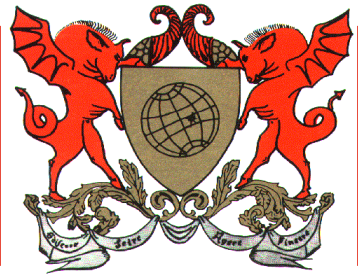

Universidade Federal de Viçosa
Campus de Florestal

Algoritmos e Estruturas de Dados I (CCF 211)

Ordenação Simples
(Cap 03 - Ziviani)

Profa.Thais R. M. Braga Silva
<thais.braga@ufv.br>

Ordenação

- Ordenar corresponde ao processo de rearranjar um conjunto de objetos em ordem ascendente ou descendente

- Objetivo principal: facilitar recuperação posterior de itens
  - Catálogo de telefone
  - Dicionário
  - Lista de Produtos

Ordenação

- Algoritmos de ordenação ilustram regras básicas de manipulação de estruturas de dados

- Ordenar é uma tarefa presente na maioria dos sistemas

Critério de Ordenação

- Ordena-se de acordo com uma chave:
- Além da chave, podem existir outros componentes no registro
  - Não influenciam na ordenação
- O tipo da chave depende da aplicação
  - Mais comuns são numérica e alfabética

```
typedef int ChaveTipo;

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

Características

- Estabilidade: relativo à manutenção da ordem original de itens de chaves iguais
  - Um método de ordenação é **estável** se a ordem relativa dos itens com chaves iguais não se altera durante a ordenação.
- Ordenação interna: arquivo a ser ordenado cabe todo na memória principal.
- Princípio: comparação x distribuição

Critério de Avaliação

- Sendo n o número de registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações C(n) entre chaves.
  - Número de movimentações M(n) de itens

Outras Considerações

- O uso econômico da memória disponível é um requisito primordial na ordenação interna.
- Métodos de ordenação ***in situ*** são os preferidos.
- Métodos que utilizam listas encadeadas não são muito utilizados.
- Métodos que fazem cópias dos itens a serem ordenados possuem menor importância.

Métodos Simples

- Bolha (*BubbleSort*)
- Seleção (*SelectSort*)
- Inserção (*InsertSort*)

Método Bolha

- Os elementos vão “borbulhando” a cada iteração do método até a posição correta para ordenação da lista

- Como os elementos são trocados (borbulhados) frequentemente, há um alto custo de troca de elementos

Método Bolha

```
void Bolha (Item* v, int n )
{
  int i, j;
  Item aux;
  for( i = 0 ; i < n-1 ; i++ )
  {
    for( j = 1 ; j < n-i ; j++ )
      if ( v[j].Chave < v[j-1].Chave )
      {
        aux = v[j];
        v[j] = v[j-1];
        v[j-1] = aux;
      } // if
   }
}
```

Análise de Complexidade

- Comparações – C(n)

- Movimentações – M(n)

Análise de Complexidade

- Comparações – C(n)

$$\sum_{i=0}^{n-2} \sum_{j=1}^{n-i-1} 1 = (n-1) + (n-2) + (n-3) + \ldots + 1 = \frac{(n^2 - n)}{2}$$

- Movimentações – M(n)

$$M(n) = 3C(n)$$

Ordenação por Bolha

- Vantagens:
  - Algoritmo simples
  - Algoritmo estável
- Desvantagens:
  - O arquivo já estar ordenado não ajuda a reduzir o número de comparações (custo continua quadrático), mas o número de movimentação cai a zero.
- Possível modificação na atual implementação?
  - O método poderia parar quando nenhum elemento borbulhasse/trocasse de posição

Método Bolha

```
void BolhaMelhorado (Item* v, int n ){
  int i, j, troca;
  Item aux;

  for( i = 0 ; i < n-1 ; i++ )
  {
    troca = 0;
    for( j = 1 ; j < n-i ; j++ )
      if ( v[j].Chave < v[j-1].Chave )
      {
        aux = v[j];
        v[j] = v[j-1];
        v[j-1] = aux;
        troca = 1;
      } // if
    }
    if (troca == 0)
      break;
  }
```

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

	1	2	3	4	5	6
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*
i = 1	***A***	*R*	*D*	*E*	*N*	***O***
i = 2	*A*	***D***	***R***	*E*	*N*	*O*
i = 3	*A*	*D*	***E***	***R***	*N*	*O*
i = 4	*A*	*D*	*E*	***N***	***R***	*O*
i = 5	*A*	*D*	*E*	*N*	***O***	***R***

Método Seleção

```
void Selecao (Item* v, int n){
  int i, j, Min;
  Item aux;
  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;
    aux = v[Min];
    v[Min] = v[i];
    v[i] = aux;
  }
}
```

Análise de Complexidade

- Comparações – C(n)

- Movimentações – M(n)

Análise de Complexidade

- Comparações – C(n)

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1=(n-1)+(n-2)+(n-3)+....+1=\frac{(n^2-n)}{2}$$

- Movimentações – M(n)

$$M(n)=3(n-1)$$

Ordenação por Seleção

- Vantagens:
  - Custo linear no tamanho da entrada para o número de movimentos de registros.
  - É o algoritmo a ser utilizado para arquivos com **registros** muito grandes.
  - É muito interessante para arquivos pequenos.
- Desvantagens:
  - O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo continua quadrático.
  - O algoritmo não é **estável.**

Método Inserção

- Algoritmo utilizado pelo jogador de cartas
  - As cartas são ordenadas da esquerda para direita uma por 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
- Alto custo em remover uma carta de uma posição e colocá-la em outra quando a representação é por arranjos

Método Inserção

	1	2	3	4	5	6
Chaves iniciais:	**O**	*R*	*D*	*E*	*N*	*A*
i = 2	**O**	**R**	*D*	*E*	*N*	*A*
i = 3	**D**	**O**	**R**	*E*	*N*	*A*
i = 4	**D**	**E**	**O**	**R**	*N*	*A*
i = 5	**D**	**E**	**N**	**O**	**R**	*A*
i = 6	**A**	**D**	**E**	**N**	**O**	**R**

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;
  }
}
```

Análise de Complexidade

- Comparações – C(n)

- Movimentações – M(n)

Análise de Complexidade

- Comparações – C(n)
  - No anel mais interno, na i-ésima iteração, o valor de C_i é:
    - melhor caso : $C_i(n) = 1$
    - pior caso : $C_i(n) = i$
    - caso médio : $C_i(n) = 1/i\ (1 + 2 + ... + i) = (i+1)/2$
  - Assumindo que todas as permutações de n são igualmente prováveis no caso médio. Assim temos:
    - melhor caso: $C(n) = (1 + 1 + ... + 1) = n - 1$
    - pior caso : $C(n) = (1 + 2 + ... + n-1) = n^2/2 - n/2$
    - caso médio : $C(n) = ½\ (2 + 3 + ... + n\ ) = n^2/4 + n/4 – 1/2$

Análise de Complexidade

- Movimentações – M(n)
  - No anel mais interno, na i-ésima iteração, o valor de M_i é:
    - melhor caso : $M_i(n) = 0$
    - pior caso : $M_i(n) = i$
    - caso médio : $M_i(n) = 1/i\ (0 + 1 + 2 + ... + i-1) = (i-1)/2$
  - Assumindo que todas as permutações de n são igualmente prováveis no caso médio. Assim temos:
    - melhor caso: $M(n) = (2 + 2 + ... + 2) = 2n - 2$
    - pior caso : $M(n) = (2+1 + 2+2 + ... + 2+n-1) = (n^2+3n-4)/2$
    - caso médio : $M(n) = ½\ (2 + 3 + ... + n\ ) = (n^2 + n – 2)/2$

Ordenação por Inserção

- O número mínimo de comparações e movimentos ocorre quando os itens estão originalmente em ordem.
- O número máximo ocorre quando os itens estão originalmente na ordem reversa.
- 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.**

Ordenação Interna

- Classificação dos métodos de ordenação interna:
  - 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.
    - Usam menos comparações.
    - As comparações são mais complexas nos detalhes.
    - Métodos simples são mais eficientes para pequenos arquivos.