UNIVERSIDADE ESTADUAL DE MONTES CLAROS

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS – CCET

**LISTA DE EXERCICIOS**

Montes Claros, Minas Gerais

Maio de 2012

Patrick Pierre Fernandes Ferreira

**ALGORITMOS DE ORDENAÇÃO**

Lista de exercicios apresentado ao professor Renato Cota, como parte das exigências para avaliação da disciplina de Algoritmos e programação II, 2º período.

Montes Claros – MG

Maio/2012

**1.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3’ | 3” | 1 | 2 |
| 1 | 3’ | 3” | 4 | 2 |
| 1 | 2 | 3” | 4 | 3’ |
| 1 | 2 | 3” | 3’ | 4 |

Não conserva a ordem de entrada para chaves iguais (3” e 3’).

**2.**

a) Restrições de estabilidade:

Existindo a necessidade de preservar a ordem de entrada para elementos com chaves iguais, é recomendado a utilização do algoritmo por inserção.

b) De tolerância para o pior caso:

Neste caso, o mais recomendado é a utilização do Quicksort, pois este possui melhor desempenho para as diferentes possibilidades de organização dos dados a serem ordenados.

**3.**

A mudança a ser feita para que o algoritmo Quicksort se torne estável é somente a troca do sinal de comparação “<=” por “<” no “do { ... } while” da função Partição.

**4.**

a) [ A B A ] [ B A B A ]

b) [ Q U I C K S O R T ]

[ C U I Q K S O R T ]

[ C I U Q K S O R T ]

[C I K ] [ Q U S O R T ]

[C I K ] [ Q R S O U T ]

[C I K ] [ Q R O] [ S U T ]

[C I K ] [ O Q R ] [ S T U ]

[C I K O Q R S T U ] (ordenado)

**5.**

O Bubblesort, também conhecido como “método por bolha” é um dos algoritmos de ordenação mais simples existente. Porém, sendo um algoritmo de ordem n², possui uma velocidade de ordenação lenta, não sendo recomendado na maioria dos casos (principalmente lidando com grandes quantidades de dados).

void bubblesort(int \*v, int n){

int i, j, aux;

for(i=0; i<n; i++){

for(j=0; j<n-1; j++){

if(v[j]>v[j+1]) {

aux=v[j];

v[j]=v[j+1];

v[j+1]=aux;

}

}

}

}

Como mostrado acima, a ideia do bubblesort é verificar cada chave colocada no vetor com as próximas, encontrando uma chave maior que a primeira, por exemplo, ele irá trocar de lugar com o primeiro, em seguida, ele continua o processo verificando o novo primeiro item com os outros n itens do vetor. Feito isso, ele começa a comparação com o segundo item do vetor, verificando com todos acima deste, e assim por diante até que o “i”, acima, alcance o limite de membros listados.

**6.**

|  |  |  |
| --- | --- | --- |
| Método de ordenação: | Vantagens | Desvantagens |
| Seleção | - Custo linear para número de movimentos;  - É utilizado para arquivos com registros muito grandes e recomendado para arquivos pequenos. | - O arquivo já ordenado não ajuda na ordenação, efetuando sempre n² comparações;  - Não é um algoritmo estável |
| Inserção | - Algorítmo estável;  - Menor tempo de execução, comparações e trocas para registros já ordenados, sendo escolhido se o vetor já estiver quase ordenado;  - Muito bom quando é necessário adicionar alguns poucos itens a um arquivo já ordenado. | - Maior tempo de execução, comparações e trocas se o vetor estiver em ordem reversa. |
| Shellsort | - Ótima escolha para arquivos de tamanho moderado;  - Implementação simples com código pequeno. | - Tempo de execução do algoritmo é sensível à ordem inicial do arquivo;  - Não é estável. |
| Quicksort | - É extremamente eficiente para ordenar dados, independente de posição;  - Requer cerca de nlogn comparações em média para ordenar n itens.  - Um dos algoritmos mais rápidos para as diferentes possibilidades de disposição dos dados. | - Pior caso O(n²), quando o pivô escolhido fica situado na primeira posição;  - Não é estável. |
| Heapsort | - Sua ordem é sempre O(nlogn), independente da ordem de entrada dos dados. | - Código extremamente complexo, comparado ao Quicksort, se faz inviável;  - Não é estável. |

**7.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3’ | 3” | 1 | 2 |
| 3’ | 4 | 3” | 1 | 2 |
| 3’ | 3” | 4 | 1 | 2 |
| 1 | 3’ | 3” | 4 | 2 |
| 1 | 2 | 3’ | 3” | 4 |

Preserva ordem de entrada para chaves iguais (3’ e 3”).

Arvore Binária:

0

1

2

3

4

4

3’

3”

1

2

4

3’

3”

1

2

4

3’

3”

1

2

2

3’

3”

1

4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3’ | 3” | 1 | 2 |
| 4 | 1 | 3” | 3’ | 2 |
| 1 | 4 | 3” | 3’ | 2 |
| 1 | 2 | 3” | 3’ | 4 |

Não preserva a ordem de entrada para chaves iguais (3” e 3’).

**8.**

O algoritmo apresentado implementa o método Shellsort.

**10.**

- Primeiramente, para ordenar os 1.000.000 itens do banco de dados, seria melhor a implementação do algoritmo Quicksort, pois este apresenta mais facilidade de implementação que o Heapsort e um tempo de ordenação extremamente baixo.

- No segundo caso, a inserção de mais 500.000 aleatórios nós poderíamos mudar um pouco o método implementado. Como os 1.000.000 primeiros itens já estão ordenados, não precisamos mais utilizar um Quicksort no conjunto todo dos elementos, mas poderíamos aplicá-lo somente aos 500.000 novos itens, diminuindo bastante o número de comparações e trocas realizadas, e utilizarmos um algoritmo por Inserção para ordenar o conjunto completo dos itens. Somente o inserção também teria efeito bem melhor, sendo mais recomendado para conjuntos de dados quase ordenados.

- Para que todas as inserções fossem feitas de maneira ordenada poderíamos utilizar o método de alocação por lista encadeada, assim, seria necessário somente adicionarmos um novo item na posição desejada sem precisar deslocar os demais itens.

**11.**

Shellsort:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 3 | 8 | 6 | 5 | 2 | 4 | 1 | 7 |
| 2 | 8 | 6 | 5 | 3 | 4 | 1 | 7 | H=4 |
| 2 | 4 | 6 | 5 | 3 | 8 | 1 | 7 |
| 2 | 4 | 1 | 5 | 3 | 8 | 6 | 7 |
| 1 | 4 | 2 | 5 | 3 | 8 | 6 | 7 | H=2 |
| 1 | 4 | 2 | 5 | 3 | 7 | 6 | 8 |
| 1 | 2 | 4 | 5 | 3 | 7 | 6 | 8 | H=1 |
| 1 | 2 | 4 | 3 | 5 | 7 | 6 | 8 |
| 1 | 2 | 3 | 4 | 5 | 7 | 6 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**12.**

[ K H Z Y S A I R M ] h = 5

[ A H Z Y S K I R M ]

[ A H R Y S K I R M ]

[ A H R M S K I Z Y ] h = 3

[ A H K M S R I Z Y ]

[ A H K I S R M Z Y ] h=1

[ A H I K S R M Z Y ]

[ A H I K R S M Z Y ]

[ A H I K R M S Z Y ]

[ A H I K M R S Z Y ]

[ A H I K M R S Y Z ] (ordenado)