MÉTODOS DE ORDENAÇÃO: UMA ANÁLISE QUANTITATIVA PARA O PIOR CASO

Schwade, Darlan Eduardo[1]; Mühlbeier, Andreia Rosangela Kessler[2];
Mozzaquatro[3], Patricia Mariotto; Antoniazzi[4], Rodrigo Luiz

Palavras-chave: Métodos de ordenação. Pior caso. Vetor. Desordenação.

Introdução

No decorrer do dia a dia, o ser humano sente a necessidade de consultar dados ordenados. Este fato se dá em virtude da agilidade em ganhar tempo em tarefas rotineiras. Imagine como seria consultar o telefone de uma pessoa se os nomes não estivessem classificados em ordem alfabética. Por isso uma das atividades mais utilizada na computação é a ordenação. As ordens mais utilizadas são as numéricas e as lexicográficas. Existem diversos algoritmos para ordenação interna.

Métodos de Ordenação

A ordenação é uma atividade importante na organização de classes de objetos, visando à sua rápida localização. Assim, ordenação corresponde ao método de rearranjar um conjunto de objetos em uma ordem crescente ou decrescente, com o objetivo de facilitar a recuperação dos itens desse conjunto, tornando-se assim fundamental em processamento de dados (VARELLA, 1996). Existem muitos algoritmos de ordenação, a escolha mais eficiente vai depender de vários fatores, tais como: número de itens a ser classificado; se os valores já estão agrupados em subconjuntos ordenados, desordenados, aleatórios e parcialmente ordenados. A seguir são apresentados os métodos de ordenação analisados na pesquisa proposta.

O método de ordenação *Bubble sort* usa uma estratégia de "comparação e troca", que pode ser aplicada em vários vetores a ser ordenados(OLIVEIRA, 2002).

O método da Inserção direta é considerado um dos métodos mais simples, onde primeiramente, são ordenados os 2 primeiros membros de um vetor. Após, é inserido o 3º elemento

[1] Acadêmico do Curso de Ciência da Computação, UNICRUZ – Universidade de Cruz Alta, e-mail: s.darlaneduardo@gmail.com.
[2] Bacharel em Ciência da Computação, UNICRUZ – Universidade de Cruz Alta, e-mail: andreiamuhlbeier@yahoo.com.br.
[3,4] Professores do Curso de Ciência da Computação, UNICRUZ – Universidade de Cruz Alta, e-mail: patriciamozzaquatro@gmail.com, rodrigoantoniazzi@yahoo.com.br.

na sua posição ordenada com relação aos 2 primeiros. O processo irá continuar até que todos os elementos do vetor estejam devidamente ordenados (ZIVIANI, 2007).

A classificação por seleção é aquela na qual sucessivos elementos são selecionados em sequência e dispostos em suas posições corretas pela ordem. Esses elementos da entrada precisem ser pré-processados acontecendo a seleção ordenada.

O *selection sort* (do inglês, ordenação por seleção) é um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os (n-1) elementos restantes. (TOSCANI; VELOSO, 2002).

O método *shellsort* pode ser considerado um refinamento do método de inserção direta, diferindo pelo fato de no lugar de considerar o vetor a ser ordenado como um único segmento, ele considera vários segmentos sendo aplicado o método de inserção direta em cada um deles. Basicamente o algoritmo passa várias vezes pela lista dividindo o grupo maior em menores (OLIVEIRA, 2002).

O *Shellsort* permite trocas de registros que estão distantes um do outro. Os itens que estão separados *n* posições são rearranjados de forma que todo *n*-ésimo item leva a uma seqüência ordenada.

O método de ordenação *quicksort* é considerado um dos métodos mais rápidos. Consiste em selecionar primeiramente um dos elementos do conjunto a ordenar para ser o elemento pivô, o qual é um elemento já ordenado, em seguida faz uma subdivisão do conjunto inicial em dois subconjuntos, onde o primeiro subconjunto irá conter todos os elementos menor que o elemento pivô, e o segundo subconjunto todos os elementos maiores que o pivô (ZIVIANI, 2007). Esse método utiliza a seleção em árvore para a obtenção dos elementos do vetor na ordem desejada. Consiste em duas fases, onde a primeira monta uma árvore binária (heap) contendo todos os elementos do vetor, de tal forma que o valor contido em qualquer nó seja maior do que os valores de seus filhos. E a segunda utiliza o heap para a seleção dos elementos na ordem desejada (CORMEN, 2002).

Resultados e discussões

É possível identificar que o comportamento dos métodos de ordenação varia conforme o tamanho de entrada, onde a eficiência dos métodos esta relacionada ao tempo de execução, número de comparações e trocas a serem efetuadas. A fim de verificar e analisar o comportamento, aplicou-

se os métodos em uma escala crescente de valores de entrada, em vetores de 4000 a 100000 posições, no pior caso e de forma desordenada.

Os resultados mostram que em todos os casos o método bolha teve um desempenho semelhante ou pior que os demais métodos simples, como inserção direta e seleção, sendo que para vetores pequenos a diferença de processamento é relativamente baixa, porém tratando-se de valores mais abrangentes seu tempo de execução é bem superior. Isto se deve ao fato do bolha realizar muitas trocas ao longo de cada iteração.

Os métodos de inserção direta e seleção também possuem deficiências em seus algoritmos de ordenação, apresentando comportamentos semelhantes ao bolha, embora um pouco mais eficaz. A utilização destes métodos torna-se viável apenas para vetores muito pequenos, mas precisamente na escala de 0 a 100.

Por fim, algoritmos mais otimizados como *Shellsort, QuickSort e HeadSort* têm desempenhos muito superior aos métodos simples tanto no pior caso, como também de forma desordenada, reduzindo consideravelmente seu tempo de execução. Isto se deve ao fato destes algoritmos utilizarem técnicas que reduzem o vetor em análise a cada iteração, com menor número de trocas e algoritmos bem mais otimizados, alguns até utilizando recursividade como o *quicksort*.

A Figura 1 apresentada a seguir ilustra um comparativo após os testes realizados com os métodos de ordenação com um vetor de várias posições.

	Bubblesort	Inserção Direta	Seleção	Shellsort	Quicksort	Heapsort	
Desordenado	00:219	00:202	00:219	00:218	00:203	00:218	1000
Pior caso	00:202	00:218	00:218	00:218	00:203	00:202	
Desordenado	00:421	00:405	00:421	00:390	00:406	00:405	2000
Pior caso	00:436	00:421	00:406	00:406	00:405	00:406	
Desordenado	00:639	00:578	00:593	00:609	00:577	00:577	3000
Pior caso	00:624	00:608	00:592	00:608	00:577	00:577	
Desordenado	00:842	00:780	00:796	00:780	00:765	00:796	4000
Pior caso	00:858	00:812	00:812	00:765	00:780	00:780	
Desordenado	1:187	1:080	1:087	1:052	1:051	1:042	5000
Pior caso	1:172	1:106	1:077	1:038	1:049	1:040	
Desordenado	01:316	01:367	01:262	01:306	01:319	01:355	6000
Pior caso	01:380	01:299	01:245	01:208	01:188	01:200	
Desordenado	01:585	01:427	01:492	01:406	01:422	01:396	7000
Pior caso	01:618	01:560	01:487	01:487	01:418	01:394	
Desordenado	1:982	1:709	1:748	1:687	1:648	1:664	8000
Pior caso	2:002	1:947	1:725	1:689	1:700	1:699	
Desordenado	02:028	01:779	01:904	01:747	01:763	01:747	9000
Pior caso	02:122	01:981	01:825	01:747	01:747	01:748	
Desordenado	2:628	2:368	2:235	2:177	2:096	2:835	10000
Pior caso	2:575	2:370	2:481	2:316	2:269	2:101	
Desordenado	48:248	—	—	—	—	—	100000
Pior caso	1:14:072	51:767	33:269	28:278	21:411	21:484	
Desordenado	---	---	---	---	---	---	200000
Pior caso	5:58:071	2:50:550	1:56:654	47:579	47:316	47:044	
Desordenado	---	---	---	---	---	---	300000
Pior caso	08:35:064	05:47:618	03:07:832	01:10:635	01:16:806	01:15:189	
Desordenado	---	---	---	---	---	---	500000
Pior caso	8:33:064	5:47:618	3:07:832	1:10:635	1:16:806	1:15:189	
Desordenado	---	---	---	---	---	---	1000000
Pior caso	01:24:05:314	00:51:56:854	00:25:48:897	03:58:822	04:08:051	04:27:770	
	H:MM:SS:MIL						

Figura 1 – Comparativo dos Métodos de Ordenação com vetor de várias posições.

Considerando um escala de 1000 a 10000, nota-se que para o pior caso o método bolha é quem apresenta o pior desempenho, principalmente pelo fato de efetuar várias trocas a cada iteração. O melhor desempenho é o método Shell Sort que divide o arquivo original em sub-arquivos, classificando os valores por inserção simples e efetuando o decremento a cada iteração, satisfazendo

o nível 1 quando o vetor encontra-se ordenado. O *Quick Sort e Heap Sort* tem desempenho pouco superior, o primeiro devido à recursividade, o segundo por gastar muito processamento na montagem do head inicial.

No modo desordenado, o bolha também obteve o pior desempenho, enquanto os métodos simples de seleção e *insertion sort* foram satisfatórios em alguns casos. O *quick sort* foi o algoritmo mais eficaz na maioria dos casos.

Em arquivos de tamanho moderado e em grande escala, entre 100000 e 1000000 o método *shell* foi o mais eficiente com desempenho pouco superior aos demais algoritmos otimizados como *quicksort* e *headsort*. Enquanto os métodos simples apresentam desempenho amplamente superior, tornando totalmente ineficiente.

Considerações finais

A análise sobre os resultados processados permitiu avaliar: para vetores de tamanho entre 1000 e 10000 os melhores métodos de ordenação foram *Shellsort* e *Quicksort*, enquanto que o pior método foi o *Bubble sort*. Para vetores de tamanho entre 100000 e 500000 o melhor método de ordenação foi o Heapsort enquanto que o pior método foi o *Bubble sort*. Para vetores de tamanho superior a 500000 o melhor método foi o *Quicksort* e o pior foi o método *Bubble sort*.

Portanto, conclui-se que o pior método de ordenação aplicado na análise desenvolvida foi, em todos os casos, o método da Bolha. Para o pior caso, o método que apresentou melhor desempenho foi o método *Quicksort*.

Referências

CORMEN, Thomas H. *et al*. **Algoritmos: teoria e prática**. Ed.Campus, Rio de Janeiro, 2002.

OLIVEIRA, Alvaro Borges de.. **Métodos de Ordenação Interna**. Visual Book, São Paulo, 1st edition, 2002.

TOSCANI, Laira Vieira; Veloso, Paulo A. S. **Complexidade de algoritmos: análise, projeto e métodos,** 004.421 T713c, 1.ed. Instituto de Informática da UFRGS, 2002.

VARELLA, Irineu Gomes. **Métodos de Ordenação**. São Paulo, 1996.

ZIVIANI, Nivio. **Projeto de Algoritmos com implementação em C++ e Java**. Thomson Learning, São Paulo, 1st edition, 2007.