

Estrutura de Dados

Métodos de Ordenação por Troca

Alessandro Ferreira Leite

12/03/2012

1 Classificação por Trocas

- Método da Bolha - Bubblesort
 - Implementação
 - Análise de Desempenho
- Método de Partição e Troca - QuickSort
 - Implementação

Classificação por Trocas

- Os métodos de classificação por trocas caracterizam-se por efetuarem a classificação por comparação entre pares de chaves, trocando-as de posição caso estejam fora de ordem no par.

Método da Bolha - Bubblesort

- Neste método, o princípio geral é aplicado a todos os pares consecutivos de chaves.
- Esse processo é executado enquanto houver pares consecutivos de chaves não ordenados.
- O processo finaliza quando não mais restarem pares não ordenados.
- O método da bolha é um dos algoritmos mais simples que existem.

Método da Bolha - Bubblesort

- **Exemplo:** Suponha que se deseje classificar em ordem crescente o seguinte vetor:

28	26	30	24	25
----	----	----	----	----

- 1 Comparamos todos os pares de chaves consecutivas, a partir do par mais à esquerda ($V_i = 0$).
- 2 Caso as chaves de um certo par encontrem fora da ordem desejada, efetuamos a troca das mesmas.
- 3 Ao processo de comparação das $n - 1$ chaves denominamos de **varredura**.
- 4 O método efetuará tanto varreduras quanto forem necessárias para que todos os pares consecutivos de chaves se apresentem na ordem deseja.

Método da Bolha - Bubblesort

28	26	30	24	25	compara par(28,26) : troca
26	28	30	24	25	compara par(28,30) : não troca
26	28	30	24	25	compara par(30,24) : troca
26	28	24	30	25	compara par(30,25) : troca
26	28	24	25	30	fim da primeira varredura

- Como ainda existem pares não ordenados, reiniciamos o processo de comparações de pares de chaves, executando mais uma varredura.
- Ao término da 1ª varredura, a chave de maior valor já está posicionada na sua posição definitiva.
- Isto significa que na segunda varredura podemos desconsiderar a última posição do vetor, que portanto fica reduzido de um elemento.

Método da Bolha - Bubblesort

- Segunda varredura

26	28	24	25	30	compara par(26,28) : não troca
26	28	24	25	30	compara par(28,24) : troca
26	24	28	25	30	compara par(28,25) : troca
26	24	25	28	30	fim da segunda varredura

- Terceira varredura

26	24	25	28	30	compara par(26,24) : troca
24	26	25	28	30	compara par(26,25) : troca
24	25	26	28	30	fim da terceira varredura

Método da Bolha - Bubblesort

- A denominação desse método resulta da associação das chaves com bolhas dentro de um fluido.
- Cada bolha teria um diâmetro proporcional ao valor de uma chave.
- Dessa forma, as bolhas maiores subiriam com velocidades maiores, o que faria com que, após um certo tempo, elas se arrandassem em ordem de tamanho.

• Algoritmo

- 1 Percorra o vetor inteiro comparando os elementos adjacentes (*dois a dois*).
- 2 Troque as posições dos elementos se eles estiverem fora de ordem.
- 3 Repita os dois passos acima com os primeiros $n - 1$ elementos, depois com os primeiros $n - 2$, até que reste apenas um elemento.

Implementação - Bubblesort

```
/**
 * Classifica o vetor v, em ordem crescente ,
 * utilizando o método da bolha.
 */
void bubblesort(int v[]){
    boolean troca = true;
    int m = |v| // tamanho do vetor v
    int k = 1; // indica a posição onde ocorreu a última troca
    int i;
    while (troca){
        troca = false;
        for (i = 0; i < m ; i++){
            if (v[i] > v[i + 1]){
                int ch = v[i];
                v[i] = v[i + 1];
                v[i + 1] = ch;
                k = i; // posição da última troca.
                troca = true;
            }
        }
        m = k; // vetor já ordenado de m + 1 até n.
    }
}
```

Análise de Desempenho - Bubblesort

- O pior caso acontece quando os elementos do vetor encontram-se na ordem inversa à desejada.
- Nesses casos, a cada varredura, apenas uma chave será colocada no seu local definitivo.
- Desse modo, as quantidades de comparações que serão efetuadas a cada varredura são as seguintes:

nº da varredura	comparações efetuadas
1	$n - 1$
2	$n - 2$
3	$n - 3$
\vdots	\vdots
$n - 1$	1

- Portanto, o número de comparações efetuadas será a soma do número de comparações de cada varredura.

$$C(n) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \frac{n^2 - n}{2}$$

- Sendo n^2 a parcela dominante, o método é de complexidade quadrática, ou seja, $O(n^2)$.

QuickSort

- O **quicksort** é um algoritmo de classificação que se baseia no paradigma de **dividir para conquistar**.
- O paradigma de dividir e conquistar pode ser descrito em 3 passos para ordenar um vetor de chaves $V[p..r]$:
 - 1 **Dividir** - O conjunto $V[p..r]$ é particionado em dois subconjuntos $V[p..q-1]$ e $V[q + 1..r]$, tal que cada elemento de $V[p..q - 1]$ é menor ou igual a $v[q]$ que, por sua vez é igual ou menor a cada elemento $V[q + 1..r]$. O índice **q** é calculado como parte desse processo de particionamento.
 - 2 **Conquistar** - Os dois subconjuntos $V[p..q - 1]$ e $V[q + 1..r]$ são ordenados por chamadas recursivas a *quicksort*.
 - 3 **Combinar** - Como os subconjuntos são ordenados localmente, não é necessário nenhum trabalho para combiná-lo.

Idéia Básica

- O algoritmo divide o vetor de chaves em dois subconjuntos, através de um elemento denominado **pivô**, ou elemento de particionamento (k).
- Compara recursivamente os elementos dos dois conjuntos com o **pivô**, e move-os para o conjunto correto, da direita ou esquerda, de modo que **todo elemento a esquerda** de k **seja menor ou igual** a k , e os da direita sejam maior ou iguais a k .

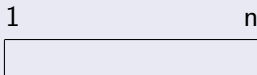
Descrição do método

- Dado um vetor $V[1..n]$ a ser ordenado.
- Primeiro esse vetor é particionado em 3 segmentos, denotados como: S_1 , S_2 , S_3 da seguinte forma:
 - 1 S_2 terá comprimento 1 e conterá a chave denominada particionadora(*pivô*), representada como k .
 - 2 S_1 terá comprimento ≥ 0 e conterá **todas as chaves** cujos valores forem **menores ou iguais** ao da chave particionadora. Esse segmento é particionado à esquerda de S_2 .
 - 3 S_3 também terá comprimento ≥ 0 e conterá todas as chaves cujos valores forem **maiores ou iguais** do que k . Esse segmento é posicionado à direita de S_2 .

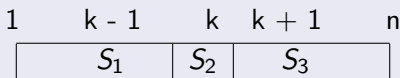
Método de Partição e Troca - QuickSort

Representação do particionamento do vetor $V[1..n]$

vetor inicial:



vetor particionado:



- O processo de particionamento é reaplicado aos subconjuntos S_1 e S_3 e todos os subconjuntos correspondentes daí resultantes que tiverem comprimento ≥ 1 .
- Quando não restarem subconjuntos a serem particionados, o vetor estará ordenado.

Método de Partição e Troca - QuickSort

Processo de Particionamento

- Suponha o vetor $V[1..n]$, abaixo:

9	25	10	18	5	7	15	3
---	----	----	----	---	---	----	---

- Escolhemos a primeira chave, como sendo a **particionadora** e a guardamos em uma variável temporária (k).
- A posição ocupada pela chave particionadora ficará vazia, visualmente indicada pelo símbolo \emptyset .

\emptyset 25 10 18 5 7 15 3 $k = 9$

- Marcamos também o início e o fim do vetor com dois ponteiros: i (de início) e f (de fim).

Método de Partição e Troca - QuickSort

Particionamento - 1º Passo

i								f
↓								↓
\emptyset	25	10	18	5	7	15		3

- Em seguida, comparamos o valor da chave apontada por f com k .
- Se $v[f] < k$, então deslocamos a chave apontada por f para o lado esquerdo do vetor, e avançamos o ponteiro i , para indicar que a chave recém movida já se encontra na posição correta.
- A nova posição vaga passa a ser apontada por f .

	i							f
	↓							↓
3	25	10	18	5	7	15		\emptyset

Particionamento - 2º Passo

- Agora comparamos a chave **25** com k . Como 25 é maior que k , deslocamos para a posição vaga, ao mesmo tempo que recuamos o ponteiro f uma posição para a esquerda, indicando que a chave 25 já se encontra no subconjunto correto.

	i					f	
	↓					↓	
3	∅	10	18	5	7	15	25

Particionamento - 3º Passo

- O processo prossegue comparando a chave **15**. Nesse caso, $15 > k$, não deve ser trocada de posição, pois já se encontra no subconjunto correto. Apenas deslocamos o ponteiro de f para a esquerda.

	i				f		
	↓				↓		
3	∅	10	18	5	7	15	25

Particionamento - 4º Passo

- No passo seguinte a chave 7 é colocada na posição vaga apontada por i , pois $7 < k$, e o ponteiro i é ajustado.

		i			f		
		↓			↓		
3	7	10	18	5	∅	15	25

Particionamento - 5º Passo

- Em seguida a chave 10 é comparada com k . Como $10 > k$, a chave é movida para a posição vazia, apontada por f e recuamos o ponteiro f em uma posição.

		i		f				
		↓		↓				
03	07	∅	18	05	10	15	25	

Particionamento - 6º Passo

- Novamente, comparamos a chave 5 com k . Como $5 < k$, colocamos a chave **5** na posição vazia indicada por i , e avançamos o ponteiro i uma posição. A nova posição vazia agora é a indicada pelo ponteiro f .

			i	f			
			↓	↓			
03	07	05	18	∅	10	15	25

Particionamento - 7º Passo

- Por fim, comparamos a chave 18 com k . Como $18 > k$, colocamos na posição vazia, indica por f , e recuamos o ponteiro f uma posição. A nova posição vazia agora é indicada pelo ponteiro i .

i, f
↓
03 07 05 \emptyset 18 10 15 25

Método de Partição e Troca - QuickSort

Particionamento - 8º Passo

- O resultado do 7º passo (correspondente ao $n - 1$), produz a situação onde os ponteiros i e f se encontram.
- A posição vaga apontada por eles, corresponde à posição do subconjunto S_2 , ou seja, a posição onde o valor de k deve ser inserido.
- Assim, basta copiar o valor de k para a posição apontada por i e f que o processo de particionamento estará concluído.

03 07 05 09 18 10 15 25

- Observe que embora os subconjuntos S_1 e S_3 ainda não estejam ordenados, k já se encontra na posição definitiva.
- O processo de classificação prossegue com o particionamento dos subconjuntos S_1 e S_3 e todos os demais subconjuntos de tamanho maior ou igual a um que forem se formando.

Implementação do Quicksort

```
void quicksort(int v[], int i, int f){
    if (f > i){
        int k = partition(v,i,f);
        quicksort(v,i, k -1);
        quicksort(v, k + 1, f);
    }
}

int partition(int [] v, int p, int r){
    int c = v[r], j = p, k;
    for (k = p; k < r; k++)
        if (v[k] < c){
            trocar(v,j,k);
            j++;
        }
    v[r] = v[j]; v[j] = c;
    return j;
}

void trocar(int v[], int a, int b){
    int t = v[a]; v[a] = v[b]; v[b] = t;
}
```