Lógica de Programação

BUSCA

Evidentemente, possuir os dados não ajuda o programador ou o usuário se eles não souberem onde os dados estão. Imagine, por exemplo, uma festa de casamento com cem convidados na qual não se sabe quem eles são ou se determinada pessoa foi ou não convidada. Imagine, nas eleições, que você queira votar naquele único político honesto que conhece, mas não sabe qual é seu número!

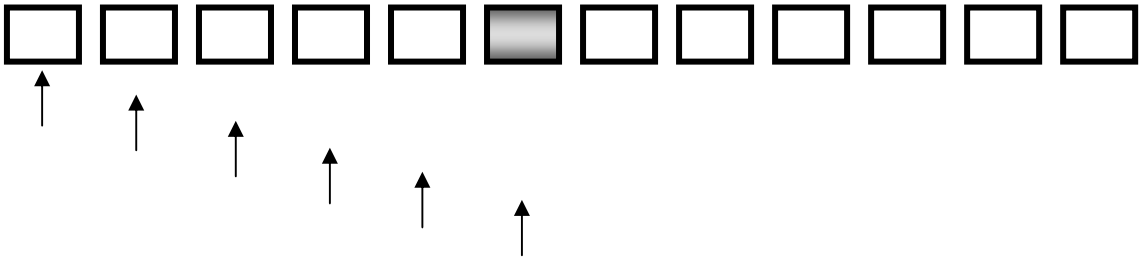
Os algoritmos de busca são alguns dos mais utilizados no mundo da informática, sendo usados em bancos de dados, internet, jogos, entre outros. Aqui serão apresentados alguns exemplos de algoritmos de **busca linear** e **busca binária**.

A escolha do método a ser utilizado para busca depende muito da quantidade de dados envolvidos, do volume de operações de inclusão e exclusão a serem realizadas, entre outros fatores que devem ser considerados quando do desenvolvimento da aplicação.

BUSCA LINEAR (OU SEQÜENCIAL)

A maneira mais óbvia de fazer uma busca é, com certeza, comparar o elemento que se está procurando com todos os elementos guardados, um a um, isto é, o mais simples é procurar o elemento seqüencialmente até que seja encontrado.

O algoritmo que realiza essa busca é realmente muito simples e consiste em uma estrutura de repetição que varre toda a seqüência de elementos, realizando uma condição que compara o elemento desejado com os elementos existentes na seqüência.



| Figura 1 | Busca Linear

Quando o elemento é encontrado, retorna-se o valor *verdadeiro*, o que indica o sucesso da busca e encerra a estrutura de repetição. É claro que a maneira de encerrar essa estrutura dependerá da linguagem de programação a ser utilizada.

Lembre-se: a execução da rotina de busca termina quando a condição de busca é satisfeita, ou então quando todo o conjunto é percorrido e o elemento não foi encontrado.

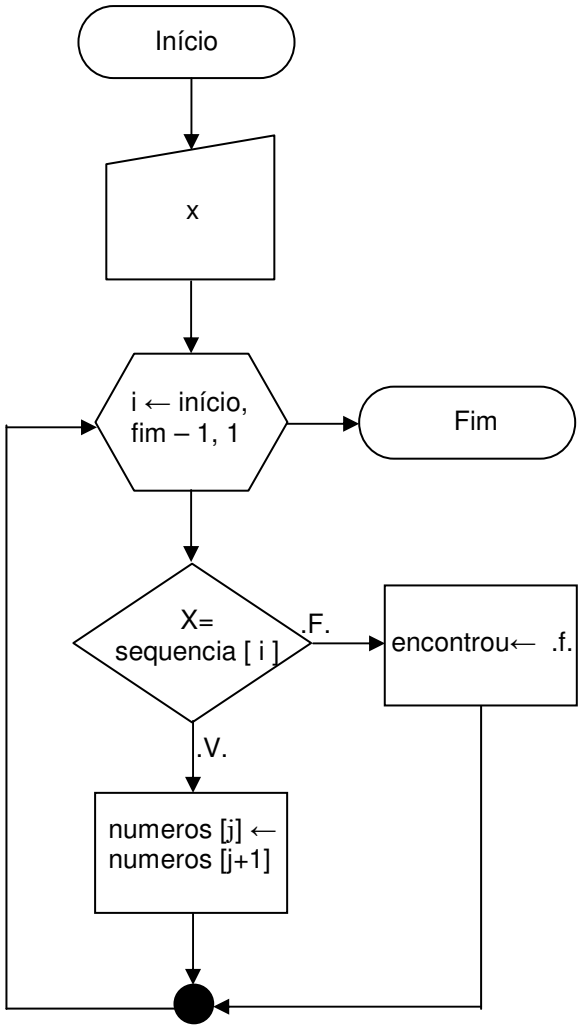
Lógica de Programação

EXEMPLO 1: ALGORITMO DE BUSCA LINEAR (SEQUENCIAL)

```
1.  Algoritmo Exemplo1
2.  Var
3.        sequencia: vetor de elementos
4.        x          : elemento a ser procurado
5.        encontrou: logica
6.        i          : inteiro
7.  Início
8.       Ler (x)
9.       Para i de <inicio> até <fim – 1 > Faça
10.            Se (x = sequencia[i])
11.                 encontrou ← .v.
12.            <fim da busca>
13.            Senão
14.                 encontrou ← .f.
15.            Fim-se
16.      Fim-Para
17. Fim.
```

No algoritmo EXEMPLO 1, para a variável *sequencia*, devem-se declarar o tamanho do vetor e o tipo de dados que esse vetor receberá, assim como para a variável x deve-se declarar o tipo de dado da informação a ser procurada.

FLUXOGRAMA:



Uma variável bastante comum dessa e das demais estruturas de busca retorna o índice do elemento procurado na seqüência. Para isso, basta substituir a variável lógica por uma variável de tipo inteiro que receberá o índice do elemento, quando ele for encontrado, e poderá receber um valor não associado a nenhum índice, por exemplo, -1. A variação escolhida sempre será uma opção do programador que refletirá suas necessidades.

BUSCA BINÁRIA (OU BUSCA LOGARÍTMICA)

O método de busca linear é o mais adequado quando não se tem nenhuma informação a respeito da estrutura em que a busca será realizada. Por outro lado, se o elemento procurado estiver entre os últimos ou não estiver no conjunto, esse método poderá ser demasiadamente lento, pois ocorrerão comparações com todos os elementos de um conjunto que pode ser muito grande – imagine a relação dos clientes de um banco!

Quando temos uma seqüência ordenada de elementos, existem outros métodos de busca que são muito mais adequados, pois permitem realizar uma busca por meio de algoritmos mais eficientes, que podem utilizar um número menor de comparações.

Considere uma seqüência ordenada de elementos de qualquer tipo. Em vez de se comparar o elemento procurado ao primeiro elemento da seqüência, pode-se compara-lo a um elemento do meio da seqüência. Se o elemento comparado é o desejado, a busca termina; senão, pode-se verificar se o elemento desejado é maior ou menor que o elemento encontrado. Como todos os elementos estão ordenados, essa verificação elimina a metade da seqüência onde o elemento não pode estar. Por exemplo, se o elemento procurado for maior que o elemento encontrado, a metade inferior da lista poderá ser descartada para a próxima comparação.

A segunda comparação será feita com o elemento do meio da seqüência que restou e o procedimento anterior se repetirá. Dessa forma, cada vez que o elemento não for encontrado, o conjunto a ser pesquisado será reduzido pela metade, aproximadamente, até que o elemento seja encontrado ou até que a lista não possa mais ser dividida.

Esse método foi batizado de método de busca algorítmica pelo fato de haver uma redução algorítmica de elementos a serem pesquisados, mas, como ocorre essa redução pela metade dos elementos da busca em cada comparação, esse método é conhecido como método de **busca binária** ou **busca logarítmica**.

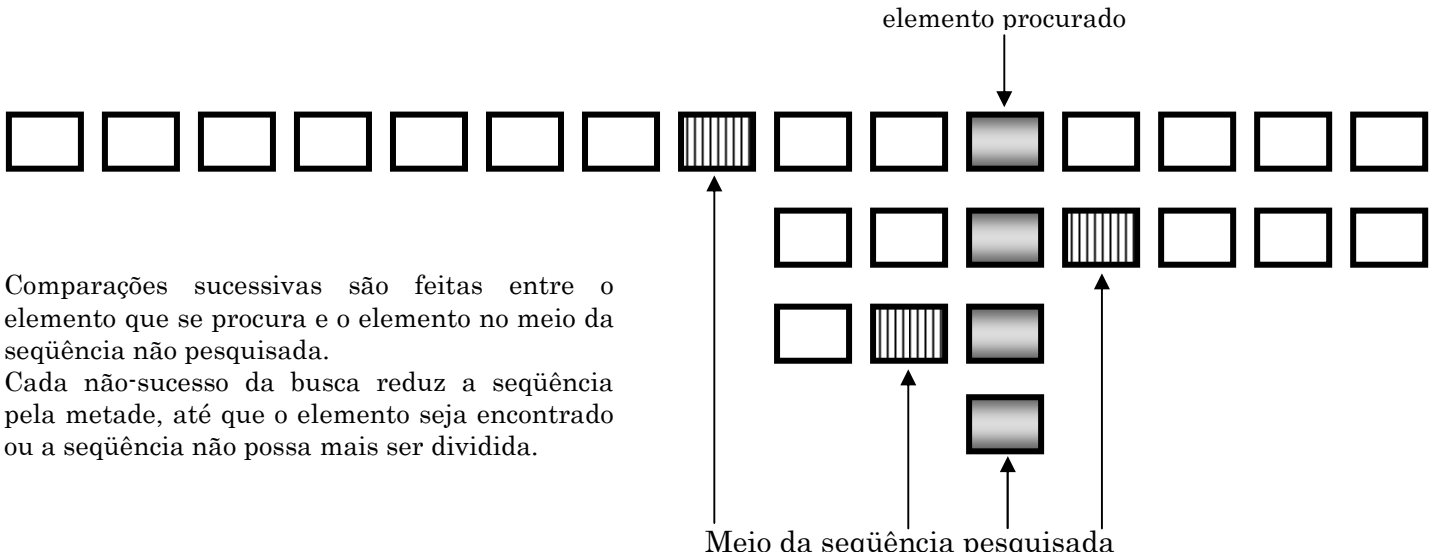
Para um conjunto de 15 elementos, o método de busca linear realizaria, no mínimo, uma comparação e, no máximo, 15 comparações. No caso da busca binária, no mínimo ocorreria uma comparação e, no máximo, quatro comparações, o que geraria um ganho considerável, mais sensível para conjuntos de elementos muito grandes. Considere o exemplo abaixo, que demonstra isso.

Suponha que o elemento a ser localizado seja o 329, que será chamado de *x*:

500	178	2	487	158	47	35	78	329	215	19	25	214	38	77

1º passo – Ordenar o conjunto:

2	19	25	35	38	47	77	78	158	178	214	215	329	487	500



| Figura 2 | *Busca binária*

2º Passo – Dividir o conjunto ao meio:

2	19	25	35	38	47	77	78	158	178	214	215	329	487	500

3º Passo – Efetuar a comparação para verificar se o elemento procurado é igual ao elemento central, que será chamado de *meio*:

x = meio

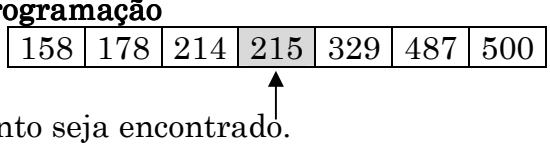
Se o resultado for *verdadeiro*, a busca deverá ser encerrada. Caso contrário, serão executados os passos seguintes. No caso do nosso exemplo:

329 = 78, a resposta é *falso*.

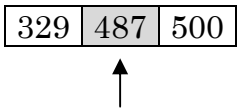
4º Passo – efetuar a comparação para verificar se o elemento procurado é maior ou menor do que o elemento central:

329 >= 78, a resposta é *verdadeiro*.

5º Passo – Proceder a uma nova divisão do conjunto que atenda à condição do 4º passo, isto é, se *x* for maior que *meio*, será dividido o conjunto da direita, senão, o conjunto da esquerda. No caso do exemplo, será utilizado o conjunto da direita.



Repetir os passos 4 e 5 até que o elemento seja encontrado.



EXEMPLO 1: BUSCA BINÁRIA (OU LOGARÍTMICA)

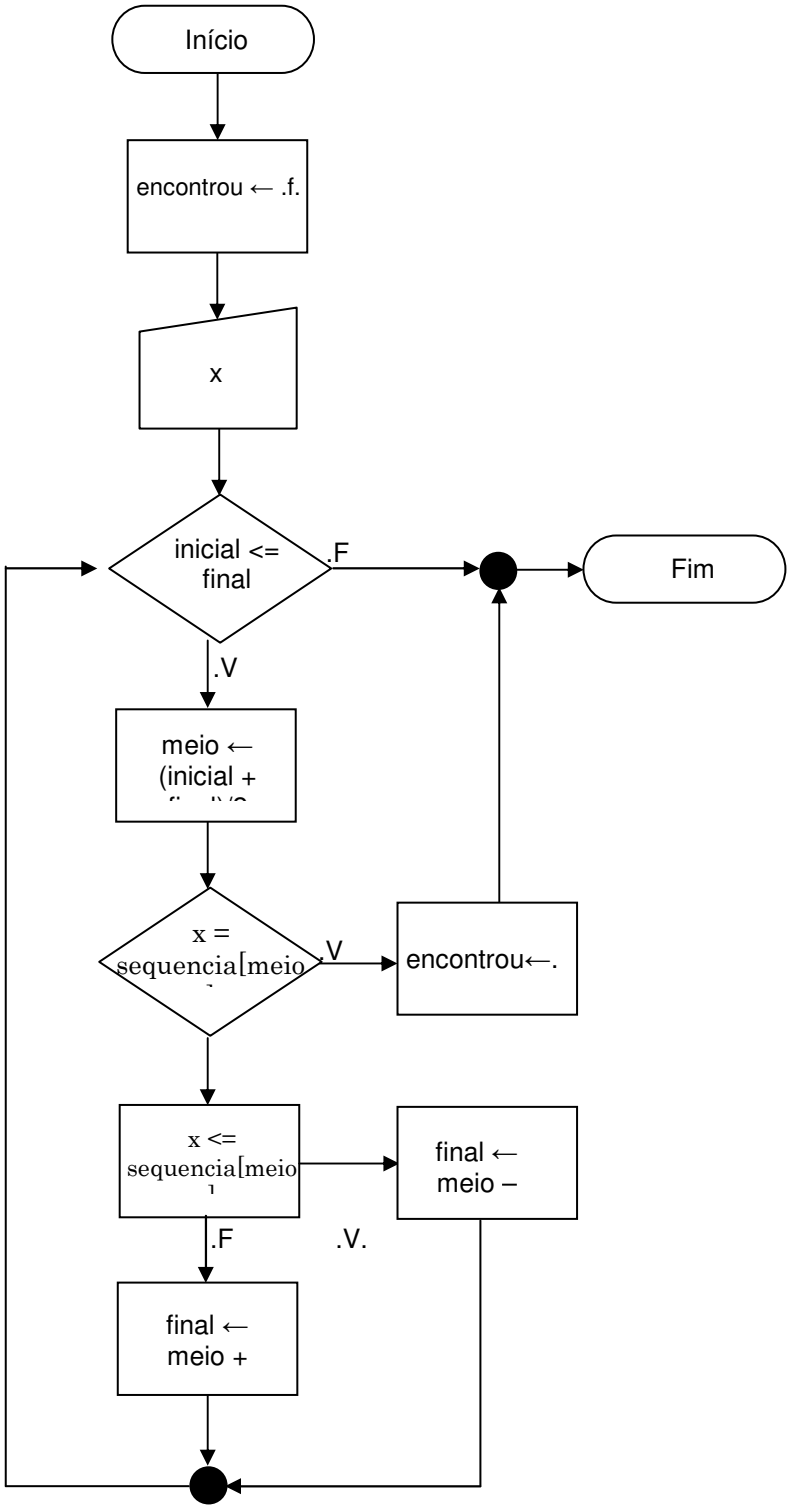
```
1.  Algoritmo Exemplo8.8
2.  Var
3.      sequencia : vetor de elementos
4.      x         : inteiro
5.      encontrou : logica
6.      inicial, meio, final : números inteiros
7.  Inicio
8.      encontrou ← .f.
9.      Ler ( x )
10.     Enquanto (inicial < = final) Faça
11.         meio ←  (inicial + final) / 2
12.         Se (x = seqüência [meio])
13.             encontrou ← .v.
14.             <fim da busca>
15.         Senão
16.             Se (x <=  seqüência [meio])
17.                 final ← meio – 1
18.             Senão
19.                 inicial ← meio + 1
20.             Fim-Se
21.         Fim-Se
22.     Fim-Enquanto
23. Fim.
```

O algoritmo de busca binária (Exemplo 8.8) é, na verdade, bem simples. Considerando uma seqüência qualquer, o índice do meio da seqüência é dado pela divisão por 2 da soma do primeiro com o último índice do conjunto que se deseja pesquisar (linha 11) – cuidado para garantir o resultado inteiro dessa divisão, uma vez que índices só podem ser números inteiros. Compara-se o elemento procurado ao elemento do meio da seqüência (linha 12) e,se forem iguais, a busca é encerrada (linha 14). Caso contrário, uma segunda comparação é realizada para determinar se o elemento procurado é maior ou menor que o elemento encontrado no meio (linha16). Se o elemento é menor, significa que a segunda metade da seqüência pode ser ignorada para a próxima pesquisa, e isso pode se alcançado alterando-se o valor do último índice do conjunto que se deseja pesquisar para o valor inferior do meio (linha17). Caso contrário, isto é, caso o elemento seja maior que o elemento encontrado, então a primeira parte da seqüência pode ser ignorada, alterando-se o índice do início da busca para o valor imediatamente superior ao do meio (linha 19).

Isso será repetido até que o elemento seja encontrado ou até que a seqüência não possa mais ser dividida, o que ocorre, na prática, quando o índice do fim

torna-se menor que o índice do início. Nesse caso, o elemento não se encontra na seqüência e retorna-se *falso*.

FLUXOGRAMA:



Lembre-se: Para implementar estes algoritmos, as variáveis inicial *e* final *devem receber algum valor!*