (vetor invertido) a mesma complexidade que o *bubble sort*.