Trabalho: Algoritmos de Busca e Ordenação

Prof. Bruno Emerson Gurgel Gomes
IFRN - Câmpus Currais Novos

31 de outubro de 2012

1 Introdução

Os algoritmos de busca e de ordenação compreendem um conjunto de algoritmos de elevada importância para a computação. A busca de informações pode ser encontrada em uma planilha de cálculos ou editor de texto, em um sítio da Internet ou ainda em um programa aplicativo que faz uma consulta a um banco de dados. Em todos esses casos, há um algoritmo que é empregado para buscar o termo (um nome de pessoa, um CPF, um valor em R$, etc.) que foi requisitado pelo usuário. De outro modo, é possível empregar algoritmos para ordenar elementos tendo em vista a sua apresentação ou facilitar a busca de um valor.

Neste trabalho, deve-se implementar dois algoritmos simples de busca e outros dois de ordenação sobre elementos armazenados em um vetor. Ao término do trabalho, você deverá estar melhor familiarizado com a linguagem C++, com o uso de funções, e com a passagem de vetores a funções.

1.1 Objetivos

- Compreender e implementar alguns algoritmos de busca e de ordenação em C++;
- Treinar a programação em C++ através da codificação dos algoritmos;
- Aprender a passagem de vetores a funções;
- Definir funções sobrecarregadas. Ou seja, funções de mesmo nome, mas com quantidade e/ou tipo de parâmetros diferentes.

2 Instruções gerais quanto à implementação

Para cada algoritmo abaixo, você deve:

1. Compreender o funcionamento do algoritmo. Caso tenha dúvidas, faça um exemplo de execução do algoritmo "no papel", usando para isso um vetor de tamanho pequeno, apenas para entender como ele opera sobre o vetor para realizar o seu propósito;

2. Fazer a implementação do algoritmo diretamente na função *main*. Quando você já estiver familiarizado, já é possível implementar direto como uma função independente (passo 3);

3. Transpor a sua implementação para uma função de forma independente da função *main*. Ou seja, em *main*, você vai apenas requisitar as informações que a função precisa, chamar a função passando para ela essas informações em seus parâmetros e, com base no que a função retornar, comunicar ao usuário do programa os resultados de saída (com *cout*);

4. Para cada algoritmo você deve fazer duas versões, uma com números e outra com textos (vetores de caracteres ou *strings*). As funções podem ter o mesmo nome. Uma vez que um dos parâmetros vai ser diferente quanto ao tipo (o vetor), quando função for chamada, a diferenciação será feita automaticamente através do tipo de vetor que o usuário passar (*float* ou *char**). Esse processo é conhecido como sobrecarga de função. Tome-se como exemplo o algoritmo de busca linear. As assinaturas das duas funções seriam, por exemplo:

bool *buscalin* (**float** v[], **int** tam, **int** termo);

Para a versão com números. Onde "v" é o vetor no qual será feita a busca, "tam" é o tamanho do vetor e "termo" é o termo (número, no caso) que se quer buscar.

bool *buscalin* (**char*** v, **int** termo);

Para a versão vetor de caracteres. Neste caso, passar o tamanho do vetor é opcional, pois lembre-se que o fim de um vetor de caracteres é determinado por '\0'. No caso dos algoritmos de ordenação, é melhor deixar o algoritmo internamente inalterado, passado o tamanho do vetor da mesma forma que o vetor numérico.

3 Algoritmos de busca

3.1 Busca Linear

Pesquisa um elemento em um vetor qualquer (ordenado ou não) até que ele seja encontrado ou até que o fim do vetor seja atingido.

```
Real : x[100], num
Inteiro : n, i
Lógico : achou
Início
    Repita
        Escreva(``Quantos números?'')
        Leia( n )
    Até( n <= 0 .e. n > 100)

    Escreva(``Digite todos os números:'')
    Para i de 0 até n-1 passo 1 faça
        Leia( x[i] )
    Fim_para

    Escreva(``Digite o número que procura:'')
    Leia (num)
    achou = .F.
    i = 0
    Enquanto ( i < n .E. .Não. achou )faça
        Se( x[i] == num) então
            achou = .V.
        senão
            i  =  i+1
        Fim_se
    Fim_enquanto

    Se( achou ) então
        Escreva(``Número encontrado. '')
    Senão
        Escreva(``Número não encontrado.'')
    Fim\_se
Fim
```

3.2 Busca Binária

Pesquisa um elemento em um vetor ordenado. Neste caso, inicialmente, forneça um vetor já ordenado. Quando as funções de ordenação da seção 4 já estiverem prontas, você deve utilizar uma delas para ordenar o vetor antes de iniciar a busca.

```
Real: x[100], num
```

```
Inteiro: n, i, meio, alto, baixo
Lógico: achou
Início
    Repita
        Escreva(``Quantos números?'')
        Leia( n )
    Até ( n <=  0 .e. n > 100 )
    Escreva(``Digite todos os números:'')

    Para i de 0 até n-1 passo 1 faça
        Leia ( x[i] )
    Fim_para

    Escreva (``Digite o número que procura: '')
    Leia( num )
    alto = n - 1
    baixo = 0
    achou = .F.

    Enquanto( baixo <= alto .e. (.não. achou) ) faça
        meio = (baixo + alto) / 2
        Se( num < x[meio] )então
            alto =  meio - 1
        senão
            Se( num > x[meio] ) então
                baixo =  meio + 1
            senão
                achou = .V.
            Fim_se
        Fim_se
    Fim_enquanto

    Se( achou ) então
        Escreva(``Número encontrado.'')
    Senão
        Escreva(``Número não encontrado.'')
    Fim_se
Fim
```

4 Algoritmos de ordenação

Os dois algoritmos de ordenação a seguir são lentos e realizam muitas operações para o caso em que um vetor está totalmente desordenado. No entanto, eles foram selecionados devido a sua implementação simples. As ordenações foram feitas na ordem crescente.

4.1 Bolha (*bubble sort*)

```
Real: x[100], aux
Inteiro: n, i, j
Início
    Repita
    Escreva(``Quantos números?'')
    Leia( n )
    Até( n <= 0 .e. n > 100 )
    Escreva(``Digite os números:'')

    Para i de 0 até n-1 passo 1 faça
        Leia ( x[i] )
    Fim_para

    Para i de 0 até n-1 passo 1 faça
        Para j de 0 até j < i passo 1 faça
            Se( x[i] < x[j] ) então
                aux = x[i]
                x[i] = x[j]
                x[j] = aux
            Fim_se
        Fim_para
    Fim_para

    Escreva(``Vetor ordenado: '')
    Para i de 1 até n faça
        Escreva ( x[i] )
    Fim_para
    Fim
```

4.2 Inserção (*Insertion sort*)

```
var
   Inteiro : v : [100]
   Inteiro : j,k,i,temp
inicio
   leia (tam) //<= 100
   para j de 0 ate tam -1 faca
      leia(v[j])
   fimpara
```

```
    para j de 0 ate tam - 2 faca
        k <- j
        para i de (j+1) ate tam - 1 faca
            se v[i] < v[k] entao
                k <- i
            fimse
        fimpara

        se k > j entao
            temp <- v[j]
            v[j] <- v[k]
            v[k] <- temp
        fimse
    fimpara

    para j de 0 ate tam -1 faca
        escreval(v[j])
    fimpara
fimalgoritmo
```