

Introdução à Ciência da Computação II

Ordenação Multi-Chaves & Digital

Prof. Ricardo J. G. B. Campello

Aula de Hoje

- ◆ Ordenação com Múltiplas Chaves
- ◆ Ordenação Digital
 - Bucket-Sort
 - Radix-Sort

Ordenação Multi-Chaves

- ◆ Um conjunto de elementos quaisquer pode ser ordenado de formas distintas, dependendo da **chave** utilizada para a ordenação
- ◆ Por exemplo, os alunos da classe podem ser ordenados por nome, idade, no. usp, ...
- ◆ Porém, dada uma chave para a ordenação, como resolver empates (ou seja, elementos com chaves iguais) ???

3

Ordenação Multi-Chaves

- ◆ Em muitas aplicações, a ordem relativa de elementos com chaves iguais não importa
 - Qualquer algoritmo de ordenação pode ser usado
- ◆ Em outras aplicações, apenas requer-se que a ordem relativa original seja mantida
 - Nesse caso, deve-se utilizar um algoritmo estável
- ◆ Entretanto, existem aplicações que exigem o uso hierárquico de um conjunto de chaves...
 - Chaves com ordem de prioridade...

4

Ordenação Multi-Chaves

- ◆ A própria ordenação lexicográfica de strings pode ser vista como um exemplo:
 - ordem é determinada prioritariamente pelo 1º caractere
 - em caso de empate, considera-se o 2º caractere
 - e assim por diante...
- ◆ Procedimentos de ordenação que utilizam múltiplas chaves são denominados ordenação **multi-chaves** ou **multi-critérios**
 - Existem basicamente duas abordagens para o problema

5

Ordenação Multi-Chaves

- ◆ **Abordagem 1:** ordenar sucessivas vezes, utilizando uma chave por vez, iniciando com a chave de **menor** prioridade e finalizando com a de **maior** prioridade
 - a ordenação com cada chave serve para resolver eventuais empates com a chave imediatamente acima em importância
 - deve-se utilizar um algoritmo **estável**:
 - ◆ assim, cada ordenação não irá alterar as ordens relativas entre elementos estabelecidas pelas ordenações anteriores
- ◆ Exemplo: (20, A), (10, Q), (35, F), (10, C), (20, X), (45, Q), (35, G)
 - no quadro...

6

Ordenação Multi-Chaves

- ◆ A desvantagem da abordagem anterior é que ela demanda m ordenações se existirem m chaves
- ◆ É possível, no entanto, resolver o problema com apenas um procedimento de ordenação...
- ◆ **Abordagem 2:** construir uma única chave de ordenação composta das m chaves individuais
 - Cada comparação poderá ser computacionalmente mais cara
 - No entanto, a demanda por uma única execução do algoritmo de ordenação deverá compensar tal custo adicional
 - Além disso, **o algoritmo não precisa ser estável !**

7

Exemplo da Abordagem 2 (Skiena & Revilla, 2003)

- ◆ **Ordenar os pretendentes de "Polly"**
 - Primeiro critério:
 - ◆ Altura mais próxima possível de 180cm
 - ◆ Não importa se para mais ou para menos
 - Segundo critério (em caso de empate no primeiro):
 - ◆ Peso mais próximo possível de 75Kg, não mais que isso
 - ◆ O mais leve caso todos estejam acima desse peso
 - Terceiro critério (em caso de empate nos anteriores):
 - ◆ Ordem pelo último nome
 - ◆ Ordem pelo primeiro nome

8

Exemplo da Abordagem 2 (Skiena & Revilla, 2003)

◆ Entrada:

George Bush	195	110
Harry Truman	180	75
Bill Clinton	180	75
John Kennedy	180	65
Ronald Reagan	165	110
Richard Nixon	170	70
Jimmy Carter	180	77

◆ Saída:

Clinton, Bill
Truman, Harry
Kennedy, John
Carter, Jimmy
Nixon, Richard
Bush, George
Reagan, Ronald

9

Exemplo da Abordagem 2 (Skiena & Revilla, 2003)

- ◆ Para resolver esse problema, podemos inicialmente construir as chaves individuais

- **chave_altura =**
 $\text{abs}(\text{altura_ideal} - \text{altura_candidato})$

- Por exemplo:

- ◆ $\text{chave_altura}(\text{Bill Clinton}) = 0$
 - ◆ $\text{chave_altura}(\text{Richard Nixon}) = 10$
 - ◆ $\text{chave_altura}(\text{George Bush}) = 15$

10

Exemplo da Abordagem 2 (Skiena & Revilla, 2003)

◆ Continuando...

- **chave_peso =**
 - [**(peso_ideal – peso_candidato)** se $\text{peso_candidato} \leq 75$
 - [**peso_candidato** caso contrário
- Por exemplo:
 - ◆ $\text{chave_peso}(\text{Bill Clinton}) = 0$
 - ◆ $\text{chave_peso}(\text{John Kennedy}) = 10$
 - ◆ $\text{chave_peso}(\text{Jimmy Carter}) = 77$

- ◆ Tomando a ordem dos nomes como lexicográfica, todas as 4 chaves serão inversamente proporcionais às preferências de Polly...

11

Exemplo da Abordagem 2 (Skiena & Revilla, 2003)

- ◆ Logo, poderíamos aplicar a abordagem 1, ordenando 4 vezes os pretendentes com um algoritmo estável e as 4 chaves (na ordem nome, sobrenome, peso e altura)
- ◆ Para aplicar a **abordagem 2**, precisamos compor uma única chave a partir das 4 chaves individuais...
 - basta definir um registro com as 4 chaves como campos
 - e definir uma função capaz de comparar esses registros
 - ◆ esse tipo de função é denominada de **comparador**

12

Exemplo da Abordagem 2 (Skiena & Revilla, 2003)

◆ Exemplo de composição de chave:

```
typedef struct {
    float c_altura;           /* chave p/ altura */
    float c_peso;             /* chave p/ peso   */
    char c_nome[10];          /* chave nome      */
    char c_sobrenome[10];     /* chave sobrenome */
} chave;
```

13

Exemplo da Abordagem 2 (Skiena & Revilla, 2003)

◆ Exemplo de comparador:

```
int comparador(chave *a, chave *b){
    int aux;
    if (a->c_altura > b->c_altura) return 1;
    if (a->c_altura < b->c_altura) return -1;
    if (a->c_peso > b->c_peso) return 1;
    if (a->c_peso < b->c_peso) return -1;
    if (aux = strcmp(a->c_sobrenome, b->c_sobrenome) != 0)
        return aux;
    return ( strcmp(a->c_nome, b->c_nome) );
}
```

14

Outros Paradigmas de Ordenação

- ◆ Todos os algoritmos de ordenação discutidos no curso são baseados no **paradigma de comparação**
 - comparação de chaves
- ◆ Vimos que não existe algoritmo de ordenação baseado em comparações mais rápido do que $O(n \log n)$
- ◆ Um outro paradigma de ordenação é aquele baseado em **distribuição**
- ◆ Um bom exemplo é aquele de ordenar as cartas de um baralho pela sua importância:

$A < 2 < \dots < 10 < J < Q < K$ e $\spadesuit < \heartsuit < \clubsuit < \diamondsuit$

15

Outros Paradigmas de Ordenação

- ◆ Para tanto basta:
 - distribuir as cartas em 13 montes de 4 cartas cada
 - ◆ ases, dois, três, ..., dez, valetes, damas, reis
 - re-distribuir os montes (pela ordem) em quatro montes dados pelos diferentes naipes
 - reunir seqüencialmente (pela ordem dos naipes) os montes em um único monte
- ◆ Métodos de ordenação por distribuição são também conhecidos como **ordenação digital**
- ◆ Dois algoritmos desse tipo são:
 - **bucket-sort** e **radix-sort**

16

Bucket-Sort

- ◆ Suponha que as chaves dos n elementos a serem ordenados sejam (ou possam ser convertidas para) inteiros no intervalo $[0, N - 1]$
 - n pode ser maior ou menor que N
- ◆ Nesse caso, podemos ordenar esses elementos em tempo $O(n + N)$, sem fazer comparações
- ◆ A idéia básica é colocar o(s) elemento(s) com chave k em uma lista associada à célula $B[k]$ de um vetor B
 - B indexado de 0 a $N - 1$
- ◆ Em seguida, retira-se um a um os elementos de cada célula (possivelmente vazia), obtendo-os em ordem

Bucket-Sort

- ◆ Exemplo (cartas):

4, A, 7, 2, Q, 9, 6, 3, J, 5, 8, K, 10, 5, 4, 10, J, 8, 3, Q, 2, 9, A, 6, K, 7



$B =$

	A	2	3	4	5	6	7	8	9	10	J	Q	K
	A	2	3	4	5	6	7	8	9	10	J	Q	K
0	1	2	3	4	5	6	7	8	9	10	11	12	13



A, A, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, J, J, Q, Q, K, K

Bucket-Sort

- ◆ Retirando os elementos de cada célula na mesma ordem que foram inseridos, garante-se **estabilidade**
 - Se cada célula armazena um ponteiro para uma lista ligada, deve-se fazer as remoções no início (cabeça) da lista e as inserções no final (cauda), ou vice-versa
- ◆ Complexidade:
 - Na 1a etapa percorre-se os n elementos atribuindo-os a B
 - Na 2a etapa percorre-se as N células de B removendo cada um dos n elementos
 - Logo, a complexidade do algoritmo é $O(n + N)$
 - Se $N \ll n$ ou $N \approx n$ ou ainda $N \propto n$ tem-se $O(n)$

Radix-Sort

- ◆ O algoritmo Radix-Sort é essencialmente uma versão multi-critério de Bucket-Sort
 - conforme abordagem 1 vista para ordenação multi-chaves
- ◆ Se ao invés de uma única chave contida no intervalo $[0, N - 1]$ tivermos m chaves, basta executar Bucket-Sort (implementação estável) m vezes
 - utilizando 1 chave diferente a cada execução
 - ◆ iniciando com a chave de menor prioridade
 - ◆ finalizando com a chave de maior prioridade
- ◆ Algoritmo (**Radix-Sort**) executa em tempo $O(m \cdot (n + N))$

Radix-Sort

◆ Exemplo:

- ordenar a seqüência de chaves duplas abaixo utilizando radix-sort (1a chave > prioridade)

(3, 3), (1,5), (2,5), (1,2), (2,3), (1,7), (3,2), (2,2)

- No quadro...

Exercícios

- ◆ Implemente em C um programa que receba como entrada uma lista de "pretendentes de Polly", conforme exemplo nos slides, e ordene essa lista utilizando a **abordagem 1** (use insertion-sort para estabilidade)
- ◆ Repita o exercício anterior utilizando a **abordagem 2**, conforme estratégia de composição de chaves vista em aula. Utilize a rotina `qsort` (quick-sort) de `stdlib.h`
- ◆ Implemente em C o algoritmo Bucket-Sort
 - Sugestão: Inicie com uma implementação ingênua, na qual B é uma matriz $N \times N$. Em seguida melhore a implementação para que B seja um vetor cujas células apontam para listas ligadas

Bibliografia

- ◆ M. T. Goodrich and R. Tamassia, *Data Structures and Algorithms in C++/Java*, John Wiley & Sons, 2002/2005
- ◆ N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Edição, 2004
- ◆ S. Skiena & M. Revilla, *Programming Challenges: The Programming Contest Training Manual*, Springer-Verlag, 2003