Algoritmos de ordenação (classificação)

Prof. Tiago Massoni
Prof. Fernando Buarque

Engenharia da Computação

Poli - UPE

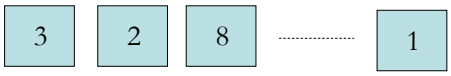
Motivação

Lema: dados quase sempre se apresentam desorganizados

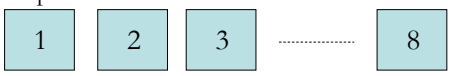
Problema: e se se quiser um processamento (busca) rápido?

Intuição

Antes



Depois



x[i] <= x[j] para **0<=i<j<n**

Definição

"**Algoritmos de ordenação** implementam operações computacionais que visam impor certa ordem ou agrupamento à conjuntos de dados"

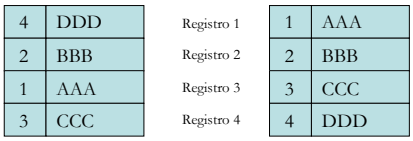
Por que ordenar?

- Necessidade inerente de algumas aplicações
  - Exemplo: lista dos cheques ordenada por número do cheque no extrato
- Normalmente é uma subrotina chave para um algoritmo
  - Exemplo: programa que desenho elementos gráficos em termos de uma ordem (over, under)
- Grande variedade de técnicas; evolução da computação
- Exemplos de uso
  - Lista telefônica
  - Lista de chamada de alunos
  - Cadastro nacional de pessoas físicas (CPF)
  - Cadastro de placas de veículos automotores

Terminologia

- Os algoritmos trabalham sobre os registros de um arquivo
- Cada registro possui uma **chave** utilizada para controlar a ordenação
- Podem existir outros componentes em um registro

Ordenação de registros



Antes da ordenação

Depois da ordenação

Ordenação de registros

Tabela de ponteiros
Original

Tabela de ponteiros
Ordenada/Classificada

Classificação dos métodos

- Ordenação interna: arquivo a ser ordenado cabe todo na memória principal
- Ordenação externa: arquivo a ser ordenado não cabe na memória principal
- Diferenças entre os métodos
  - Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado
  - Em um método de ordenação externa, os registros são acessados seqüencialmente ou em grandes blocos

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 complexa nos detalhes.
- Métodos simples são mais eficientes para pequenos arquivos

Ordenação interna

- A classe mostrada a seguir apresenta os métodos de ordenação interna que serão estudados
- Utilizaremos um array v de registros do tipo Comparable e uma variável inteira n com o tamanho de v

```
public class Ordenacao {
  public static void selection(Comparable v[],int n){..}
  public static void insertion(Comparable v[],int n){..}
  public static void shellsort(Comparable v[],int n){..}
  public static void heapsort(Comparable v[],int n){..}
  public static void quicksort(Comparable v[],int n){..}
  public static void heapsort(Comparable v[],int n){..}
}
```

Seleção (selection)

- Um dos algoritmos mais simples de ordenação
- Algoritmo:
  - Selecione o menor item do vetor
  - Troque-o com o item da primeira posição do vetor
- Repita essas duas operações com os n – 1 itens restantes, depois com os n – 2 itens, até que reste apenas um elemento

Seleção (selection)

- Ilustração
  - As chaves em negrito sofreram uma troca entre si
- Cada passo procura o menor à direita e troca com o atual

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**

Seleção (selection)

```
public static void selecao(Comparable v [],int n) {
 for ( int i = 0; i < n - 1; i ++) {
  int min=i ;
  for ( int j = i + 1; j<n; j++)
    if (v[j].compareTo(v[min])<0) min=j ;
    Comparable tmp = v[min];
    v[min]=v[i];
    v[i]=tmp;
  }
}
```

- Vantagens
  - Custo linear no tamanho da entrada para o número de movimentos de registros
  - Bom para arquivos com registros muito grandes (se for movê-lo)
  - É muito interessante para arquivos pequenos
- Desvantagens
  - Arquivo já ordenado não ajuda em nada, pois o custo continua quadrático

Inserção (insertion)

- Algoritmo
  - Em cada passo a partir de i=2 faça:
    - Selecione o i-ésimo item da seqüência fonte
    - Coloque-o no lugar apropriado na seqüência destino de acordo com o critério de ordenação.
- Método preferido dos jogadores de cartas

Inserção (insertion)

- Ilustração
  - As chaves em negrito representam a seqüência destino
  - move-se itens com chaves maiores para a direita e então inserindo o item na posição deixada vazia

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**

Inserção (insertion)

```
public static void insercao (Comparable v[], int n){
  int j;
  for (int i=1; i<n; i++){
    Comparable x= v[i];
    for (j=i;j>0 && x.compareTo(v[j-1])<0;j--) {
      v[j]= v[j-1];
    }
    v[j]= x;
  }
}
```

- O número mínimo de comparações e movimentos ocorre quando os itens estão originalmente em ordem - O(N)
- O número máximo ocorre quando os itens estão originalmente na ordem reversa – $O(N^2)$
- É o método a ser utilizado quando o arquivo está "quase" ordenado

Shellsort

- Proposto por Donald Shell em 1959
- É uma extensão do algoritmo de ordenação por inserção
- Problema com o algoritmo de ordenação por inserção
  - Troca itens adjacentes para determinar o ponto de inserção
  - São efetuadas n – 1 comparações e movimentações quando o menor item está na posição mais à direita no vetor
- O método de Shell contorna este problema permitindo trocas de registros distantes um do outro

Shellsort

- Os itens separados de h posições são rearranjados
- Todo h-ésimo item leva a uma seqüência ordenada
- Tal seqüência é dita estar h-ordenada
- Melhor sequência de h: h(s) = 1, 4, 13, 40, 121, 364, ...
  - 3 * h(anterior) + 1
  - h(1) = 1

Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*
h = 4	*N*	*A*	*D*	*E*	*O*	*R*
h = 2	*D*	*A*	*N*	*E*	*O*	*R*
h = 1	*A*	*D*	*E*	*N*	*O*	*R*

Shellsort

```
public static void shellsort(Comparable v[],int n){
  int h = n/2; //não eh a melhor distribuicao

  do {
    for (int i=h; i<n; i++) {
      Comparable x = v[i];
      int j = i;

      while (x.compareTo(v[j-h])>0 && j>h) {
        v[j] = v[j-h];
        j-=h;
      }
      v[j]= x;
    }

    h /= 2;
  }while (h>0);
}
```


Shellsort

- A razão da eficiência do algoritmo ainda não é conhecida
  - Ninguém ainda foi capaz de analisar o algoritmo
- A sua análise contém alguns problemas matemáticos muito difíceis
- A começar pela própria seqüência de
- incrementos
  - O que se sabe é que cada incremento não deve ser múltiplo do anterior