Métodos de Ordenação

Parte 1

SCC-214 Projeto de Algoritmos

Prof. Thiago A. S. Pardo

Baseado no material do Prof. Rudinei Goularte

O Problema da Ordenação

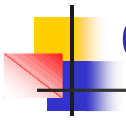
- Ordenação (ou classificação) é largamente utilizada
  - Listas telefônicas e dicionários
  - Grandes sistemas de BD e processamento de dados
    - 25% da computação em ordenação
  - Algoritmos de ordenação são ilustrativos
    - Como resolver problemas computacionais
    - Como lidar com estruturas de dados
    - Como desenvolver algoritmos elegantes e como analisar e comparar seus desempenhos

2

O Problema da Ordenação

- Ordenar (ou classificar)
  - *Definição: organizar uma seqüência de elementos de modo que os mesmos estabeleçam alguma relação de ordem*
    - *Diz-se que os elementos $k_1,\ldots,k_n$ estarão dispostos de modo que $k_1 \leq k_2 \leq \ldots \leq k_n$*
  - Facilita a busca/localização/recuperação de um elemento dentro do conjunto a que pertence
    - Será?

3



O Problema da Ordenação

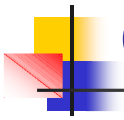
- Ocasionalmente, dá menos trabalho buscar um elemento em um conjunto desordenado do que ordenar primeiro e depois buscar
- Por outro lado, se a busca for uma operação freqüente, vale a pena ordenar
  - A classificação pode ser feita somente uma vez!
- Depende das circunstâncias!

4

O Problema da Ordenação

- Terminologia/conceitos
  - Ordenação de **registros** (em um "arquivo"), em que cada registro é ordenado por sua **chave**
  - Ordenação **interna** vs. **externa**
  - Ordenação **estável**: ordenação original de registros com mesma chave é preservada após a ordenação dos registros

5



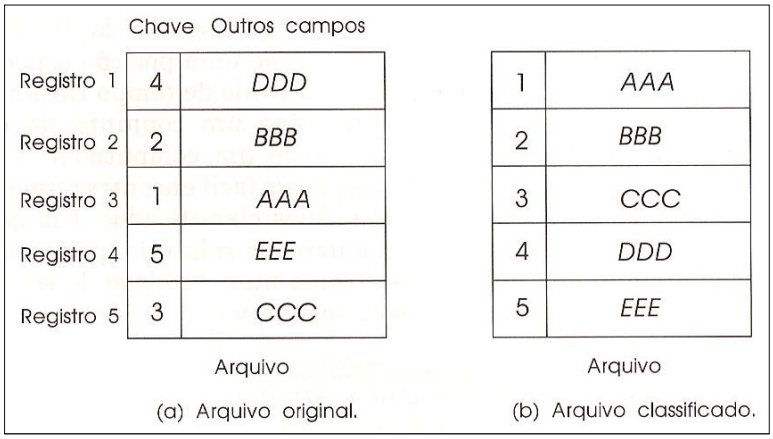
O Problema da Ordenação

- Terminologia/conceitos
  - Ordenação sobre os próprios registros
    - Os registros são trocados de posição
  - Ordenação por endereços
    - Mantém-se uma tabela de ponteiros para os registros e alteram-se somente os ponteiros durante a ordenação

6

O Problema da Ordenação

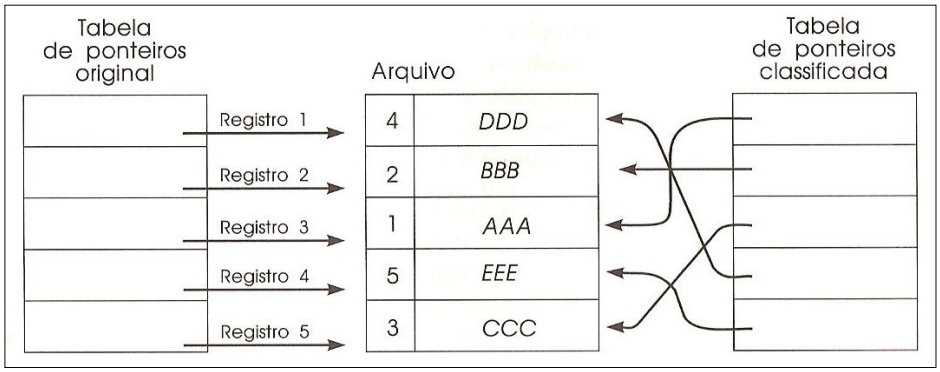
- Exemplo: ordenação sobre os próprios registros



7

O Problema da Ordenação

- Exemplo: ordenação por endereços

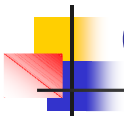


8

O Problema da Ordenação

- Terminologia/conceitos
  - Registros a serem ordenados podem ser complexos ou não
    - Exemplos
      - Dados de empregados de uma empresa, sendo que a ordenação deve ser pelo RG do empregado
      - Números inteiros
    - Métodos de ordenação independem desse fator!

9



O Problema da Ordenação

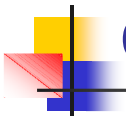
- Existem vários meios de implementar ordenação
- Dependendo do problema, um algoritmo apresenta vantagens e desvantagens sobre outro
- Como comparar?

10

O Problema da Ordenação

- Devemos comparar as complexidades dos algoritmos
- Qual a operação dominante?

11



O Problema da Ordenação

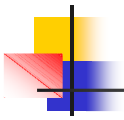
- Devemos comparar as complexidades dos algoritmos
- Qual a operação dominante?
  - Número de comparações entre elementos, na maioria dos casos
    - Somente as comparações que podem resultar em trocas

12

Algoritmos de Ordenação

- Tradicionalmente, nos estudos dos métodos de ordenação, assume-se que a entrada dos algoritmos é um vetor de números inteiros
  - Procura-se ordem crescente

13



Algoritmos de Ordenação Baseados em Troca

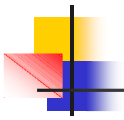
- Mais conhecidos algoritmos baseados em troca
  - Bubble-sort, também chamado método da bolha
  - Quick-sort, ou ordenação rápida ou, ainda, ordenação por troca de partição

14

Bubble-sort

- É um dos métodos mais conhecidos e intuitivos
- Idéia básica
  - Percorrer o vetor várias vezes
  - A cada iteração, comparar cada elemento com seu sucessor (vetor[i] com vetor[i+1]) e trocá-los de lugar caso estejam na ordem incorreta

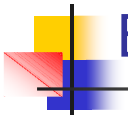
15



Bubble-sort: um passo

- X = (25 , 57 , 48 , 37 , 12 , 92 , 86 , 33)
  - X[0] com X[1] (25 com 57) não ocorre permutação
  - X[1] com X[2] (57 com 48) ocorre permutação
  - X[2] com X[3] (57 com 37) ocorre permutação
  - X[3] com X[4] (57 com 12) ocorre permutação
  - X[4] com X[5] (57 com 92) não ocorre permutação
  - X[5] com X[6] (92 com 86) ocorre permutação
  - X[6] com X[7] (92 com 33) ocorre permutação

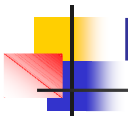
16



Bubble-sort

- Depois do primeiro passo
  - vetor = (24 , 48 , 37 , 12 , 57 , 86 , 33 , 92)
  - O maior elemento (92) está na posição correta
- Para um vetor de n elementos, são necessárias n-1 iterações
- A cada iteração, os elementos vão assumindo suas posições corretas
  - Por que se chama método das bolhas?

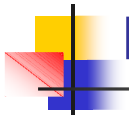
17



Bubble-sort

- Exercício em grupos de 2 (valendo nota)
  - Para entregar
    - Implementar bubble-sort
    - Calcular complexidade do algoritmo

18

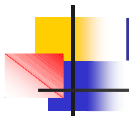


Bubble-sort

- Que melhorias podem ser feitas?

passo 0 (vetor original)	25 57 48 37 12 92 86 33
passo 1	25 48 37 12 57 86 33 92
passo 2	25 37 12 48 57 33 86 92
passo 3	25 12 37 48 33 57 86 92
passo 4	12 25 37 33 48 57 86 92
passo 5	12 25 33 37 48 57 86 92
passo 6	12 25 33 37 48 57 86 92
passo 7	12 25 33 37 48 57 86 92

19



Bubble-sort aprimorado

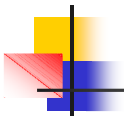
- Detectar quando o vetor já está ordenado
  - Isso ocorre quando, em um determinado passo, nenhuma troca é realizada
- Após o passo j, garante-se que o elemento vetor[n-j] está em sua posição correta
  - Para um vetor de n elementos são necessárias n-j iterações

20

Bubble-sort: exercício

- Implementação do bubble-sort aprimorado

21



Bubble-sort aprimorado

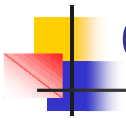
- Número de comparações na iteração j é n-j:
  - $(n-1) + (n-2) + (n-3) + \ldots (n-k) = (2kn - k^2 - k) / 2$
  - Número de iterações para k=n: $(2kn - k^2 - k) / 2 = (2n^2 - n^2 - n)/2 = ½(n^2 - n) = O(n^2)$, para médio e pior caso
- E se o vetor já estiver ordenado?
- E a complexidade de espaço?

22

Bubble-sort aprimorado

- Número de comparações na iteração j é n-j:
  - $(n-1) + (n-2) + (n-3) + \ldots (n-k) = (2kn - k^2 - k) / 2$
  - Número de iterações para k=n: $(2kn - k^2 - k) / 2 = (2n^2 - n^2 - n)/2 = ½(n^2 - n) = O(n^2)$, para médio e pior caso
- E se o vetor já estiver ordenado? O(n), para melhor caso
- E a complexidade de espaço? O(n)

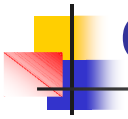
23



Quick-sort

- Melhoramento do bubble-sort
  - Troca de elementos distantes são mais efetivas
- Idéia básica: dividir para conquistar
  - Dividir o vetor em dois vetores menores que serão ordenados independentemente e combinados para produzir o resultado final

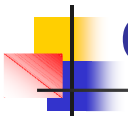
24



Quick-sort

- Considere um vetor v de n posições
- Primeiro passo
  - Elemento pivô: x (escolha do pivô é importantíssima)
    - Colocar x em sua posição correta
    - Ordenar de forma que os elementos à esquerda do pivô são menores ou iguais a ele e os elementos à direita são maiores ou iguais a ele
      - Percorrer o vetor v da esquerda para a direita até v[i] >= x; e da direita para a esquerda até v[j] <= x
        - Troca v[i] com v[j]
      - Quando i e j se cruzarem, a iteração finaliza, de forma que v[0]...v[j] são menores ou iguais a x e v[i]...v[n-1] são maiores ou iguais a x
- Segundo passo
  - Ordenar sub-vetores abaixo e acima do elemento pivô

25



Quick-sort: exemplo

25 57 35 37 12 86 92 33

26

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 ponteiros inicializados
i j

27

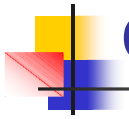
Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 ponteiros inicializados
i j

25 57 35 37 12 86 92 33 procura-se i>=pivô
i j

28

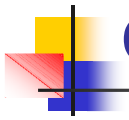


Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 i → ← j	ponteiros inicializados
25 57 35 37 12 86 92 33 i j	procura-se i>=pivô
25 57 35 37 12 86 92 33 i j	procura-se j<=pivô

29



Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 i → ← j	ponteiros inicializados
25 57 35 37 12 86 92 33 i j	procura-se i>=pivô
25 57 35 37 12 86 92 33 i j	procura-se j<=pivô
25 33 35 37 12 86 92 57 i j	***troca***

30

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 i → ← j	ponteiros inicializados
25 57 35 37 12 86 92 33 i j	procura-se i>=pivô
25 57 35 37 12 86 92 33 i j	procura-se j<=pivô
25 33 35 37 12 86 92 57 i j	***troca***
25 33 35 37 12 86 92 57 i j	procura-se i>=pivô

31

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 i → ← j	ponteiros inicializados
25 57 35 37 12 86 92 33 i j	procura-se i>=pivô
25 57 35 37 12 86 92 33 i j	procura-se j<=pivô
25 33 35 37 12 86 92 57 i j	***troca***
25 33 35 37 12 86 92 57 i j	procura-se i>=pivô
25 33 35 37 12 86 92 57 i j	procura-se j<=pivô

32

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 i → ← j	ponteiros inicializados
25 57 35 37 12 86 92 33 i j	procura-se i>=pivô
25 57 35 37 12 86 92 33 i j	procura-se j<=pivô
25 33 35 37 12 86 92 57 i j	***troca***
25 33 35 37 12 86 92 57 i j	procura-se i>=pivô
25 33 35 37 12 86 92 57 i j	procura-se j<=pivô
25 33 35 12 37 86 92 57 i j	***troca***

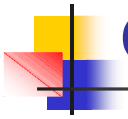
33

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 ij	procura-se i>=pivô

34

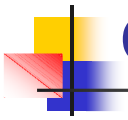


Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 — procura-se i>=pivô
i j

25 33 35 12 37 86 92 57 — procura-se j<=pivô
i j
→ como i e j se cruzaram, fim do processo



Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 — procura-se i>=pivô
i j

25 33 35 12 37 86 92 57 — procura-se j<=pivô
i j
→ como i e j se cruzaram, fim do processo

Todos à esquerda do pivô são menores ou iguais a ele
→ v[0]...v[j]<=pivô

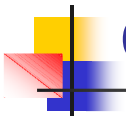
Todos à direita do pivô são maiores ou iguais a ele
→ v[i]...v[n-1]>=pivô

Quick-sort: exercício

- Fazer as ordenações dos subvetores, repetindo o processo

25 33 35 12 (37) 86 92 57

37



Quick-sort: exercício

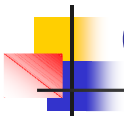
- Implementação do quicksort

38

Quick-sort

- Complexidade
  - Se vetor já ordenado com escolha do pivô como um dos extremos (elemento 0 ou n-1)
    - Subvetores desiguais, com n chamadas recursivas da função partição, eliminando-se 1 elemento em cada chamada
    - Cada chamada recursiva faz n comparações
      - $T(n)=O(n^2)$, no pior caso
        - Igual ao bubble-sort

39



Quick-sort

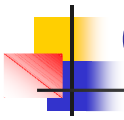
- Complexidade
  - Se a escolha do pivô divide o arquivo em partes iguais
    - O vetor de tamanho n é dividido ao meio, cada metade é dividida ao meio, ..., m vezes => $m \approx \log_2 n$
    - Cada parte do vetor realiza n comparações (com n = tamanho da partição atual do vetor)
    - Pelo método da árvore de recorrência, tem-se que $T(n)=O(n \log_2 n)$, no melhor caso

40

Quick-sort

- Complexidade
  - Caso médio (Sedgewick e Flajelot, 1996)
    - $T(n) \approx 1{,}386n \log_2 n - 0{,}846n = O(n \log_2 n)$

41



Quick-sort

- Complexidade
  - Um dos algoritmos mais rápidos para uma variedade de situações, sendo provavelmente mais utilizado do que qualquer outro algoritmo
  - Pergunta: como evitar o pior caso?

42

Quick-sort

- Complexidade
  - Um dos algoritmos mais rápidos para uma variedade de situações, sendo provavelmente mais utilizado do que qualquer outro algoritmo
  - Pergunta: como evitar o pior caso?
    - Boa abordagem: escolher 3 elementos quaisquer do vetor e usar a mediana deles como pivô
    - Alternativa: escolha aleatória do pivô

43