

Estrutura de Dados II

Ordenação em Memória Primária - Parte I

Prof. Dr. Marcelo Otone Aguiar

Universidade Federal do Espírito Santo - UFES

11 de Maio de 2023

Conteúdo

- Conceitos Gerais
- Ordenação por troca direta (Bolha)
- Ordenação por inserção direta
- Ordenação por inserção binária
- Ordenação por inserção com diminuição de incremento (Shellsort)
- Ordenação por seleção direta

Conceitos Gerais

- Ordenar corresponde ao processo de reorganizar um conjunto de objetos em ordem ascendente ou descendente
- O objetivo principal é facilitar a recuperação posterior de itens do conjunto ordenado
- Em geral os algoritmos trabalham sobre os registros de um arquivo. Apenas uma parte do registro, chamada chave, é utilizada para controlar a ordenação
- Além da chave, podem existir outros componentes em um registro, os quais não têm influência no processo de ordenar

Conceitos Gerais

- A escolha do tipo para a chave é arbitrária
- As ordens numéricas ou alfabéticas são usuais
- Um método de ordenação é dito **estável** se a ordem relativa dos itens com chaves iguais mantém-se inalterada pelo processo de ordenação
- **Exemplo:** se uma lista alfabética de nomes de funcionários de uma empresa é ordenada pelo campo salário, então um método estável produz uma lista em que os funcionários com mesmo salário aparecem em ordem alfabética

Conceitos Gerais

- Os métodos de ordenação são classificados em dois grandes grupos:
 - **Ordenação interna:** quando o arquivo a ser ordenado cabe todo na memória principal
 - **Ordenação externa:** quando o arquivo não cabe na memória principal e precisa ser armazenado na memória secundária
- A principal diferença é que na **ordenação interna** qualquer registro pode ser **imediatamente acessado**, na **ordenação externa** os registros são **acessados sequencialmente** ou em grandes blocos

Conceitos Gerais

- A grande maioria dos métodos de ordenação é baseada em **comparações das chaves**
- Porém, existem métodos de ordenação que utilizam o princípio da **distribuição**
- Os métodos de **ordenação por distribuição** são também conhecidos, como **ordenação por contagem**, **ordenação digital**. Exemplos de métodos são: **radixsort** ou **bucketsort**
- Uma das dificuldades de implementar esses métodos está relacionada com o problema de lidar com a memória necessária para armazenar os itens distribuídos

Conceitos Gerais

- O aspecto predominante na escolha do algoritmo de ordenação é o tempo gasto para ordenar um arquivo
- A estratégia mais adotada é a comparação de chaves, assim, as medidas de complexidade contam o número de comparações entre chaves e o número de trocas de itens do arquivo
- A quantidade extra de memória auxiliar utilizada pelo algoritmo é também um aspecto importante
- O uso econômico da memória disponível é um requisito primordial na ordenação

Conceitos Gerais

- Os métodos que utilizam o princípio da comparação de chaves podem ser divididos em **métodos simples** e **métodos eficientes**
- Os métodos simples são adequados para pequenos arquivos e requerem $O(n^2)$ comparações
- Os métodos eficientes são adequados para arquivos maiores e requerem $O(n \log n)$ comparações
- Os métodos simples produzem programas pequenos, fáceis de entender, que ilustram com simplicidade os princípios de ordenação por comparação

Ordenação por Troca Direta (Bolha)

- O método de **ordenação Bolha** é bastante simples, e talvez seja o método de ordenação mais difundido
- Uma iteração do mesmo se limita a percorrer a tabela do início ao fim, sem interrupção, trocando de posição dois elementos consecutivos sempre que estes se apresentem fora de ordem
- Intuitivamente percebe-se que a intenção do método é mover os elementos maiores em direção ao fim da tabela
- Ao terminar a primeira iteração pode-se garantir que as trocas realizadas posicionam o maior elemento na última posição
- Na segunda iteração, o segundo maior elemento é posicionado, e assim sucessivamente
- O processamento é repetido então $n-1$ vezes

Ordenação por Troca Direta (Bolha)

índices	0	1	2	3	4
chaves	4	1	3	7	2
$i=4$ $j=0$	1	4	3	7	2
$j=1$	1	3	4	7	2
$j=2$	1	3	4	7	2
$j=3$	1	3	4	2	7
$i=3$ $j=0$	1	3	4	2	
$j=1$	1	3	4	2	
$j=2$	1	3	2	4	
$i=2$ $j=0$	1	3	2		
$j=1$	1	2	3		
$i=1$ $j=0$	1	2			

Ordenação por Troca Direta (Bolha)

- Vantagens:
 - Algoritmo simples
 - O método é estável
- Desvantagens:
 - Muitas trocas de itens
 - Complexidade:
 - Melhor caso: $O(n^2)$
 - Pior caso: $O(n^2)$

Pseudo-código

```
1 In: conjunto contendo dados a serem ordenados
2 Out: conjunto ordenado
3 begin
4   for i ← n-1 to 1 do
5     for j ← 0 to j < i do
6       if V[j] > V[j+1] then
7         aux ← V[j]
8         V[j] ← V[j+1]
9         V[j+1] ← aux
10      end if
11    end for
12  end for
13 end
```

Bolha com critério de parada

- O algoritmo apresentado é claramente ruim, pois sua complexidade de pior caso é igual à de melhor caso, $O(n^2)$, devido aos percursos estipulados para as variáveis i e j
- Pode-se, entretanto, pensar em alguns critérios de parada que levariam em consideração comparações desnecessárias, isto é, comparações executadas em partes da tabela sabidamente já ordenadas
- Conforme pode ser visto no próximo algoritmo

Pseudo-código

```
1 In: conjunto contendo dados a serem ordenados
2 Out: conjunto ordenado
3 begin
4   fim ← n-1
5   trocou ← verdade
6   while trocou = verdade do
7     trocou ← falso
8     for j ← 0 to j < fim do
9       if V[j] > V[j+1] then
10        aux ← V[j]
11        V[j] ← V[j+1]
12        V[j+1] ← aux
13        trocou = verdade
14      end if
15    end for
16    fim ← fim - 1
17  end while
18 end
```

Bolha com critério de parada

- O critério de parada resulta do fato de que, se a tabela está ordenada, o algoritmo é executado em toda sua extensão desnecessariamente
- Uma variável lógica *trocou* é então introduzida com a finalidade de sinalizar se pelo menos uma troca foi realizada
- Caso isso não ocorra, o algoritmo pode ser encerrado. Essa simples alteração afeta a complexidade do melhor caso do algoritmo, que passa a ser $O(n)$, uma vez que, se a tabela já está ordenada, apenas um percurso é realizado

Bolha par-ímpar

- Variação do algoritmo Bolha, útil para processamento em paralelo
- O algoritmo é dividido em duas fases: par e ímpar
- Na **fase par** a classificação ocorre pelo método Bolha aplicada aos os elementos indexados pares
- Na **fase ímpar** a classificação é aplicada aos elementos ímpares

Pseudo-código

```
1 In: conjunto contendo dados a serem ordenados
2 Out: conjunto ordenado
3 begin
4   trocou ← verdade
5   while trocou = verdade do
6     trocou ← falso
7     for j ← 0 to j ≤ n-2 passo 2 do
8       if V[j] > V[j+1] then
9         aux ← V[j]
10        V[j] ← V[j+1]
11        V[j+1] ← aux
12        trocou = verdade
13      end if
14    end for
15    for j ← 1 to j ≤ n-2 passo 2 do
16      if V[j] > V[j+1] then
17        aux ← V[j]
18        V[j] ← V[j+1]
19        V[j+1] ← aux
20        trocou = verdade
21      end if
22    end for
23  end while
24 end
```

Ordenação por Inserção Direta

- Princípio do funcionamento:
 - Em cada passo, a partir de $i = 2$, o $i - \text{esimo}$ item da sequência fonte é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado
 - A troca é feita sempre que apropriado
 - Basicamente, ele percorre um vetor de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados

Ordenação por Inserção Direta

- Para arquivos já ordenados, o algoritmo descobre a um custo $O(n)$ que cada item já está no seu lugar
- Logo, o método da inserção é indicado quando o arquivo está **quase** ordenado
- É também um método indicado quando se deseja adicionar poucos itens a um arquivo já ordenado e depois obter outro arquivo ordenado: neste caso o custo é linear

Ordenação por Inserção Direta

índices	0	1	2	3	4
chaves	4	1	3	7	2
$i=1$ $j=0$ $aux=1$	4	4	3	7	2
	1	4	3	7	2
$i=2$ $j=1$ $aux=3$	1	4	4	7	2
$j=0$ $aux=3$	1	4	4	7	2
	1	3	4	7	2
$i=3$ $j=2$ $aux=7$	1	3	4	7	2
$j=1$ $aux=7$	1	3	4	7	2
$j=0$ $aux=7$	1	3	4	7	2
	1	3	4	7	2

Ordenação por Inserção Direta

índices	0	1	2	3	4
chaves	4	1	3	7	2
	1	3	4	7	2
$i=4$ $j=3$ $aux=2$	1	3	4	7	7
$j=2$ $aux=2$	1	3	4	4	7
$j=1$ $aux=2$	1	3	3	4	7
$j=0$ $aux=2$	1	3	3	4	7
	1	2	3	4	7

Ordenação por Inserção Direta

- Vantagens:
 - Algoritmo simples
 - O método é estável
 - Eficiente para arquivos quase ordenados (melhor caso)
- Desvantagens:
 - Muitas trocas de itens
 - Complexidade:
 - Melhor caso: $O(n)$
 - Pior caso: $O(n^2)$

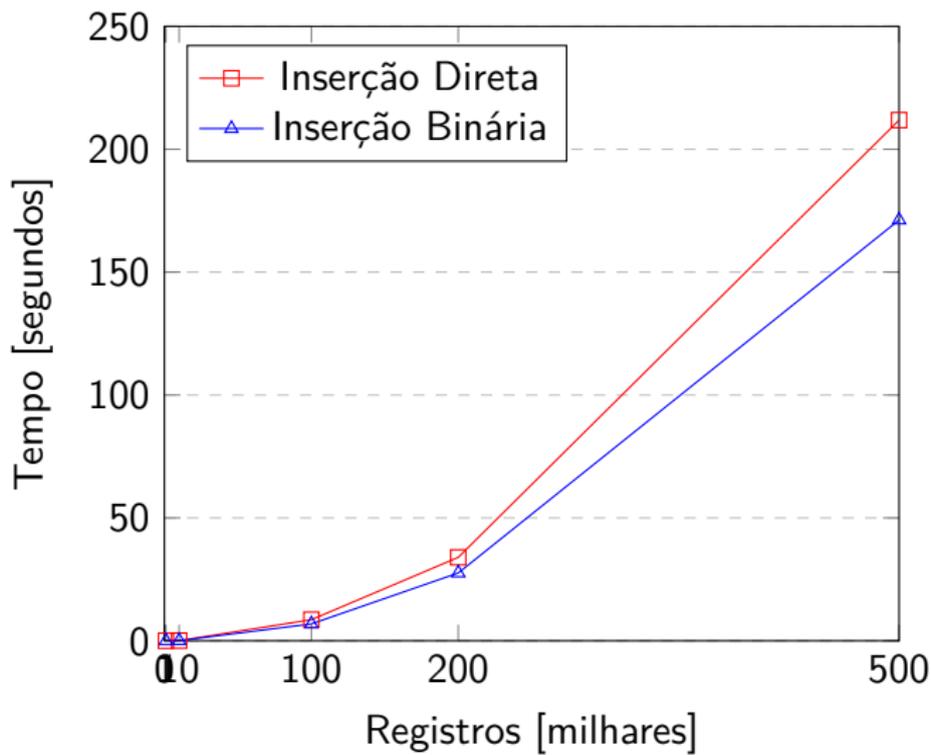
Pseudo-código

```
1 In: conjunto contendo dados a serem ordenados
2 Out: conjunto ordenado
3 begin
4   for i ← 1 to n-1 do
5     aux ← V[i]
6     j ← i-1
7     while j >= 0 and aux < V[j] do
8       V[j+1] ← V[j]
9       j ← j-1
10    end while
11    if j < (i-1) then
12      V[j+1] ← aux
13    end if
14  end for
15 end
```

Ordenação por inserção binária

- O algoritmo de inserção direta pode ser aperfeiçoado observando-se que a sequência destino $a[1], a[2], \dots, a[i - 1]$, na qual deve ser inserido o elemento x , quando a mesma já está ordenada
- Assim, pode-se utilizar um método mais rápido para determinar o ponto correto de inserção
- A escolha óbvia é a busca binária, que divide a sequência destino no seu ponto central, continuando a divisão até encontrar o ponto correto de inserção

Análise do Comportamento de Ordenação por Inserção



Ordenação por Inserção Binária

- Vantagens:
 - Reduz o número de comparações em relação à Inserção Direta
 - O método é estável
 - Eficiente para arquivos quase ordenados (melhor caso)
- Desvantagens:
 - Muitas trocas de itens
 - Complexidade:
 - Melhor caso: $O(n)$
 - Pior caso: $O(n^2)$

Pseudo-código

```
1 In: conjunto contendo dados a serem ordenados
2 Out: conjunto ordenado
3 begin
4   for i<-1 to n-1 do
5     aux <- V[i]
6     esq <- 0
7     dir <- i
8     while esq < dir do
9       meio <- (esq + dir) / 2
10      if V[meio] <= aux then
11        esq = meio + 1
12      else
13        dir = meio
14      end if
15    end while
16    for (j<-i to esq) do
17      V[j] <- V[j-1]
18    end for
19    V[dir] <- aux
20  end for
21 end
```

Ordenação por inserção com diminuição de incremento (Shellsort)

- Shell (1959) propôs uma extensão do algoritmo de ordenação por inserção
- O método da inserção troca itens adjacentes quando se está procurando o ponto de inserção na sequência destino
- Assim, se o menor item estiver na posição mais a direita no vetor, o número de comparações e movimentações é igual a $n - 1$ para encontrar o seu ponto de inserção
- O método de **Shell** contorna esse problema, permitindo trocas de registros que estão distantes um do outro

Ordenação por inserção com diminuição de incremento (Shellsort)

- Os itens que estão separados h posições são rearranjados de tal forma que todo h – *esimo* item leva a uma sequência ordenada. Tal sequência é dita h – *ordenada*
- **Shellsort** é uma ótima opção para arquivos de tamanho moderado, mesmo porque sua implementação é simples e requer uma quantidade de código pequena
- O método não é estável, pois ele nem sempre deixa registros com chaves iguais na mesma posição relativa

Shellsort

índices	0	1	2	3	4
chaves	4	7	3	1	2
$h=4$ $i=4$ $j=0$ $aux=2$	4	7	3	1	4
	2	7	3	1	4
$h=2$ $i=2$ $j=0$ $aux=3$	2	7	3	1	4
$i=3$ $j=1$ $aux=1$	2	7	3	7	4
	2	1	3	7	4
$i=4$ $j=2$ $aux=4$	2	1	3	7	4

Shellsort

índices	0	1	2	3	4
chaves	4	7	3	1	2
	2	1	3	7	4
$h=1$ $i=1$ $j=0$ $aux=1$	2	2	3	7	4
	1	2	3	7	4
$i=2$ $j=1$ $aux=3$	1	2	3	7	4
$i=3$ $j=2$ $aux=7$	1	2	3	7	4
$i=4$ $j=3$ $aux=4$	1	2	3	7	7
	1	2	3	4	7

Shellsort

- Vantagens:
 - Implementação simples
 - Contorna o problema da ordenação por Inserção, possibilitando a troca de itens distantes um do outro
 - Com a troca de itens distantes, a etapa final do algoritmo se resume à aplicação da ordenação por Inserção em um conjunto quase ordenado
 - Ótima opção para arquivos de tamanho moderado
- Desvantagens:
 - O tempo de execução é sensível à ordem inicial do conjunto
 - O método não é estável
 - Complexidade:
 - Ainda há muita dúvida sobre a complexidade desse algoritmo
 - Melhor caso: $O(n \log_2 n)$
 - Pior caso (melhor conhecida): $O(n \log_2 n)$

Pseudo-código

```
1 In: conjunto contendo dados a serem ordenados
2 Out: conjunto ordenado
3 begin
4   h ← -1
5   while h < n do
6     h ← 3 * h + 1
7   end while
8   while h > 1 do
9     h ← h / 3
10    for i ← h to n do
11      aux ← V[i]
12      j ← i - h
13      while j >= 0 and aux < V[j] do
14        V[j+h] ← V[j]
15        j ← j - h
16      end while
17      V[j+h] ← aux
18    end for
19  end while
20 end
```

Ordenação por Seleção Direta

- Princípio do funcionamento:
 - Selecione o **menor item** do vetor e a seguir **troque-o** com o **item** que está **na 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
- O algoritmo de ordenação por seleção é um dos métodos de ordenação mais simples que existem
- O número de movimentações de registro é linear no tamanho da entrada, conseqüentemente, este é indicado para arquivos com registros muito grandes

Ordenação por Seleção Direta

índices	0	1	2	3	4
chaves	4	1	3	7	2
min=0 i=0 j=1	4	1	3	7	2
min = 1 j=2					
min = 1 j=3					
min = 1 j=4					
min = 1	1	4	3	7	2
min=1 i=1 j=2	1	4	3	7	2
min = 2 j=3					
min = 2 j=4					
min = 4	1	2	3	7	4

Ordenação por Seleção Direta

índices	0	1	2	3	4
chaves	4	1	3	7	2
	1	2	3	7	4
min=2 i=2 j=3	1	2	3	7	4
min = 2 j=4					
	1	2	3	7	4
min=3 i=3 j=4	1	2	3	7	4
min=4	1	2	3	4	7

Ordenação por Seleção Direta

- Vantagens:
 - Algoritmo simples
 - Realiza poucas trocas
 - Não necessidade de vetor auxiliar (*in-place*)
 - Útil na ordenação de conjuntos pequenos
- Desvantagens:
 - Método **não** é estável
 - Muitas comparações independente do vetor estar ou não ordenado
 - Muito lento para conjuntos grandes de dados
 - Complexidade:
 - Melhor caso: $O(n^2)$
 - Pior caso: $O(n^2)$

Pseudo-código

```
1 In: conjunto contendo dados a serem ordenados
2 Out: conjunto ordenado
3 begin
4   for i ← 0 to n-1 do
5     min ← i
6     for j ← i+1 to n do
7       if V[j] < V[min] then
8         min ← j
9       end if
10    end for
11    if i < min then
12      aux ← V[i]
13      V[i] ← V[min]
14      V[min] ← aux
15    end if
16  end for
17 end
```