Introdução à Ciência da Computação II

Ordenação: Métodos da Bolha, Seleção & Inserção

Prof. Ricardo J. G. B. Campello

Sumário

- Introdução à Ordenação
  - Ordenação por Comparações e por Trocas
  - Estabilidade e Execução In-Place
- Método da Bolha (Bubble Sort)
- Ordenação por Seleção (Selection Sort)
- Ordenação por Inserção (Insertion Sort)

Ordenação

- Ordenação é um dos problemas algorítmicos mais fundamentais em Ciência da Computação
- Algumas Aplicações:
  - Teste de Unicidade e Remoção de Duplicatas
    - Verificar se existem elementos repetidos em uma única passagem
    - Remover os elementos repetidos → p. ex. conjuntos
  - Restabelecimento de Ordem
    - Recuperar uma dada ordem (a ser modificada) de um conjunto de elementos via armazenamento da posição original de cada um
  - Histogramas

3

Ordenação

- Algumas Aplicações (cont.):
  - Operações com Conjuntos via Fusão
    - Intersecção, União, Diferença, ...
  - Busca Eficiente
    - Binária
  - Seleção Simples (embora não eficiente)
    - Selecionar o k-ésimo maior/menor valor
  - Priorização de Eventos
    - Substituição de filas de prioridade em aplicações estáticas simples
  - E muito mais...

4

Ordenação por Comparação

- Dada uma seqüência de elementos:

  $a_1, a_2, a_3, \ldots, a_{n-1}, a_n$

- Obter, via comparações aos pares, uma permutação desses elementos:

  $a'_1, a'_2, a'_3, \ldots, a'_{n-1}, a'_n$

  de modo que

  $a'_1 \leq a'_2 \leq \ldots \leq a'_{n-1} \leq a'_n$

  onde ≤ é uma relação de ordem total

5

Ordenação por Trocas

- Ordena a seqüência permutando-a, ou seja, **trocando as posições relativas** de seus elementos
- Existem diversos algoritmos de ordenação por trocas
- Os **métodos da bolha, seleção e inserção** são os mais fáceis de entender e simples de implementar
- Por simplicidade, introduziremos esses métodos assumindo que a seqüência de elementos a serem ordenados resume-se a um vetor de números

6

Método da Bolha (Bubble-Sort)

- Percorra o vetor da esquerda para direita:
  - comparando elementos vizinhos, troque os que estiverem fora de ordem
- Considere o vetor inicial: | 7 | 5 | 9 | 3 | 2 |

comparando v[0] com v[1] ⇒ **troca**

comparando v[1] com v[2] ⇒ **não troca**

comparando v[2] com v[3] ⇒ **troca**

comparando v[3] com v[4] ⇒ **troca**

- Após a 1a iteração o vetor fica: | 5 | 7 | 3 | 2 | 9 |

Execução Completa

7	5	9	3	2	Disposição Inicial
5	7	3	2	9	1ª iteração
5	3	2	7	9	2ª iteração
3	2	5	7	9	3ª iteração
2	3	5	7	9	4ª iteração

8

Algoritmo - Bolha

```
void Bolha (double v[], int n) {
    int aux, j, i;
    for(j=n-1; j>=1; j--)
        for(i=0; i<j; i++)
           if (v[i] > v[i+1]) {
              aux = v[i];
              v[i] = v[i+1];
              v[i+1] = aux;
           }
} /* ordena vetor v com n elementos em ordem crescente */
```

9

Desempenho

- Na j-ésima iteração o algoritmo executa $n - j$ comparações e, no máximo, $n - j$ trocas
- O vetor estará ordenado no máximo após $n - 1$ iterações, quando o algoritmo encerra a execução
- Logo, considerando que os elementos a serem ordenados podem ser comparados e trocados em tempo constante, o algoritmo roda em tempo :

$$O(n-1 + \ldots + 2 + 1) \Rightarrow O((n-1)n/2) \Rightarrow O(n^2)$$

10

Características Desejáveis

- Existem Algumas Características Desejáveis em um Algoritmo de Ordenação:
  - **Estabilidade**: Um algoritmo é dito estável se não altera a ordem relativa original de elementos iguais (repetidos) no vetor
    - Veremos porque isso pode ser muito importante ao discutir ordenação com múltiplas chaves, posteriormente no curso
  - **In-Place**: Um algoritmo é dito *in-place* se requer apenas alguma memória auxiliar adicional que não depende do vetor a ser ordenado (nem em termos dos valores nem do tamanho)
    - Além da memória necessária para armazenar o próprio vetor

Características Desejáveis

- Algoritmos de ordenação por trocas em geral são *in-place*, mas não necessariamente são estáveis
- **Bubble-Sort**, embora seja ineficiente em termos computacionais, possui ambas as propriedades:
  - **Estável**
  - **In-Place**
  - além de ser muito simples e didático

12

Ordenação por Seleção

- Outro algoritmo de ordenação que também pode ser implementado por trocas chama-se **Selection-Sort**:
  - Na iteração i (i = 0, ..., n–2), localiza-se nos $n - i$ elementos restantes (ainda não ordenados) o elemento que deve ocupar a célula de índice i no vetor final ordenado
    - o menor dentre os elementos à direita da célula de índice i (inclusa)
  - Move-se então o elemento para aquela célula
  - Exemplo:

v = [8 4 7 2]	(i = 0)
v = [**2** 8 4 7]	(i = 1)
v = [**2 4** 8 7]	(i = 2)
v = [**2 4 7 8**]	

13

Ordenação por Seleção (Ingênua)

```
void Selection_Sort(double v[], int n) {
 int min, i, j;
 double temp;
 for(i=0; i<=n-2; i++){                  /* Itera */
       min = i;
       for(j=i+1; j<=n-1; j++)           /* Localiza */
             if (v[j]<v[min]) min = j;
       for(j=min; j>i; j--){             /* Move (trocas) */
             temp = v[j];
             v[j] = v[j-1];
             v[j-1] = temp;
       }
 }
} /* ordena vetor v com n elementos em ordem crescente */
```

Desempenho

- Na i-ésima iteração ($i = 0, \ldots, n-2$):
  - o algoritmo executa $n - i - 1$ comparações e, no máximo, um número equivalente de atribuições para localizar o menor valor
  - um número não maior que este de trocas é realizado após o menor valor ter sido encontrado
- Logo, considerando que os elementos a serem ordenados podem ser comparados e trocados em tempo constante, o algoritmo roda em tempo :

$$O(n-1 + \ldots + 2 + 1) \Rightarrow O((n-1)n/2) \Rightarrow O(n^2)$$

15

Observações

- A implementação vista anteriormente é ingênua porque, na verdade, não é preciso realizar mais que uma única troca para reposicionar o menor valor
  - Exemplo:

[7 4 8 2 5 3 9]
[**2** 4 8 7 5 3 9]
[**2 3** 8 7 5 4 9]
[**2 3 4** 7 5 8 9]
[**2 3 4 5** 7 8 9]
[**2 3 4 5 7** 8 9]
[**2 3 4 5 7 8 9**]

16

Observações

- Isso torna o algoritmo mais eficiente em termos de fatores constantes, porém não altera a sua complexidade quadrática em termos assintóticos
  - Porque... ?
- Além disso, esse ganho de eficiência em fatores constantes vem com um preço associado:
  - Enquanto a implementação ingênua **é estável**, a implementação com troca única **não é estável !**
  - Vide exercícios

17

Ordenação por Inserção

- Outro algoritmo de ordenação que também opera por meio de trocas é denominado **Insertion-Sort**:
  - Na iteração i (i = 1, ..., n–1), toma-se o elemento na i-ésima célula do vetor e move-o para a posição relativa apropriada referente aos i elementos à sua esquerda (já ordenados)
  - Exemplo:

v = [8 4 7 2]	(i = 1)
v = [4 8 7 2]	(i = 2)
v = [4 7 8 2]	(i = 3)
v = [2 4 7 8]	

18

Ordenação por Inserção

```
insertion_sort(int s[], int n)
{
    int i,j;

    for (i=1; i<n; i++) {
        j=i;
        while ((j>0) && (s[j] < s[j-1])) {
            swap(&s[j],&s[j-1]);
            j = j-1;
        }
    }
}
```

```
void swap(int *a, int *b){
      int aux;
      aux = *a;
      *a = *b;
      *b = aux;
}
```

(7,4,8,2,5,3,9)

(**4**,**7**,8,2,5,3,9)

(**4**,**7**,**8**,2,5,3,9)

(**2**,**4**,**7**,**8**,5,3,9)

(**2**,**4**,**5**,**7**,**8**,3,9)

(**2**,**3**,**4**,**5**,**7**,**8**,9)

(**2**,**3**,**4**,**5**,**7**,**8**,**9**)

19

Desempenho

- Na i-ésima iteração (i = 1, ..., n–1):
  - o algoritmo executa no mínimo 1 e no máximo i comparações
  - o algoritmo executa no mínimo 0 e no máximo i – 1 trocas
- Logo, considerando que os elementos a serem ordenados podem ser comparados e trocados em tempo constante, o algoritmo roda em tempo :

$$O(1 + 2 + \ldots + n-1) \Rightarrow O(n^2) \quad \text{[Pior Caso]}$$

$$O(\underbrace{1 + \ldots + 1}_{n-1 \text{ vezes}} + 1) \Rightarrow O(n) \quad \text{[Melhor Caso]}$$

- Ou seja, a complexidade do algoritmo é $\Omega(n)$ e $O(n^2)$

20

Para Pensar...

- Quais são os tipos de seqüências a serem ordenadas que levam ao pior e ao melhor caso do tempo de execução de Insertion-Sort ?

21

Resumo

- **Bubble-Sort**, **Selection-Sort** e **Insertion-Sort** são todos algoritmos simples (de fácil compreensão e implementação)
- Todas as implementações vistas são **In-Place** e **Estáveis**
  - Exceto a versão melhorada de Selection-Sort (com troca única a cada iteração), que **não mais garante estabilidade**
- Em termos assintóticos, **Insertion-Sort** é tão ineficiente quanto os outros no pior caso, assim como na média...
  - Porém, é muito mais eficiente no melhor caso
- Apesar da ineficiência em termos assintóticos, esses algoritmos podem ser mais rápidos que algoritmos mais sofisticados para problemas com entradas de tamanho pequeno (n pequeno)

22

Exercícios

1. Modifique o código do algoritmo "Bolha" para que este interrompa antecipadamente se o vetor ficar ordenado antes de n – 1 iterações
   - Essa modificação altera a complexidade do algoritmo no melhor ou no pior caso ? Discuta
2. Mostre como fica o vetor v abaixo após cada passagem do algoritmo Buble-Sort :

v =

4	5	10	2	10	8	1	7

Mostre ainda que a **estabilidade** é mantida

Exercícios

3. Modifique a implementação ingênua de Selection-Sort para que esta se torne mais eficiente por realizar uma única troca a cada iteração (conforme exemplo nos slides)
4. Mostre através de um exemplo que a implementação de Selection-Sort com troca única não garante estabilidade
5. Modifique o algoritmo Selection-Sort para que ele ordene o vetor em ordem decrescente, ao invés de crescente
6. Repita o Exercício 5 para Insertion-Sort. As seqüências que levavam ao pior e melhor casos em termos de tempo de execução permanecem as mesmas? Discuta

Bibliografia

- N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Edição, 2004
- M. T. Goodrich & R. Tamassia, *Data Structures and Algorithms in C++/Java*, John Wiley & Sons, 2002/2005

25