Universidade Positivo Sistemas de Informação
11/02/2019 - 12:33:55.9
Algoritmos Prof Dr P Kantek (pkantek@up.edu.br)
Algoritmos de ordenação - família O(n^2)
VIVO132a, V: 1.02
1
Exercício : 1- _____ / _____ / _____

Definição

Seja X um conjunto composto por i elementos entre os quais se pode estabecer uma relação de ordem. Dados X_i e X_j com $i \neq j$, sempre pode-se estabelecer $X_i > X_j$ ou $X_i = X_j$ ou $X_i < X_j$.

- X estará em ordem crescente se e somente se $X_i \leq X_j,\ \forall i < j$.
- X estará em ordem estritamente crescente se e somente se $X_i < X_j,\ \forall i < j$.
- X estará em ordem decrescente se e somente se $X_i \geq X_j,\ \forall i < j$.
- X estará em ordem estritamente decrescente se e somente se $X_i > X_j,\ \forall i < j$.

Um algoritmo de ORDENAÇÃO é aquele que recebe um X qualquer (possivelmente desordenado), permuta seus elementos e devolve X em ordem.

Tipicamente fazem parte de X_i um conjunto de informações. Neste caso, haverá uma parte de X_i denominada CHAVE é identificada por k (k=key) pela qual se fará a ordenação. Embora devamos ter em mente a existência dos outros campos, apenas a chave será tratada nos algoritmos.

Existem 2 classes de algoritmos de ordenação: os que trazem todos os dados para a memória (sort interno) e os que ordenam dados em mídia magnética sequencial (sort externo). Note que neste caso o algoritmo apenas acessa uma parte do subconjunto total.

Um algoritmo de ordenação é estável se no caso particular em que k[i] = k[j], a ordem original dos dados é preservada. É sempre bom que um método de ordenação seja estável.

Antes de prosseguir, vale uma lembrança. Para conjunto de dados pequenos (quanto?) não há necessidade de grandes pesquisas e deve-se usar o algoritmo mais simples possível.

Primeiro critério: buscar efetuar as trocas insitu. Por isso, o algoritmo a seguir deixa a desejar. (Ele movimenta os dados de um vetor para o outro)

Algoritmo de movimentação Este caso é um contra-exemplo. O algoritmo a seguir é um que apresenta uma ordenação ineficiente

```
 1: algoritmo MOVIMENTA (inteiro V[1000])
 2: inteiro Y[1000]
 3: inteiro MAIOR, QUAL
 4: inteiro I,J
 5: leia X
 6: para I de 1 até 1000
 7:     MAIOR ← −∞
 8:     para J de 1 até 1000
 9:         se V[J] > MAIOR
10:             MAIOR ← X[J]
11:             QUAL ← J
12:         fim{se}
13:     fim{para}
14:     Y[I] ← X[QUAL]
15:     V[QUAL] ← −∞
16: fim{para}
17: devolva Y
18: fimalgoritmo
```

Os algoritmos de ordenação, grosso modo, podem ser divididos em 2 categorias: aqueles de complexidade $O(n^2)$ e os de complexidade $O(n \times log_2 n)$, ambas as expressões – como sempre – a menos das constantes multiplicativas.

O estudo da primeira família se justifica pelas seguintes razões:

- Constantes multiplicativas eventualmente menores
- Abundância de recursos de hardware
- Irrelevância do desempenho para pequenas instâncias
- Simplicidade (inclusive conceitual) dos algoritmos envolvidos

Este exercício tratará de 3 algoritmos, além de uma pequena derivação de um deles, todos com desempenho quadrático. São eles:

Bolha Trata-se do algoritmo mais simples que existe. Pares de vizinhos são comparados e se eles estiverem desordenados, são invertidos. A cada passada, o elemento mais leve é levado para a ponta. Daí o nome.

Bolha* O mesmo algoritmo acima, mas com um pequeno truque para encerrar o algoritmo mais cedo caso o vetor a ordenar seja do tipo "quasi-ordenado".

Inserção O mesmo algoritmo usado pelo jogador de buraco ao ordenar as cartas que recebe. Há um monte de cartas desordenadas. As cartas, uma a uma, vão sendo retiradas desse monte e colocadas em ordem na mão do freguês. Quando o monte terminar, a mão está ordenada.

Seleção O menor elemento do vetor a ordenar é intercambiado com o primeiro elemento do vetor. Depois, o segundo menor com o segundo elemento, e assim por diante.

Algoritmo Bolha

```
 1: inteiro I,J,AUX
 2: I ← tamanho do vetor a ordenar
 3: enquanto (I ≥ 2)
 4:     J ← 2
 5:     enquanto (J ≤ I)
 6:         se V[J − 1] > V[J]
 7:             AUX ← V[J − 1]
 8:             V[J − 1] ← V[J]
 9:             V[J] ← AUX
10:         fim{se}
11:         J + +
12:     fim{enquanto}
13:     I − −
14: fim{enquanto}
```

Note as duas características que este algoritmo tem: uma ineficiência grande aliada à simplicidade do código. A grande maioria dos programadores profissionais, conhece "de cabeça"este algoritmo.

Algoritmo Bolha-Estrela

```
 1: inteiro I,J,AUX
 2: lógico MUDOU ← .V.
 3: I ← tamanho do vetor a ordenar
 4: enquanto (I ≥ 2) ∧ MUDOU
 5:     J ← 2
 6:     MUDOU ← .F.
 7:     enquanto (J ≤ I)
 8:         se V[J − 1] > V[J]
 9:             AUX ← V[J − 1]
10:             V[J − 1] ← V[J]
11:             V[J] ← AUX
12:             MUDOU ← .V.
13:         fim{se}
14:         J + +
15:     fim{enquanto}
16:     I − −
17: fim{enquanto}
```

A única diferença em relação ao original é a presença da variável MUDOU que determina o fim do loop externo, quando nenhuma mudança mais for feita.

Algoritmo de Inserção

```
 1: inteiro I, J, X
 2: I ← 2
 3: TAM ← Tamanho do vetor a ordenar
 4: enquanto (I ≤ TAM)
 5:     X ← V[I]
 6:     J ← I − 1
 7:     enquanto (J ≥ 1) ∧ (V[J] > X)
 8:         V[J + 1] ← V[J]
 9:         J − −
10:     fim{enquanto}
11:     V[J + 1] ← X
12:     I + +
13: fim{enquanto}
```

Na linha 5, o elemento que vai ser colocado em seu lugar definitivo é sacado do vetor. Com isso ele abre um espaço para remanejamento dentro do vetor. Isso é feito enquanto $V[J] > X$ nas linhas 8 e 9. Se o sujeito que foi sacado é o menor do vetor, a condição do enquanto é encerrada pela outra condição ($J \geq 1$). Depois que o remanejamento foi feito, o elemento sacado volta para o vetor, agora no seu lugar certo (linha 11).

Algoritmo Seleção

```
 1: inteiro I,J,COR,INUI,AUX
 2: TAM ← Tamanho do vetor a ordenar
 3: I ← 1
 4: enquanto (I ≤ TAM)
 5:     COR ← V[I]
 6:     INUI ← I
 7:     J ← I + 1
 8:     enquanto (J ≤ TAM)
 9:         se V[J] < COR
10:             COR ← V[J]
11:             INUI ← J
12:         fim{se}
13:         J + +
14:     fim{enquanto}
15:     AUX ← V[I]
16:     V[I] ← V[INUI]
17:     V[INUI] ← AUX
18:     I + +
19: fim{enquanto}
```

O menor elemento é procurado (linhas 8 a 14). Encontrado (agora ele é apontando por $INUI$, ele é trocado de lugar com o elemento apontado por I (linhas 15 a 17).

Desempenho

Compare os desempenhos dos 4 algoritmos estudados:

Tam.	Bol	Bol*	Ins	Sel
100 aleat.	0.6	0.7	0.3	0.4
100 q.-o.	0.4	0.06	0.01	0.4
500 aleat.	16	17	7	10
500 q.-o.	11	0.1	0.1	10
1000 aleat.	68	70	31	41
1000 q.-o.	47	0.3	0.2	40
2500 aleat.	429	436	198	255
2500 q.-o.	294	0.6	0.5	255

Obs1: q-o. significa quasi-ordenado, ou seja apenas um elemento está fora de ordem no conjunto.
Obs2: todos os tempos acima estão em segundos.
Obs3: medidas feitas em um Pentium 500Mhz com Windows ME, usando APL*PLUS, versão 6.0, interpretada.

Para você fazer

Estão disponíveis na intranet para esta aula, 6 arquivos denominados ORDE01.ord até ORDE06.ord Cada um deles contém 5000 registros de 40 bytes cada. Desses 40 bytes, os primeiros 4 correspondem a um número binário e deverão ser interpretados como a chave dos registros. Com base nos algoritmos acima, crie programas em C capazes de implementar os 4 algoritmos estudados (bolha, bolha*, seleção e inserção). Não esqueça de imprimir no início e no final do seu programa a hora do processamento ou então use qualquer ferramenta de aferição de tempo.

Arquivo	algoritmo	tempo (seg)
ORDE02	s e l e c a o	1.
ORDE04	bolha	2.
ORDE01	bolh*	3.

IMPORTANTE: junto com esta folha entregue as listagens dos 3 programas produzidos por você.