06 – Ordenação em Memória Interna (parte 1)

SCC201/501 - Introdução à Ciência de Computação II

Prof. Moacir Ponti Jr.

www.icmc.usp.br/~moacir

Instituto de Ciências Matemáticas e de Computação – USP

2010/2

Sumário

Ordenação, Classificação e Organização

- Um algoritmo de ordenação é um algoritmo que recebe uma lista de elementos e gera como saída uma nova lista de elementos ordenados (classificados, organizados) em uma determinada ordem.
- O processo de organizar os dados em uma determinada ordem é parte importante de muitos métodos e técnicas em computação.
  - Uma série de algoritmos de busca, intercalação/fusão, utilizam ordenação como parte do processo
  - Aplicações em geometria computacional, bancos de dados, entre outras necessitam de listas ordenadas para funcionar.
- A organização dos dados também é utilizada para apresentação à humanos por proporcionar leitura facilitada.

Os dados ordenados

- A saída de um algoritmo de ordenação deve satisfazer duas condições:
  1. estar em uma ordem crescente ou decrescente,
  2. ser uma permutação da entrada.

```
Entrada: 6 5 7 1 4 3 2

Saídas possíveis:  1 2 3 4 5 6 7
                   7 6 5 4 3 2 1


Entrada: V U X Z Y

Saídas possíveis:  U V X Y Z
                   Z Y X V U
```

Motivação

- O estudo de algoritmos de ordenação é importante pois, entre outros:
  1. permite compreender e praticar uma série de conceitos importantes:
     - as notações assintóticas,
     - análises de pior, melhor e caso esperado,
     - paradigmas de projetos de algoritmos,
     - compromisso entre uso de espaço e consumo tempo, etc.
  2. novos algoritmos e extensões de algoritmos já existentes continuam sendo propostas.
  3. ordenação é parte de muitos métodos em computação — é preciso conhecer os principais algoritmos e saber escolher qual utilizar.
  4. é possível que um dia alguém lhe pergunte a "maneira mais eficiente de ordenar 1 milhão de inteiros de 32 bits" (`http://youtu.be/HAY4TKIvSZE`).

Sumário

Terminologia

- um **arquivo** de tamanho n é uma sequência de n itens `r[0]`, `r[1]`, `...`, `r[n-1]`
- cada item no arquivo é chamado de **registro**
- uma **chave** `k[i]` é associada a cada registro `r[i]`, e geralmente é um dos dados (um campo) do registro
- **ordenação pela chave** é quando os registros são classificados por um campo chave

Terminologia

interna ou externa

- **ordenação interna**: os dados estão na memória principal e todo o processo é feito usando a memória principal.
- **ordenação externa**: os dados estão em uma memória secundária/auxiliar.

estabilidade

um algoritmo de ordenação é **estável** se para todo registro i, j, sendo que `k[i] = k[j]`,

- quando `k[i]` precede `k[j]` no arquivo de entrada, `k[i]` precede `k[j]` no arquivo ordenado.

ou seja, o algoritmo preserva a ordem relativa original dos registros de valor de chave igual

Terminologia

ordenação sobre registros

- quando a ordenação é feita movendo/copiando os registros

ordenação sobre endereços

- ordenação é feita sobre uma tabela auxiliar de ponteiros

Sumário

Eficiência

aspectos de eficiência

- complexidade de tempo do algoritmo
- espaço de memória necessário
- adequação da simplicidade (e tamanho) do problema com o método utilizado

medida de eficiência

- a **contagem de operações** para análise da eficiência é feita pelo número de operações críticas (ao problema), geralmente:
  1. comparações entre chaves
  2. movimentação de registros ou ponteiros
  3. trocas entre registros
- o **tamanho da entrada** representa o número de registros no arquivo a ser ordenado.

Sumário

Insertion sort (ordenação por inserção)

Estratégia

Percorre uma conjunto de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados.

```
passo 1      | 6 9 7 5 1 2 3
passo 2      6 | 9 7 5 1 2 3
passo 3      6 9 | 7 5 1 2 3
passo 4      6 7 9 | 5 1 2 3
passo 5      5 6 7 9 | 1 2 3
passo 6      1 5 6 7 9 | 2 3
passo 7      1 2 5 6 7 9 | 3
passo 8      1 2 3 5 6 7 9 |
```

Insertion sort (ordenação por inserção)

Estratégia

Percorre uma conjunto de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados.

```
Insertion_Sort (A, tamanho)
    para j de 2 ate tamanho faca
        chave = A[j] // guarda a chave a ser testada
        i = j - 1  // indice anterior (lista ordenada)

        enquanto (i > 0) e (A[i] > chave) faca
           A[i+1] = A[i]  // move as chaves p/ prox. posicao

           i = i - 1

        A[i+1] = chave
```

Insertion sort: análise

Comparações

- No anel mais interno, na i-ésima iteração, o número de comparações realizadas por iteração ($C_i(n)$) é:
  - melhor caso: $C_i(n) = 1$
  - pior caso: $C_i(n) = n$
  - caso esperado: $C_i(n) = \frac{1}{i}(1 + 2 + \cdots + i) \approx \frac{i+1}{2}$,

  assumindo que todas as permutações de n são igualmente prováveis para o caso médio($C(n)$).
- O número total de comparações é
  - melhor caso: $C(n) = (1 + 1 + \cdots + 1) = n - 1$
  - pior caso: $C(n) = (2 + 3 + \cdots + n) = \frac{n^2}{2} + \frac{n}{2} - 1$
  - caso esperado: $C(n) = \frac{1}{2}(3 + 4 + \cdots + n + 1) = \frac{n^2}{4} + \frac{3n}{2} - 1$

Insertion sort: análise

Movimentações

- Na i-ésima iteração, o número de movimentações é: $M_i(n) = C_i(n) - 1 + 3 = C_i(n) + 2$
- O número total é:
  - melhor caso: $M(n) = (3 + 3 + \cdots + 3) = 3(n-1)$
  - pior caso: $M(n) = (4 + 5 + \cdots + n + 2) = \frac{n^2}{2} + \frac{5n}{2} - 3$
  - caso esperado: $M(n) = \frac{1}{2}(5 + 6 + \cdots + n + 3) = \frac{n^2}{4} + \frac{11n}{4} - 3$

Análise assintótica

- O algoritmo de ordenação por inserção é $O(n^2)$

Insertion sort: análise

...comentáros...

- O número **mínimo** de comparações e movimentos ocorre quando os itens estão originalmente **ordenados**.
- O número **máximo** ocorre quando os itens estão originalmente na **ordem reversa**.
- Em arquivos ordenados, o algoritmo descobre a um custo $O(n)$ que cada item já está em seu lugar.
- Logo, o algoritmo de ordenação por inserção é interessante quando se tem arquivos "quase" ordenados
- Isso é util, por exemplo, quando precisamos inserir poucos itens a um arquivo já ordenado.
- Esse algoritmo é, também, **estável** (deixa os registros com chaves iguais na mesma posição relativa).

Sumário

Bubble sort (ordenação por flutuação)

Estratégia

Percorrer a lista de elementos diversas vezes, a cada passagem fazendo "flutuar" para o final o maior elemento da sequência.
Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

```
passo 1
   7 9 6 1
   7 9 6 1
   7 6 9 1
   7 6 1 9
```

```
passo 2
   7 6 1 9
   6 7 1 9
   6 1 7 9
```

```
passo 3
   6 1 7 9
   1 6 7 9
   1 6 7 9
```

Bubble sort (ordenação por flutuação)

Estratégia

Percorrer a lista de elementos diversas vezes, a cada passagem fazendo "flutuar" para o final o maior elemento da sequência.

```
Bubble_Sort (A, tamanho)
    para i de tamanho ate 2 faca
        para j de 1 ate (i-1) faca
            se (A[j] > A[j+1]) entao
                temp = A[j+1]
                A[j+1] = A[j]
                A[j] = temp
```

Bubble sort: análise

Comparações

- O número de comparações realizados na i-ésima iteração não depende da disposição dos dados. No loop mais interno, o número de comparações realizadas por iteração ($C_i(n)$) é:
  - melhor caso: $C_i(n) = i - 1$
  - pior caso: $C_i(n) = i - 1$
  - caso esperado: $C_i(n) = i - 1$
- No entanto, a cada etapa o método realiza uma comparação a menos. O número total de comparações seria, então:
  - casos melhor, pior e médio: $C(n) = (n-1) + (n-2) + \cdots = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx n^2$

Análise assintótica

- O algoritmo *bubblesort* é $O(n^2)$ tanto no pior quanto no melhor e no caso esperado.

Bubble sort: análise

...comentáros...

- O algoritmo original consome muito tempo mesmo quando todos os itens estão ordenados.
- É possível realizar uma pequena modificação para que, em arquivos ordenados, o algoritmo descubra a um custo $O(n)$ que cada item já está em seu lugar.
- No entanto, o bubblesort não tem a mesma performance do insertion sort para arquivos quase ordenados.
- Segundo Donald Knuth, "the bubble sort seems to have **nothing** to recommend it, except a **catchy name** and the fact that it leads to some interesting theoretical problems" (grifos meus)

Exercícios

(1) Implemente o insertion sort e o bubble sort em `C` e, utilizando o mesmo conjunto de dados (um arranjo de n posições), meça o tempo que cada um demora para ordenar os dados (OBS: note que ambos tem a mesma eficiência assintótica).

(3) Faça a análise do número de movimentações realizadas pelo *bubblesort*

Bibliografia

- ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C (Capítulo 4). 2.ed. Thomson, 2004.
- CORMEN, T.H. et al. **Algoritmos: Teoria e Prática** (Capítulo 2). Campus. 2002.
- FEOFILOFF, P. **Ordenação: algoritmos elementares**. Disponível em: `http://www.ime.usp.br/~pf/algoritmos/aulas/ordena.html`.