Programação 2

Classificação por Inserção

Instituto Federal do Espírito Santo - Ifes
Campus Colatina
Professor: Giovany Frossard Teixeira

Definição

- A classificação por inserção é caracterizada pelo principio no qual os n dados a serem ordenados são divididos em 2 segmentos:
  - *Um já ordenado e outro a ser ordenado*
- Num momento inicial, o primeiro segmento é formado por apenas um elemento, que, portanto, pode ser considerado já ordenado. O segundo segmento é formado pelos n-1 elementos restantes. A partir daí, o processo se desenvolve em n-1 iterações, sendo que em cada uma delas um elemento do segmento não ordenado é inserido no segmento ordenado.

Insertion Sort – Inserção Direta

- **Insertion sort**, ou *ordenação por inserção direta*, é um algoritmo de ordenação, eficiente quando aplicado a um pequeno número de elementos. Em termos gerais, ele percorre um vetor de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados.
- O algoritmo de inserção direta funciona da mesma maneira com que muitas pessoas ordenam cartas em um jogo de baralho como o pôquer.

Insertion Sort

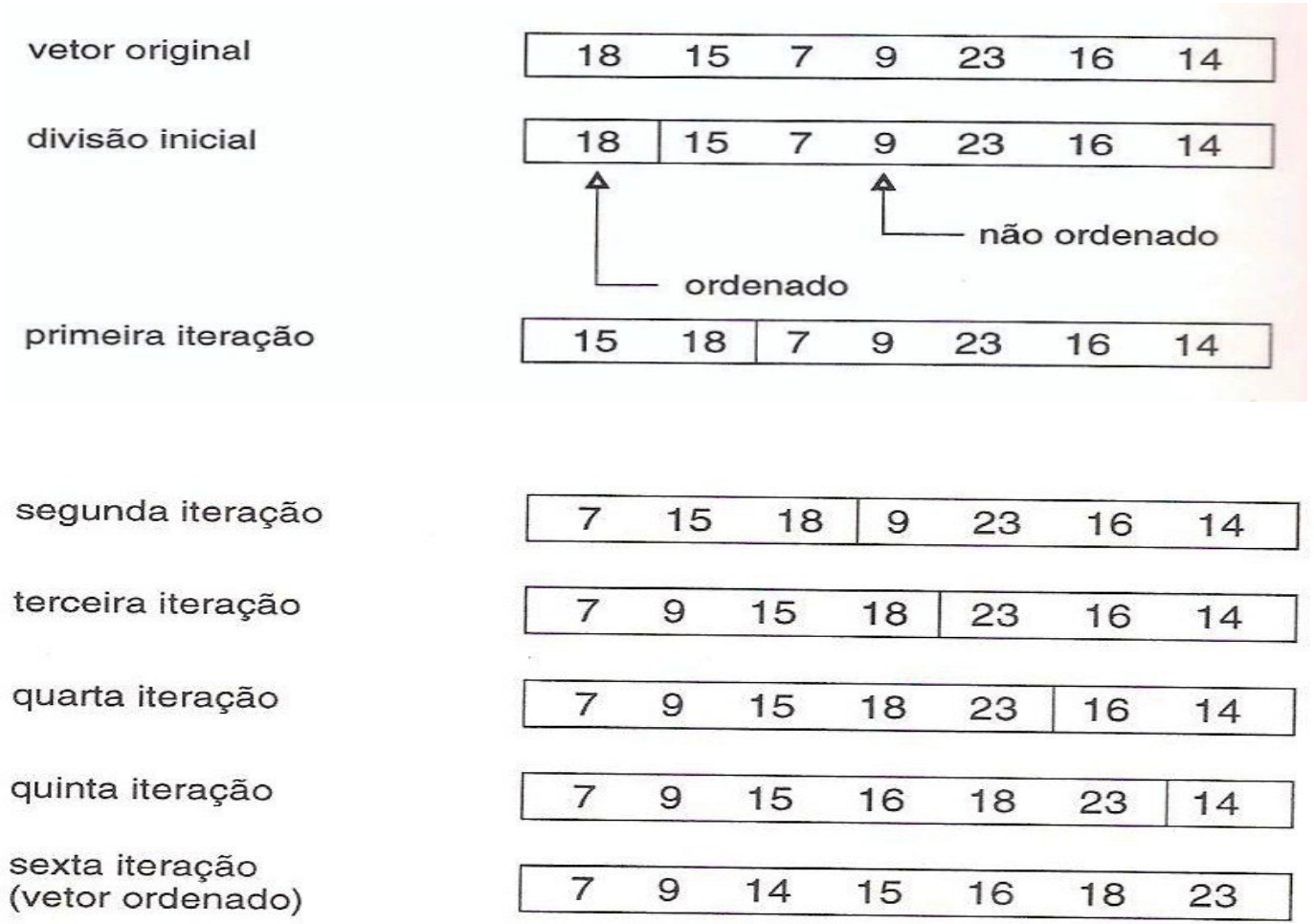
```
void insertionSort(int *vetor, int n){
    // Valor a ser inserido
    int valor;
    int k, j;
    for(int i=1; i<n; i++) { // i: posição do elemento a ser inserido
        valor = vetor[i];
        k = 0;                  // k é o inicio do segmento ordenado
        j = i-1;                // j é o fim do segmento ordenado
        while ( (j>=0) && (k==0) ){
            if (valor < vetor[j]) {
                vetor[j+1] = vetor[j];
                j = j - 1;
            } else {
                k = j+1;      // posição da inserção
            }
        }
        vetor[k] = valor;
    }
}
```

Exemplo

```
void escreveVetor(int *vetor, int n){
    for(int i=0; i<n; i++)
        printf("vet[%d] vale %d \n", i, vetor[i]);
    printf("\n\n");
}


int _tmain(int argc, _TCHAR* argv[])
{
    int vet[7] = {18, 15, 7, 9, 23, 16, 14};
    insertionSort(vet, 7);
    escreveVetor(vet, 7);
    return 0;
}
```

Iterações do exemplo



Análise de Desempenho

- O desempenho do método de inserção direta é fortemente influenciado pela ordem inicial das chaves a serem ordenadas.
- O melhor caso ocorre quando temos o vetor ordenado em ordem crescente, pois **valor >= vetor[j]** fazendo com que **k** seja alterado e finalizando o while em O(1). Daí como temos um for de 1 a n essa parte do algoritmo é O(n). Assim pela regra do produto nosso algoritmo no melhor caso seria O(n).
- O pior caso ocorre quando temos o vetor ordenado em ordem decrescente, pois nesse caso o while precisaria percorrer por todo segmento ordenado para inserir cada elemento, esse segmento é proporcional a n e na sua última inserção será exatamente n-1. Assim é fácil ver que esse trecho é O(n), como o for é O(n) pela regra do produto nosso algoritmo é $O(n^2)$ no pior caso.

Análise de Desempenho

- O caso médio é semelhante ao pior caso. Suponhamos que a distância média dos valores nos locais incorretos seja n/2, ou seja, que é necessário fazermos n/2 trocas para inserir o elemento, isso é razoável pois a distância mínima é 1 e a máxima n-1.
- Se temos que fazer n/2 trocas, em geral, temos que esse trecho do algoritmo será O(n) pela regra da constante.
- Como o for é O(n) pela regra do produto nosso algoritmo é $O(n^2)$ no caso médio.

Shell Sort

- Esse método explora o fato de que o método de inserção direta apresentar um desempenho aceitável quando o número de elementos é pequeno e/ou estes já possuem uma ordenação parcial.
- A ideia consiste em dividir o vetor em *h* segmentos, de tal forma que cada segmento possua aproximadamente *n/h* elementos. Daí aplicamos o método de inserção direta (com pequenas adaptações) em cada segmento, tendo um tempo de ordenação do segmento em torno de $(n/h)^2$. Para classificarmos todos os *h* segmentos, o desempenho será proporcional a $h * (n/h)^2 = n^2/h$.
- Além disso, teremos rearranjado os elementos no vetor de modo que se aproximem do arranjo final, isto é, teremos feito uma ordenação parcial do vetor, o que contribui para a diminuição da complexidade do método.

Shell Sort

```
#include "stdafx.h"
#include <math.h>

void insertionSortAdaptado(int *vetor, int n, int f, int h){
    int valor;
    int k, j;
    for(int i=f+h; i<n; i = i+h) {
        valor = vetor[i];
        k = f;                  // k é o inicio do segmento ordenado
        j = i-h;                // j é o fim do segmento ordenado
        while ( (j>=f) && (k==f) ){
            if (valor < vetor[j]) {
                vetor[j+h] = vetor[j];
                j = j - h;
            } else {
                k = j+h;      // posição da inserção
            }
        }
        vetor[k] = valor;
    }
}


void shellSort(int *vetor, int n, int np){
    int h;
    for(int p = np; p>0; p--){                // número de passos (cada passo é uma organização de segmentos)
        h = (int)pow(2.0, (int)p-1);          // determina o tamnho do segmento (até que h=1)
        for (int j=0; j<h; j++)               // ordena os h segmentos
            insertionSortAdaptado(vetor, n, j, h);
    }
}
```

Exemplo

```
void escreveVetor(int *vetor, int n){
    for(int i=0; i<n; i++)
        printf("vet[%d] vale %d \n", i, vetor[i]);
    printf("\n\n");
}


int _tmain(int argc, _TCHAR* argv[])
{
    int vet[16] = {17, 25, 49, 12, 18, 23, 45, 38, 53, 42, 27, 13, 11, 28, 10, 14};
    shellSort(vet, 16, 3);
    escreveVetor(vet, 16);
    return 0;
}
```

Iterações do exemplo

Convenção: os números acima de cada célula indicam os índices das chaves no vetor. Os números abaixo de cada célula indicam o número do segmento ao qual cada célula pertence.

Primeiro passo: h = 4

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	**25**	**49**	**12**	**18**	**23**	**45**	**38**	**53**	**42**	**27**	**13**	**11**	**28**	**10**	**14**
1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4

No exemplo, os índices do vetor começam de 1, nossa implementação foi feita em C, portanto começamos de 0. Ou seja, analisando os indices nosso vetor iria de 0 a 15.

Iterações do exemplo

Após classificar cada um dos quatro segmentos, o vetor terá o aspecto a seguir.

Segundo passo: h = 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
11	**23**	**10**	**12**	**17**	**25**	**27**	**13**	**18**	**28**	**45**	**14**	**53**	**42**	**49**	**38**
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2

Iterações do exemplo

Após classificar cada um dos dois segmentos, o vetor terá o aspecto seguinte:

Terceiro (e último) passo: h = 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
10	**12**	**11**	**13**	**17**	**14**	**18**	**23**	**27**	**25**	**45**	**28**	**49**	**38**	**53**	**42**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Após a execução do último passo, o vetor resultará classificado:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
10	**11**	**12**	**13**	**14**	**17**	**18**	**23**	**25**	**27**	**28**	**38**	**42**	**45**	**49**	**53**

Análise do desempenho

- Segundo KNUTH, a análise do desempenho do algoritmo é muito complexa, pois identifica alguns problemas matemáticos bastante difíceis, alguns deles ainda não resolvidos. Um dos problemas é determinar o efeito que a ordenação dos segmentos em um passo produz nos passos subsequentes. Também não se conhece exatamente a melhor sequência de incrementos que produz melhor resultado.
- Como o Shell Sort é proporcional a n^2/h podemos dizer que ele é $O(n^2)$. Entretanto os efeitos da ordenação dos segmentos podem fazer com que esse limite seja mais baixo (por exemplo $O(n * (\log n)^2)$) mas como ainda não é possível provar isso ficamos com o limite dado por $O(n^2)$.

Bibliografia

- AZEREDO, Paulo A. **Métodos de Classificação de Dados**. Rio de Janeiro: Campus,1996.
- SANTOS, Henrique José. **Curso de Linguagem C da UFMG**, apostila.
- FORBELLONE, André Luiz. **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados.** São Paulo: MAKRON, 1993.