ANTONIO MENEGHETI FACULDADE

LUCAS VENDRUSCOLO DALMOLIN

PROFESSOR: FERNANDO LUÍS HERMAN

MÉTODOS DE ORDENAÇÃO

(Insertion Sort, Selection Sort, Quick Sort, Heapsort, Shell Sort)

Restinga Seca - RS

2016

1 – INTRODUÇÃO

Ordenar é um processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente, facilitando a recuperação posterior de itens do conjunto ordenado. Os algoritmos ocupam-se sobre os registros de determinado arquivo, cada registro tem uma chave empregada para controlar a ordenação, e podendo existir outros componentes em um registro.

Um método só pode ser chamado de estável caso a ordem relativa dos itens com chaves iguais não se alterar durante a ordenação, normalmente os métodos mais eficientes são os não estáveis, onde ela é mais forçada a ter estabilidade.

Neste trabalho o objetivo é explicar alguns desses métodos de ordenação para melhorar entendimento do leitor. Será abordado os métodos Insertion Sort, Selection Sort, Quick Sort, Heap Sort e Shell Sort.

2 – INSERTION SORT

O processo de ordenação pode ser terminado caso um item com chave menor que o item em consideração for encontrado ou o final da sequência destino é atingido à esquerda.

O número mínimo de comparações e movimentos ocorre quando os itens estão originalmente em ordem. O número máximo ocorre quando os itens estão originalmente na ordem reversa. É um método utilizado para quando o arquivo está quase ordenado. Esse algoritmo é considerado estável.

```c
for (i=1; i < 100; i++) // esse for seria a lógica do insertion sort
{
    j = i;
    while ((j > 0) && (vetor[j-1] > vetor[j]))
    {
        aux = vetor[j-1]; // aqui começa a troca dos números para a ordem de insertion sort
        vetor[j-1] = vetor[j];
        vetor[j] = aux;
        j--;
    }
}
```

(Figura 1. Algoritmo Insertion Sort em C)

3 – SELECTION SORT

É um dos algoritmos mais simples de ordenação. Tem um custo linear no tamanho da entrada para o número de movimentos de registros, é um algoritmo utilizado para arquivos com registros muito grandes, mas também é interessante em arquivos pequenos. Porém, ele é considerado um algoritmo não estável.

```c
for (i - 0; i < 100 - 1; i++)
{
    min = i;
    for (j = i+1; j < 100; j++)
    {
        if (vetor[j] < vetor[min])
        {
            min = j;
        }
    }
    aux = vetor[i];
    vetor[i] = vetor[min];
    vetor[min] = aux;
}
```

(Figura 2. Algoritmo Selection Sort em C)

Conforme a lógica do Selection Sort ilustrado na Figura 1, o Algoritmo faz:

1. Seleciona o menor item do vetor.
2. Troca com o item da primeira posição do vetor.
3. Repete essas duas operações até chegar a 100 – 1, até que reste apenas um elemento.

4 – QUICK SORT

É o algoritmo mais eficiente que se conhece para uma grande variedade de situações, e também o mais utilizado. O Quick Sort é recursivo, o que significa que ele demanda uma pequena quantidade de memória adicional. Sua ideia básica é dividir o problema de ordenar em dois problemas menores, que serão ordenados independentemente, sendo que os resultados são combinados para produzir a solução final.

É um algoritmo extremamente eficiente para ordenar arquivos de dados, necessita apenas de uma pequena pilha como memória auxiliar. Porém, sua implementação é muito delicada e difícil, já que um pequeno engano levar a efeitos inesperados para algumas entradas de dados. O método Quick Sort é considerado não estável.

```c
void quick_sort(int *a, int left, int right) {
    int i, j, x, y;

    i = left;
    j = right;
    x = a[(left + right) / 2];

    while(i <= j) {
        while(a[i] < x && i < right) {
            i++;
        }
        while(a[j] > x && j > left) {
            j--;
        }
        if(i <= j) {
            y = a[i];
            a[i] = a[j];
            a[j] = y;
            i++;
            j--;
        }
    }
    if(j > left) {
        quick_sort(a, left, j);
    }
    if(i < right) {
        quick_sort(a, i, right);
    }
}
```

(Figura 3. Função quick sort em C)

5 – HEAP SORT

Algoritmo que tem os mesmos princípios de funcionamento do Selection Sort. É um método elegante e eficiente, e muito utilizado por aplicações que não podem tolerar eventuais variações no tempo de execução esperado.

Usa Heaps, que é uma sequência de itens com chaves, onde a chave de cada nó é maior do que as chaves em seus filhos e a chave no nó raiz é a maior chave do conjunto. Sua representação é muito compacta e permite caminhar pelos nós facilmente. O Algoritmo não utiliza nenhuma memória auxiliar

O comportamento do Heap Sort é sempre o mesmo, independentemente de qual seja a entrada, porém seu anel interno é bastante complexo e é considerado um algoritmo não estável.

É recomendado para aplicações que não podem tolerar eventualmente um caso desfavorável e não é recomendado para arquivos com poucos registros, por causa do tempo necessário para construir um Heap.

```
void constroiHeap( int *p_vetor, int tamanho, int indice_raiz )
{
    int ramificacao, valor;
    valor = p_vetor[indice_raiz];

    while(indice_raiz <= tamanho/2){
    ramificacao = 2 * indice_raiz;

    if(ramificacao < tamanho && p_vetor[ramificacao] < p_vetor[ramificacao + 1])
    ramificacao++;

    if(valor >= p_vetor[ramificacao])//Identifica o max-heap
    break;

    p_vetor[indice_raiz] = p_vetor[ramificacao];
    indice_raiz = ramificacao;
    }
    p_vetor[indice_raiz] = valor;
}
        int* p_vetor - Trabalho Topicos.cpp (7) - Ctrl+Click for more info
void HeapSort( int *p_vetor, int tamanho )
{
    int indice, troca;
    for( indice = tamanho/2; indice >= 0; indice-- )
    constroiHeap( p_vetor, tamanho, indice );

    while( tamanho > 0 )
    {
        troca = p_vetor[ 0 ];
        p_vetor[ 0 ] = p_vetor[ tamanho ];
        p_vetor[ tamanho ] = troca;
        constroiHeap( p_vetor, --tamanho, 0 );
    }
}
```

(Figura 4. Na primeira função, ele está construindo os Heaps e na segunda é a função Heap Sort, código em C.)

6 – SHELL SORT

Criado em 1959 por Shell, é uma extensão do algoritmo de Inserção. O problema do Insertion Sort é que ele troca itens adjacentes para determinar o ponto de inserção, já o método de Shell contorna este problema permitindo trocas de registros distantes um do outro. Sua implementação não utiliza registros sentinelas. É um algoritmo não muito conhecido, ninguém sabe da sua eficiência, pois contém alguns problemas matemáticos muito difíceis, como por exemplo, a sequência de incrementos, a única coisa que se sabe é que cada incremento não deve ser múltiplo do anterior.

O Shell é uma opção quando utilizado para arquivos de tamanho moderado, e tem uma implementação simples de poucas linhas de códigos. Porém, o seu tempo de execução é sensível à ordem inicial do arquivo. É considerado um método não estável.

```
void shellSort(int *p_vetor, int tamanho)
{
   int i = (tamanho - 1) / 2;
   int chave, k, aux;

   while(i != 0)
   {
      do
      {
         chave = 1;
         for(k = 0; k < 100 - i; ++k)
         {
            if(p_vetor[k] > p_vetor[k + i])
            {
               aux = p_vetor[k];
               p_vetor[k] = p_vetor[k + i];
               p_vetor[k + i] = aux;
               chave = 0;
            }
         }
      }while(chave == 0);
      i = i / 2;
   }
}
```

(Figura 5. Shell Sort em C)

7 – CONSIDERAÇÕES FINAIS

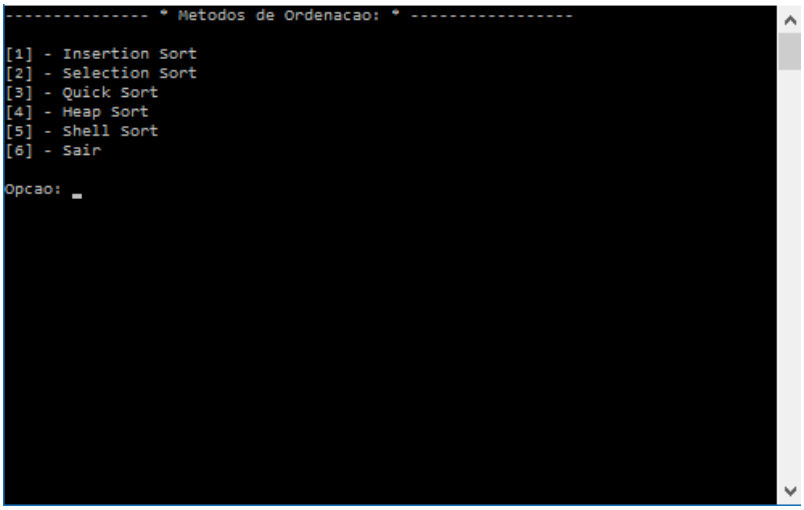
Neste trabalho foi abordado o assunto Métodos de Ordenação, que são algoritmos que colocam os elementos de uma dada sequência em uma certa ordem. Existem diversos algoritmos e neste trabalho foram estudados apenas o Insertion Sort, Selection Sort, Quick Sort, Heap Sort e Shell Sort.

Entre as diferenças entre eles chegamos a algumas conclusões:

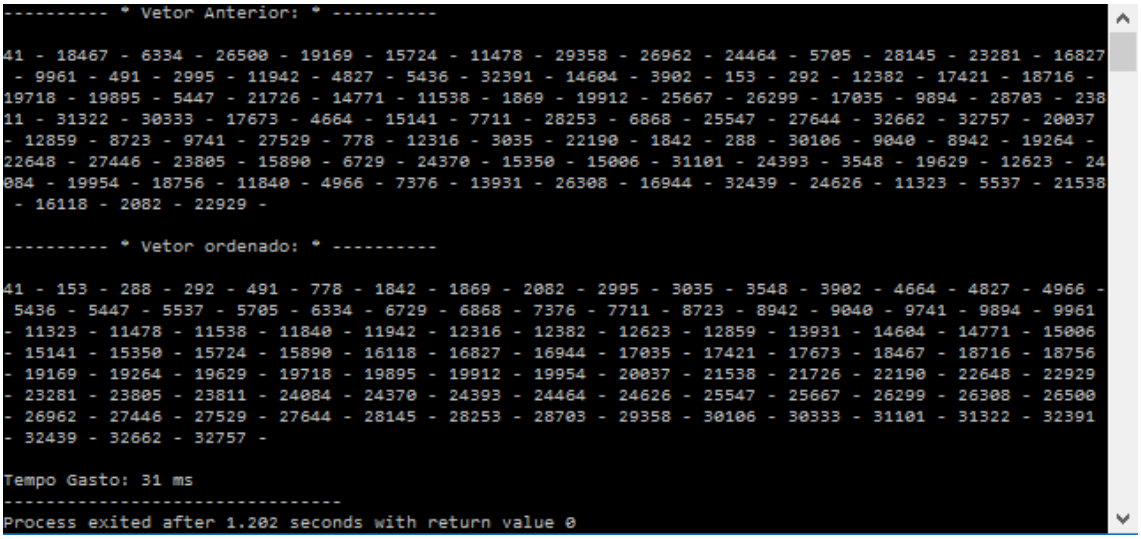
1. Quick Sort é o mais rápido para todos os tamanhos aleatórios.
2. Para arquivos pequenos, o Shell Sort é mais rápido que o Heap Sort, mas quando o tamanho de entrada cresce, o Heap Sort fica mais rápido.
3. O Insertion Sort é o mais rápido quando os elementos já estão ordenados, mas se estiverem em ordem descendente é o mais lento.

4. Heap Sort não é sensível a entrada de dados, mantém o mesmo comportamento. Já o Shell Sort é bastante sensível.
5. Insertion Sort é mais interessante para arquivos com menos de 20 elementos.
6. O Selection Sort deve ser usado para arquivos com registros grandes, mas que não passe de 1000 elementos.

8 – UTILIZAÇÃO DO MEU PROGRAMA



1. O usuário tem a opção de digitar de 1 a 5 para escolher qual método de ordenação ele deseja utilizar. Caso ele digite 6, o programa fecha e caso ele digite qualquer outra letra ou número, ele pedirá para tentar novamente.



2. Depois de selecionado o método, o programa irá criar automaticamente um vetor com 100 elementos que será mostrado no “Vetor Anterior”, logo abaixo no “Vetor Ordenado”, será mostrado os 100 números do vetor já ordenados referentes ao método selecionado.
3. Em tempo gasto, é o tempo de execução da ordenação em milissegundos, neste caso por exemplo, o programa demorou 31 milissegundos para executar a ordenação.

9 – REFERÊNCIAS BIBLIOGRÁFICAS

[1] DCC, **Ordenação**. Disponível em: <http://www2.dcc.ufmg.br/disciplinas/aeds2_turmaA1/cap4.pdf>. Acesso em: 14 de abril de 2016.

[2] DCC, **Comparação entre os métodos de ordenação.** Disponível em: <http://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/sortingcmp.pdf>. Acesso em: 14 de abril de 201