

Método dos Incrementos Decrescentes

Pode-se alcançar uma otimização mais significativa sobre a classificação por inserção direta do que a inserção binária ou de lista, usando-se a **classificação de Shell** (ou **classificação de incremento decrescente**), assim denominada em homenagem a seu descobridor Donald Shell. Esse método classifica subvetores separados do vetor original. Esses subvetores contêm todo k ésimo elemento do vetor original. O valor de k é chamado **incremento**.

Método dos Incrementos Decrescentes

Por exemplo, se k é 5, o subvetor consistindo em $v[0]$, $v[5]$, $v[10]$,... é classificado primeiro. Cinco subvetores, cada um contendo um quinto dos elementos do arquivo original, são classificados dessa maneira. São eles:

Subvetor 0 -> $v[0]$ $v[5]$ $v[10]$...

Subvetor 1 -> $v[1]$ $v[6]$ $v[11]$...

Subvetor 2 -> $v[2]$ $v[7]$ $v[12]$...

Subvetor 3 -> $v[3]$ $v[8]$ $v[13]$...

Subvetor 4 -> $v[4]$ $v[9]$ $v[14]$...

Método dos Incrementos Decrescentes

O i -ésimo elemento do j -ésimo subvetor é $v[(i * k + j)]$. Se um incremento k diferente for escolhido, os k subvetores serão divididos de modo que o i -ésimo elemento do j -ésimo subvetor seja $v[(i * k + j)]$.

Depois que os primeiros k subvetores estiverem classificados (geralmente por inserção simples), será escolhido um novo valor menor que k e o vetor será novamente particionado em novos conjuntos de subvetores. Cada um desses subvetores maiores será classificado e o processo se

repetirá novamente com um número ainda menor que k . Em algum momento, o valor de k será definido com 1, de modo que o subvetor consistindo no vetor inteiro será classificado. Uma seqüência decrescente de incrementos é determinada no início do processo inteiro. O último valor nessa seqüência deve ser 1.

Por exemplo, se o vetor original for:

75 25 95 87 64 59 86 40 16 49

e a seqüência (5, 2, 1) for escolhida, os seguintes subvetores serão classificados em cada iteração:

Método dos Incrementos Decrescentes

| k | vetor resultante | | | | | | | | | |
|---|------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 5 | <u>59</u> | 25 | <u>40</u> | <u>16</u> | <u>49</u> | <u>75</u> | 86 | <u>95</u> | <u>87</u> | <u>64</u> |
| 2 | <u>40</u> | <u>16</u> | <u>49</u> | <u>25</u> | <u>59</u> | <u>64</u> | 86 | <u>75</u> | <u>87</u> | <u>95</u> |
| 1 | <u>16</u> | <u>25</u> | <u>40</u> | <u>49</u> | 59 | 64 | <u>75</u> | <u>86</u> | 87 | 95 |

Com base no que foi discutido, codifique uma função que receba um vetor (de inteiros) e o número de elementos no mesmo e através do método *shell sort* ordene de forma crescente os elementos do vetor.

OBS.: Inicialize o incremento com $n/2$ e faça-o decrescer nesta taxa.

```
void shell_sort (int v[ ], int n) {  
    int inc, j, k, y;  
    inc=n;  
    do{  
        inc=inc>1?inc/2:1;  
        for (j=inc; j<n; j++)  
        {  
            y=v[j];  
            for (k=j-inc; k>=0 && y<v[k]; k-=inc)  
                v[k+inc]=v[k];  
            v[k+inc]=y;  
        }  
    }while(inc>1); }
```

Método dos Incrementos Decrescentes

A análise da eficiência da classificação de Shell é complicada, em termos matemáticos, e está além do escopo desta disciplina. No entanto, uma observação se faz importante, o desempenho do processo pode ser prejudicado, se os incrementos forem múltiplos entre si, pois cadeias logicamente adjacentes segundo incrementos múltiplos podem gerar passos redundantes. A probabilidade disto acontecer é baixa.

Classificação por Intercalação

Assim como a classificação quicksort, veremos agora, mais um método avançado, ou seja, outro método que emprega expedientes mais elaborados, visando, principalmente, apressar a resolução do problema pela diminuição do número de trocas necessárias. Isto é alcançado *a)* pela subdivisão do problema da ordenação em subproblemas menores ou *b)* pelo uso de áreas auxiliares, evitando o problema da liberação de casas ocupadas por elementos fora do lugar para receberem chaves relocizadas (que ocorre nos métodos *in situ* em geral). Consegue-se, nesse grupo, desempenho de ordem $O(n \log n)$.

Classificação por Intercalação

Analisaremos então, o método de **intercalação** ou *merge sort*. Uma primeira aproximação ao mesmo é propiciada por um estudo da sua aplicação sobre uma *lista*, em vez de um vetor. Para classificar uma lista, pode-se adotar o seguinte procedimento:

1. **se** a lista tem um só elemento, já está ordenada;
2. **senão**, dividir a lista em duas sublistas, ordená-las e intercalá-las.

Classificação por Intercalação

Os subproblemas que se apresentam são:

- i. ordenar uma sublista;
- ii. intercalar duas listas.

A ordenação das sublistas é resolvida pela aplicação recursiva da solução do problema principal. A intercalação de duas listas ordenadas consiste apenas em se inserir numa lista resultante cada um dos elementos das listas originais na ordem correta.

Classificação por Intercalação

Para tanto, as listas podem ser vistas como filas: os elementos do início são comparados e o menor deles é retirado da fila correspondente e levado para a lista resposta. Quando uma fila esvaziar, simplesmente copia-se o restante da outra.

Por exemplo: ao se intercalar as listas $L1 = (A D E K L P)$ e $L2 = (B C F G)$ a seqüência de movimentos seria a seguinte:

Classificação por Intercalação

Intercalação de listas ordenadas:

| L1 | L2 | lista resultante |
|---------------|-----------|-----------------------|
| (A D E K L P) | (B C F G) | (A) |
| (D E K L P) | (B C F G) | (A B) |
| (D E K L P) | (C F G) | (A B C) |
| (D E K L P) | (F G) | (A B C D) |
| (E K L P) | (F G) | (A B C D E) |
| (K L P) | (F G) | (A B C D E F) |
| (K L P) | (G) | (A B C D E F G) |
| (K L P) | () | (A B C D E F G K L P) |

Observe como se aplicariam as duas fases (subdivisão e intercalação) ao se ordenar o vetor exemplo (armazenado numa lista). Em cada fase, os trechos resultantes de subdivisão estão sublinhados e os resultantes de intercalação estão em negrito.

Exemplo: Ordenação de lista por intercalação

| | |
|--------------------------------|---|
| Lista original | (75 25 95 87 64 59 86 40 16 49) |
| <u>Subdivisão</u> | <u>(75 25 95 87 64)</u> <u>(59 86 40 16 49)</u> |
| <u>Subdivisão</u> | <u>(75 25 95)</u> <u>(87 64)</u> <u>(59 86 40)</u> <u>(16 49)</u> |
| <u>Subdivisão</u> | <u>(75 25)</u> <u>(95)</u> <u>(87)</u> <u>(64)</u> <u>(59 86)</u> <u>(40)</u> <u>(16)</u> <u>(49)</u> |
| <u>Subdivisão/intercalação</u> | <u>(75)</u> <u>(25)</u> (64 87) <u>(59)</u> <u>(86)</u> (16 49) |
| Intercalação | (25 75) (59 86) |
| Intercalação | (25 75 95) (40 59 86) |
| Intercalação | (25 64 75 87 95) (16 40 49 59 86) |
| Intercalação | (16 25 40 49 59 64 75 86 87 95) |

Tendo como base o que foi discutido, um possível algoritmo para implementar a classificação de uma lista através do merge sort seria:

```
void merge_sort (LISTA_ENC *pl) {  
    if (tam(*pl)>1) {  
        LISTA_ENC L1, L2;  
        cria_lista(&L1);  
        cria_lista(&L2);  
        subdivide (pl, &L1, &L2);  
        merge_sort (&L1);  
        merge_sort (&L2);  
        intercala (L1, L2, pl); } }
```

Classificação por Intercalação - Exercício

Como exercício, implemente as funções subdivide e intercala, constantes no algoritmo anterior.