Ordenação: Introdução e métodos elementares

Algoritmos e Estruturas de Dados II

Ordenação

- Objetivo:
  - Rearranjar os itens de um vetor ou lista de modo que suas chaves estejam ordenadas de acordo com alguma regra.
- Estrutura:

```
typedef int TipoChave;

typedef struct {
     ChaveTipo chave;
     /* outros componentes */
} Item;
```

Critérios de Classificação

- Localização dos dados:
  - Ordenação interna: todas as chaves estão na memória principal.
  - Ordenação externa: chaves na memória principal e na memória secundária.
- Estabilidade:
  - Relacionado com o manutenção da ordem relativa entre chaves de mesmo valor.
  - Método é estável se a ordem relativa dos registros com a mesma chave não se altera após a ordenação.

Critérios de Classificação

- Uso da memória:
  - ***In place:*** transforma os dados de entrada utilizando apenas um espaço extra de tamanho constante.
- Movimentação dos dados:
  - Direta: registro todo é acessado e deve ser movido
  - Indireta: apenas as chaves são acessadas e ponteiros são rearranjados e não o registro todo.
- Adaptabilidade:
  - Sequência de operações executadas conforme a entrada.
  - Não adaptável: operações executadas independe da entrada.

Critérios de Avaliação

- Seja n o número de registros em um vetor, considera-se duas medidas de complexidade:
  - Número de comparações C(n) entre as chaves;
  - Número de trocas ou movimentações M(n) de itens;

```
#define Troca(A, B) {Item c = A; A = B; B = c; }

void Ordena(Item *v, int n) {
   int i, j;
   for (i = 0; i < n-1; i++) {
      for (j = n-1; j > i; j--) {
         if (v[j-1].chave > v[j].chave)  /* comparações */
            Troca(v[j-1], v[j]);  /* trocas */
}
```


Método Bolha

- Ideia:
  - Passa no arquivo e troca elementos adjacentes que estão fora de ordem, até os registros ficarem ordenados.
- Algoritmo
  - Supondo movimentação da esquerda para direita no vetor;
  - Quando o maior elemento do vetor for encontrado, ele será trocado até ocupar a última posição;
  - Na segunda passada, o segundo maior será movido para a penúltima posição do vetor.

E	X	E	M	P	L	O
E	E	X	M	P	L	O
E	E	M	X	P	L	O
E	E	M	P	X	L	O

E	E	M	P	L	X	O
E	E	M	P	L	O	X
E	E	M	P	L	O	X

Método Bolha

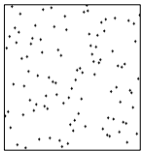
```
void Bolha (Item *v, int n) {
int i, j;

  for(i = 0; i < n-1; i++)
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave)
        Troca(v[j-1], v[j]);
}
```


Método Bolha

```
void Bolha (Item *v, int n) {
int i, j;

  for(i = 0; i < n-1; i++)
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave)
        Troca(v[j-1], v[j]);
}
```



E	X	E	M	P	L	O
E	E	M	P	L	O	X
E	E	M	L	O	P	X
E	E	L	M	O	P	X
E	E	L	M	O	P	X
E	E	L	M	O	P	X

E	E	L	M	O	P	X
E	E	L	M	O	P	X

Método Bolha: Complexidade

- Comparações – C(n):

```
void Bolha (Item *v, int n) {
int i, j;

  for(i = 0; i < n-1; i++)
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave)
        Troca(v[j-1], v[j]);
    }
}
```

$$C(n) = \sum_{i=0}^{n-2}\sum_{j=1}^{n-i} 1 = \sum_{i=0}^{n-2}(n-i-1) = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1$$

$$= n(n-1) - \frac{(n-1)(n-2)}{2} - (n-2+1)$$

$$= \frac{n^2-n}{2} = O(n^2)$$

Método Bolha: Complexidade

- Movimentações – M(n):

$$M(n) = C(n)$$

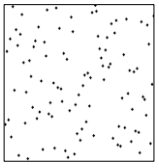
```
#define Troca(A, B) {Item c = A; A = B; B = c; }
```


Método Bolha

- Vantagens
  - Algoritmo simples
  - Algoritmo estável
- Desvantagens
  - Não adaptável
  - Muitas trocas de itens

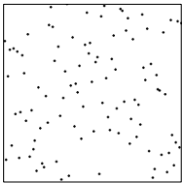
Variação Bolha: Ordenação Par-Ímpar

```
void ParImpar(Item *v, int n) {
int ordenado = 0;
while(!ordenado) {
   ordenado = 1;
   for(int i = 0; i < n-1; i += 2)
      if(v[i] > v[i+1]) {
         Troca(v[i], v[i+1]);
         ordenado = 0;
      }
   for (int i = 1; i < n-1; i += 2)
      if(v[i] > v[i+1]) {
         Troca(v[i], v[i+1]);
         ordenado = 0;
      }
}
```



Método Seleção

- Seleção do n-ésimo menor (ou maior) elemento da lista
- Troca do n-ésimo menor (ou maior) elemento com a n-ésima posição da lista
- Uma única troca por vez é realizada



Método Seleção

```
void Selecao (Item *v, int n){
int i, j, Min;

  for (i = 0; i < n - 1; i++) {
    Min = i;
    for (j = i + 1 ; j < n; j++)
      if (v[j].chave < v[Min].chave)
        Min = j;
    Troca(v[i], v[Min]);
  }
}
```

(E)	X	E	M	P	L	O
E	X	(E)	M	P	L	O
E	E	X	M	P	(L)	O
E	E	L	(M)	P	X	O
E	E	L	M	P	X	(O)
E	E	L	M	O	X	(P)
E	E	L	M	O	P	(X)
E	E	L	M	O	P	X

Método Seleção: Complexidade

- Comparações – C(n):

$$C(n) = \sum_{i=0}^{n-2}(n-1)+(i+1)+1 = \sum_{i=0}^{n-2}n - \sum_{i=0}^{n-2}i - \sum_{i=0}^{n-2}1$$
$$= n(n-1) - \frac{(n-1)(n-2)}{2} - (n-1)$$
$$= \frac{n^2-n}{2} = O(n^2)$$

- Movimentações – M(n):

$$M(n) = (n-1) = O(n)$$

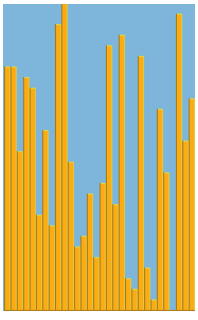
Método Seleção

- Vantagens:
  - Custo linear no tamanho da entrada para o número de movimentos de registros – a ser utilizado quando há registros muito grandes;
- Desvantagens:
  - Não adaptável (não importa se o arquivo está parcialmente ordenado);
  - Algoritmo não é estável;

Método Inserção

- Algoritmo utilizado pelo jogador de cartas
  - As cartas são ordenadas da esquerda para direita uma a uma.
  - O jogador escolhe a segunda carta e verifica se ela deve ficar antes ou na posição que está.
  - Depois a terceira carta é classificada, deslocando-a até sua correta posição.
  - O jogador realiza esse procedimento até ordenar todas as cartas.

Método Inserção



Método Inserção

```
void Insercao(Item *v, int n) {
int i,j;
Item aux;
  for (i = 1; i < n; i++) {
    aux = v[i];
    j = i - 1;
    while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {
      v[j + 1] = v[j];
      j--;
    }
    v[j + 1] = aux;
  }
}
```

E	(X)	E	M	P	L	O
E	X	(E)	M	P	L	O
E	E	X	(M)	P	L	O
E	E	M	X	(P)	L	O
E	E	M	P	X	(L)	O
E	E	L	M	P	X	(O)
E	E	L	M	O	P	X

Método Inserção: Complexidade

- Comparações – C(n):
  - Anel interno: i-ésima iteração, valor de C_i:
    - melhor caso: $C_i = 1$
    - pior caso: Ci = i
    - caso médio: $C_i = (1 + 2 + 3 + \ldots + i) / i$

$$C_i = \frac{1}{i}\sum_{k=1}^{i} k = \frac{1}{i}\left(\frac{i(i+1)}{2}\right) = \frac{i+1}{2}$$

  - Para o caso médio, assume-se que todas as permutações de entrada são igualmente prováveis.

Método Inserção: Complexidade

- Comparações – C(n):
  - Anel externo:

$$\sum_{i=1}^{n-1} C_i$$

  - Complexidade total:
    - Melhor caso (itens já estão ordenados)

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 = O(n)$$

    - Pior caso (itens em ordem reversa):

$$C(n) = \sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

Método Inserção: Complexidade

- Comparações – C(n):
  - Caso médio:

$$C(n) = \sum_{i=1}^{n-1} \frac{i+1}{2} = \frac{1}{2}\left(\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1\right) = \frac{1}{2}\left(\frac{n(n-1)}{2} + (n-1)\right)$$

$$= \left(\left(\frac{n^2-n}{4}\right) + \left(\frac{n-1}{2}\right)\right) = \frac{n^2}{4} + \frac{n}{4} - \frac{1}{2} = O(n^2)$$

Método Inserção: Exemplos

- Melhor Caso:

1	(2)	3	4	5	6
1	2	(3)	4	5	6
1	2	3	(4)	5	6
1	2	3	4	(5)	6
1	2	3	4	5	(6)
1	2	3	4	5	6

- Pior Caso:

6	(5)	4	3	2	1
5	6	(4)	3	2	1
4	5	6	(3)	2	1
3	4	5	6	(2)	1
2	3	4	5	6	(1)
1	2	3	4	5	6

Método Inserção

- Vantagens:
  - É o método a ser utilizado quando o arquivo está "quase" ordenado.
  - É um bom método quando se deseja adicionar uns poucos itens a um arquivo ordenado, pois o custo é linear.
  - O algoritmo de ordenação por inserção é **estável.**
- Desvantagens:
  - Alto custo de movimentação de elementos no vetor.

Ordenação Interna: Sumário

- Métodos Simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.
- Métodos Eficientes:
  - Adequados para arquivos maiores.
  - Requerem O(n log n) comparações.
  - As comparações são mais complexas nos detalhes.
  - Métodos simples são mais eficientes para pequenos arquivos.