FACULDADE DE IMPERATRIZ - FACIMP

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ESTRUTURA DE DADOS 2

Vieira Lima Junior

Everson Santos Araujo

ALGORITMOS DE ORDENAÇÃO:

estudo comparativo de diversos algoritmos de ordenação.

Imperatriz (MA)

2003

Everson Santos Araujo

ALGORITMOS DE ORDENAÇÃO:

Estudo comparativo de diversos algoritmos de ordenação.

Trabalho apresentado ao Curso de Sistemas de Informação, destinado à obtenção de nota na matéria de Estrutura de Dados 2.

Orientador: Prof. Vieira Lima Junior

Imperatriz (MA)

2003

Este trabalho é dedicado a todos aqueles que deram a vida por um mundo melhor: livre do LAG, NetSplit e que lutaram pela conexão em Banda Larga. Amém.

"A análise de algoritmos é uma disciplina de engenharia. Um engenheiro civil, por exemplo, tem métodos e tecnologia para prever o comportamento de uma estrutura antes de construí-la. Da mesma forma, um projetista de algoritmos deve ser capaz de prever o comportamento de um algoritmo antes de implementá-lo."

- Anônimo

APRESENTAÇÃO

Este é um trabalho de pesquisa e verificação de diversos métodos de ordenação para sua comparação e apresentação. Neste trabalho não estaremos apresentando soluções para problemas de ordenação, estaremos apenas demonstrando quais são os métodos que podem ser utilizados em determinadas situações e como melhor saber escolher um destes métodos para determinada utilização.

Antes de iniciarmos as discussões sobre os vários métodos de ordenação devemos ter em mente algumas das variáveis que métodos de ordenação devem levar em conta para serem estudados.

Uma ordenação pode ser *in-place* ou não, a ordenação *in-place* é a ordenação que é efetuada com a própria sequência sem utilizar sequências adicionais, este método é utilizado em sistemas onde não se pode gastar mémoria adicional.

A ordenação também pode ser estável ou não, uma ordenação estável se tem quando a posição original de ordem de dois valores iguais é preservada.

SUMÁRIO

1. Bolha

Algoritmo posicional comparativo *in-place*: Seleciona uma posição, compara a sequência a partir desta posição, caso a comparação não esteja ordenada, realiza a troca dos elementos. Realiza a mesma operação recursivamente em todas as posições.

Este sistema pode ser melhor explicado da seguinte maneira, imaginemos um conjunto de elementos desordenados, este método selecionaria cada uma das posições e verificaria nas posições subsequentes a esta se existe um elemento que deveria estar na posição selecionada. Essa verificação é feita através da comparação do valor da posição selecionada com o valor da posição que ele está percorrendo, caso o valor do referente à posição do percurso esteja desordenado com o valor selecionado, ele realiza a troca dos mesmo.

1.1 Utilização

Este algoritmo é desconsiderado em qualquer aplicação de ordenação atual, mas por ser o método mais simples de ser implementado ainda é estudado no meio acadêmico como aprendizado de ordenação.

2. Seleção

Algoritmo incremental comparativo de ordenação *in-place*: Percorre a sequência restante em busca do próximo valor, colocando-o na próxima posição. Esta interação é realizada até que toda a sequência tenha sido percorrida.

Este sistema de ordenação realiza a busca pelo valor seguinte entre os elementos ainda não verificados, utiliza então este valor para ser o próximo valor a ser colocado na parte já ordenada da sequência.

2.1 Utilização

Este é um algoritmo muito simples de implementação que realiza poucas trocas, o que lhe garante utilização em ambientes em que a troca é muito custosa. Em contrapartida, realiza muitas comparações, média de n log n.

Por ser um método comparativo incremental o fato dos elementos já estarem ordenados não o modifica em nada, pois ele continua realizando n*n comparações.

O seu sistema não é estável, pois ele pode realizar troca de elementos de igual valor.

3. Inserção

Algoritmo incremental de ordenação: Insere cada elemento do vetor, na sua posição correta em uma subsequência ordenada de elementos, de modo a mantê-la ordenada.

Esta forma de ordenação seleciona cada um dos elementos de uma sequência e os atribui uma posição relativa a seu valor, de acordo com a comparação com os elementos já ordenados da mesma sequência.

3.1 Utilização

Este é um algoritmo muito interessante quando os elementos já estão ordenados e deseja-se apenas inserir mais um elemento nesse conjunto de dados ordenados, ou quando se tem um conjunto que se imagina estar praticamente ordenado.

O seu pior caso de uso é quando os elementos estão em ordem inversa.

4. Shellsort

Método de ordenação por inserção através de decrementos, esta é uma extensão do método de inserção que pode realizar troca entre elementos que estão distantes um do outro sem precisar realizar a troca adjacente de cada um dos elementos entre estes.

O método de inserção realiza a comparação e a troca adjacente dos elementos até que encontre o ponto de inserção do elemento que ele pretende inserir. Caso este elemento esteja do outro lado da sequência de elementos, ele irá realizar n-1 comparações e movimentações.

4.1 Utilização

Por ter uma implementação pequena que requer poucas linhas de códigos e por ser um método eficiente, pode ser utilizado em sistemas que não dispõe de muitos recursos de memória.

O seu tempo de execução é sensível à ordem inicial do programa, o que lhe garante um bom uso em sequências já ordenadas. (herdado do método de inserção)

O seu sistema não é estável, pois ele pode realizar troca de elementos de igual valor.

5. Quicksort

Método de divisão e conquista que concentra o trabalho na divisão e faz a conquista imediata, através deste método a sequência é dividida através de um de seus valores que é

considerado mediano, a partir deste valor são ordenados os elementos restantes em seu lado esquerdo, caso menores que este, ou lado direito recursivamente.

Este método seleciona um dos valores da sequência, este numero então é considerado um pivô, e é comparado com os demais valores da sequência ainda não ordenados que são alocados em sua esquerda caso sejam menores e a sua direita caso sejam maiores, disto restam duas sequências uma a direita e outra a esquerda do pivô, em cada uma destas é realizado o mesmo método até que reste apenas um elemento nas sequências, neste instante temos a sequência ordenada.

5.1 Utilização

Este é um sistema de ordenação bem eficiente, na prática ele efetua uma média de n log n comparações.

Para termos uma ordenação boa, a escolha do pivô é a chave, então a escolha pode ser feita através da mediana de três elementos da sequência. Por exemplo, de uma sequência pegamos o primeiro valor, o valor do meio e o último, fazemos uma comparação entre eles e verificamos qual é o valor médio dentre estes três, a partir disso utilizamos este valor médio como pivô.

6. Mergesort

Método de divisão e conquista que concentra o trabalho na conquista, neste modelo de ordenação a sequência é dividida em sequências menores com metade do tamanho da original que são ordenados recursivamente.

Este método divide a sequência em sequências com metade de seu tamanho ordenando-as enquanto divide, e realiza sucessivas divisões de cada uma das sequências conseguidas, ao final destas a sequência se torna ordenada pois foi-se ordenando-a um a um os elementos.

6.1 Utilização

No pior caso de uso, este método é mais eficiente que o quicksort, apesar de que na prática o pior caso não é o que acontece em todas as vezes, tornando assim o quicksort mais eficiente na prática.

7. Heapsort

Método que simula uma árvore binária completa, através deste método utilizamos uma simulação de uma arvore binária completa e utilizamos certas funções especiais deste tipo de dado para realizar a ordenação.

Uma árvore com estrutura heap é aquela em que, para toda subárvore, o nó raiz é um divisor de valores, ou seja, os valores maiores estão sempre de um lado e os menores estão do outro lado. Deste modo o elemento no topo da árvore é sempre o elemento de inicio da sequência (em modo crescente o menor).

7.1 Utilização

Este método tem em seu pior caso de inserção de um dado na árvore em razão de log n, o que seria realizado utilizando percorrer a o tamanho da árvore, não a árvore inteira.

Este é um método bastante eficiente porém requer a construção e ordenação de uma árvore binária, o que o torna mais complexo de ser implementado, e utiliza mais recursos.

8. Counting Sort

Método de ordenação linear, este método requer o tamanho máximo de um valor dentro da sequência para então utilizar uma sequência virtual de contagem que tem este tamanho máximo, fazendo com que todos os valores da sequência original sejam contados e que seja colocado na posição referente a seu valor nesta sequência virtual a quantidade de valores referentes a cada posição.

Então utiliza outra sequência auxiliar com o tamanho da sequência original que irá conter os valores da sequência original ordenando-os através da busca nesta sequência original na quantidade de vezes que contém cada uma das posições da sequência virtual de contagem.

8.1 Utilização

Este método só pode ser utilizado com números inteiros, visto que sua forma de ordenação é através de utilização de uma sequência posicional que irá receber a quantidade de vezes que valor referente a tal posição aparece na sequência.

Para garantir estabilidade ele deve varrer os elementos do lado contrário a sua determinada ordenação, ou seja, para ordenação crescente ele deve varrer do último ao primeiro elemento.

Este é um método a ser implementado quando a comparação entre os elementos da sequência é um problema, ele utiliza operações aritméticas e atribuições para realizar seu trabalho.

No entanto para uma sequência em que o valor máximo é muito grande ele utiliza muitos recursos sem necessidade.

9. Radix Sort

Método de ordenação linear que ordena valores inteiros com um número contante de dígitos.

Este método utiliza-se do sistema de ordenação digito a digito, iniciando-se a partir do digito menos significativo (ou mais a direita), ele cria subconjuntos que contém todos os valores que possuem o digito verificado igual.

A estabilidade da ordenação toma por base a indução de que se um número for ordenado pelo seu i-ésimo dígito sucessivamente até seu primeiro dígito, ele irá ordena-los de forma a permitir que suas posições sejam mantidas. Pois para dois números com i-ésimos dígitos distintos, o de menor valor no dígito estará antes do de maior valor, e se ambos possuem o i-ésimo digito igual, então também estarão ordenados pois seus dígitos menos significativos já foram ordenados anteriormente.

9.1 Utilização

Se você possui uma sequência com valores fixos de quantidade de dígitos dos elementos então este método possui uma complexidade em função apenas deste valor fixo, porém se utilizarmos a quantidade de dígitos variáveis este método pode precisar de sistemas adicionais de verificação da quantidade destes dígitos.

A melhor utilização deste método depende da quantidade de dígitos máximos que possui o valor a ser ordenado, pra uma quantidade pequena de dígitos este é infinitamente o melhor sistema de ordenação. Para facilitar esta escolha, verifique se a quantidade de dígitos é menor que log n (sendo n a quantidade de valores na sequência).

A sua vantagem fica evidente quando interpretamos dígitos de forma mais geral que simplesmente 0 a 9, pois este algoritmo pode ser utilizado também para ordenação de literais ou qualquer outro tipo de dado que possa ser visto como uma d-upla ordenada de ítens comparáveis.

10. Bucket Sort

Método de ordenação linear que supõe que os elementos a serem ordenados possuem uma natureza peculiar.

Para discutirmos este método utilizaremos um exemplo em que os elementos estão uniformemente distribuídos em um intervalo semi-aberto, a idéia é dividir este intervalo em n segmentos de mesmo tamanho (*buckets*) e distribuir os n elementos nos seus respectivos segmentos. Como os elementos estão distribuídos uniformemente, espera-se que o número de elementos seja aproximadamente o mesmo em todos os segmentos.

Em seguida, os elementos de cada segmento são ordenados por um método qualquer e então os segmentos ordenados são concatenados em ordem.

10.1 Utilização

Este método pode ser utilizado em conjunto com outros métodos para realizar a separação de um problema grande em vários problemas pequenos, sendo mais simples de ordenar pelos métodos anteriormente descritos.

CONCLUSÃO

Através da realização deste trabalho podemos concluir que a utilização de um método de ordenação seja ele qual for deve levar em consideração muitas variáveis para que se possa ter um modelo de ordenação único que atenda a todas as necessidades da melhor maneira possível.

Então o que podemos concluir é que não existe um sistema de ordenação que seja melhor ou mais rápido que outro sistema, existem sistemas que analisando um caso único podem ser mais rápidos que outros.

O que se deve perceber então é que a definição do uso da ferramenta e o primeiro passo para a escolha de uma solução.

REFERÊNCIAS

HOROWITZ, E. SAHNI, S. 1987. **Fundamentos de Estruturas de Dados**. Ed. Campus.

SWAIT, J.D. 1991. **Fundamentos Computacionais: Algoritmos e Estrutura de Dados.** Makron Books.

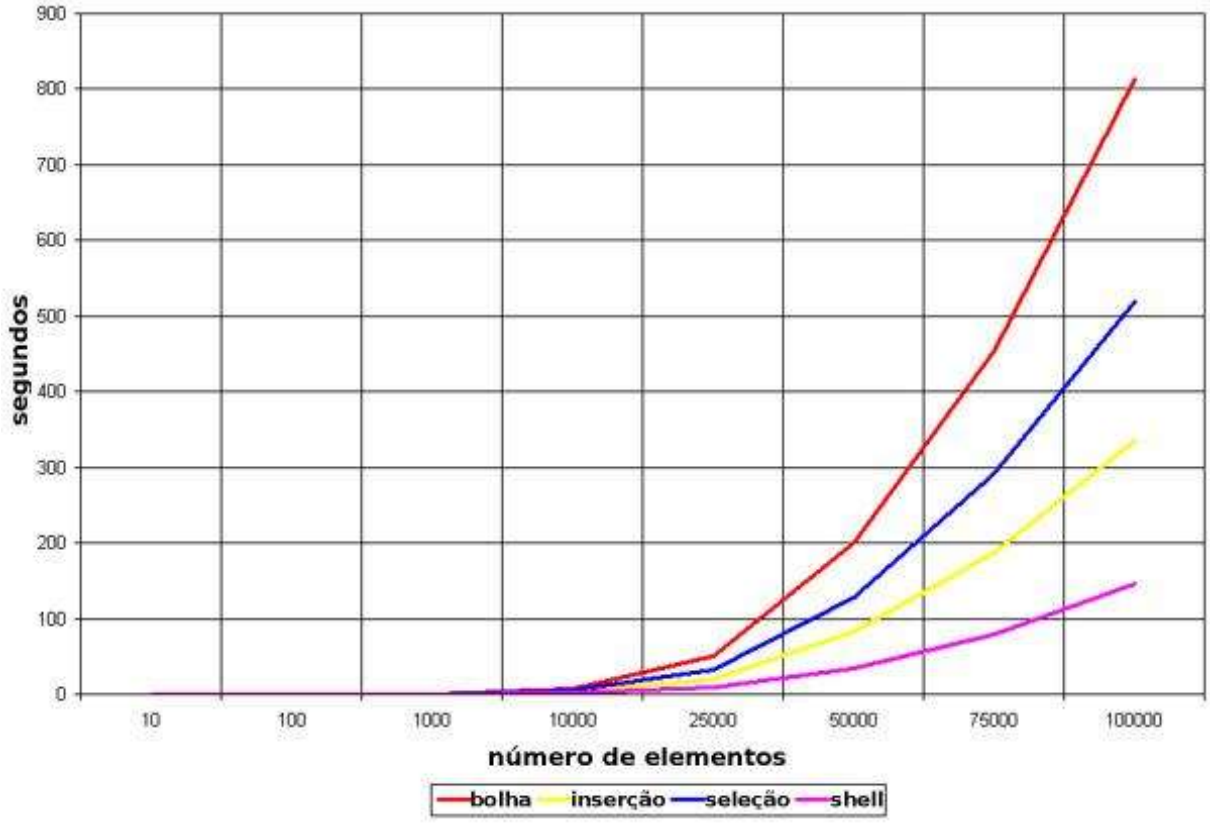
NETO, Peter T. **Ordenação de vetores** [online] disponível em, <http://www.ulbra.tche.br/~danielnm/ed/trab991/g1/t2/peter/ordenacao.htm> capturado no dia 05/11/2003.

FEODFILO, Paulo. **Análise de algoritmos** [online] disponível em, <http://www.ime.usp.br/~pf/mac338-1999/> capturado no dia 07/11/2003.

BRUMMUND, Peter. **The complete collection of Algorithm Animations** [online] disponível em, <http://www.cs.hope.edu/~alganim/ccaa/> capturado no dia 15/11/2003.

APÊNDICE

Apêndice A:



Apêndice B: