Algoritmos
de busca

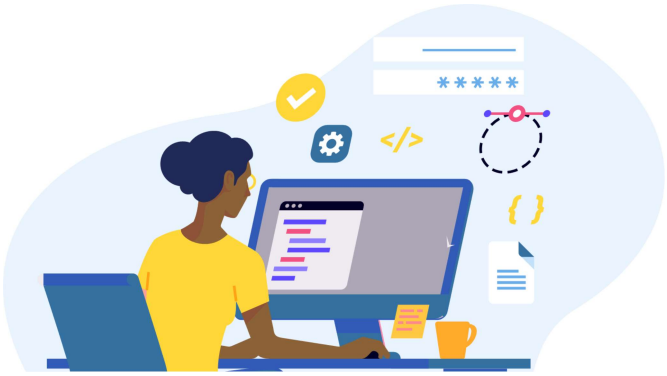
Você sabia que seu material didático é interativo e multimídia? Isso significa que você pode interagir com o conteúdo de diversas formas, a qualquer hora e lugar. Na versão impressa, porém, alguns conteúdos interativos ficam desabilitados. Por essa razão, fique atento: sempre que possível, opte pela versão digital. Bons estudos!

Os algoritmos computacionais são desenvolvidos e usados para resolver os mais diversos problemas. Algoritmos de busca, nesse universo, como o nome sugere, resolvem problemas relacionados ao encontro de valores em uma estrutura de dados.

Diversas são as aplicações que utilizam esse mecanismo.

Você já fez alguma pesquisa no Google hoje? Já utilizou seu app do banco? Já procurou alguém nas redes sociais?

Todas essas aplicações utilizam mecanismos de busca. Um grande diferencial entre uma ferramenta e outra é a velocidade da resposta: quanto mais rápido ela for, mais gostamos dela.



Fonte: Shutterstock.

Por trás de qualquer tipo de pesquisa existe um algoritmo de busca implementado.

Nesta Webaula vamos aprender sobre os algoritmos computacionais.

Busca linear ou sequencial

O algoritmo de busca sequencial, ou busca exaustiva, percorre os itens da sequência procurando o valor de destino, conforme ilustra a figura.

Busca sequencial

Fonte: elaborada pela autora.

A busca começa por uma das extremidades da sequência e vai percorrendo-a até encontrar (ou não) o valor desejado. Com essa imagem fica claro que uma pesquisa linear examina todos os itens da sequência até encontrar o item de destino, o que pode ser muito custoso computacionalmente.

Código	Explicação
def executar_busca_linear(lista, valor):	Definição da função
tamanho_lista = len(lista)	Definição do tamanho da lista
tamanho_lista = len(lista)	Estrutura de repetição para percorrer toda a lista
if valor == lista[i]:	Condição para verificar se é o valor.
return True	Retorno, caso encontre o valor
return False	Retorno, caso não encontre o valor

Fonte: elaborado pela autora.

Algoritmos de busca sequencial

Para procurar elementos em uma lista por meio de algoritmo de busca sequencial, é necessário percorrer todos os elementos da lista até encontrar o elemento procurado. Para isso, é realizada uma comparação do valor do elemento que se deseja encontrar na lista com o valor de cada posição na lista.

Fonte: Shutterstock.

Saiba mais

Mesmo quando um dos elementos não estiver na lista, todos eles devem ser visitados. Os elementos da lista podem já estar dispostos em uma sequência ordenada, ou não. Para cada um dos casos, haverá um comportamento diferenciado em termos de tempo de execução do algoritmo.

No exemplo de **algoritmo de busca sequencial** que considera um vetor de entrada com números desordenados. Temos uma estrutura de repetição "**while**".

```
1   def busca_sequencial(lista, elemento):
2        pos = 0
3        encontrado = False
4
5        while pos < len(lista) and not encontrado:
6             if lista[pos] == elemento:
7                  encontrado = True
8             else:
9                 pos = pos+1
10
11       return encontrado
12
13   testelista = [2, 10, 8, 15, 18, 20, 12, 1]
14   print(busca_sequencial(testelista, 5))
15   print(busca_sequencial(testelista, 15))
```

Fonte: elaborada pela autora.

Analisando a estrutura de busca sequencial

Linha 5 do código

A estrutura de repetição "while" permitirá percorrer a lista e comparar o elemento procurado com o elemento que está na posição da lista, nesse caso, enquanto a posição for menor que o tamanho da lista e o elemento não for encontrado.

Linha 6 do código

Na linha 6 do código, a estrutura condicional "if", "else" traz a condição para encontrar ou não o elemento do vetor – no caso, se a posição da lista corresponde ao elemento procurado (valor), será exibido True (verdadeiro), se não corresponder, haverá um incremento e o próximo elemento será visitado na sequência da lista.

Linha 13 do código

Na linha 13 é mostrado um vetor com 8 elementos (0 a 7 elementos). E, na linha 14, é buscado o elemento 5 na lista. Como o elemento não está na lista, observe que todas posições serão percorridas, razão pela qual todos os elementos da lista serão visitados.

Linha 14 do código

Na linha 14, temos um teste em que se busca o elemento de valor 15. Nesse caso, quatro elementos da lista serão percorridos até que se encontre o valor 15.

Para testar o comportamento desse algoritmo de **busca sequencial** vamos acessar o código com o Python Tutor.

Busca sequencial em um vetor ordenado

E, se a lista já estiver ordenada, como é realizada a busca do elemento procurado?

Muito bem já conhecemos como se processa uma busca sequencial que considera um vetor de entrada com números desordenados. Mas e, se a lista já estiver **ordenada**, como é realizada a busca do elemento procurado?

Observe a seguinte estrutura.

```
def busca_sequencial_ordenada(lista, elemento):
  pos = 0
  encontrado = False
  para = False
  while pos < len(lista) and not encontrado and not para:
           if lista[pos] == elemento:
                encontrado = True
           else:
               if lista[pos] > elemento:
                   para = True
               else:
                   pos = pos+1

 return encontrado

 testelista = [1, 2, 8, 10, 13, 15, 18, 20]
 print(busca_sequencial_ordenada(testelista, 5))
 print(busca_sequencial_ordenada(testelista, 15))
```

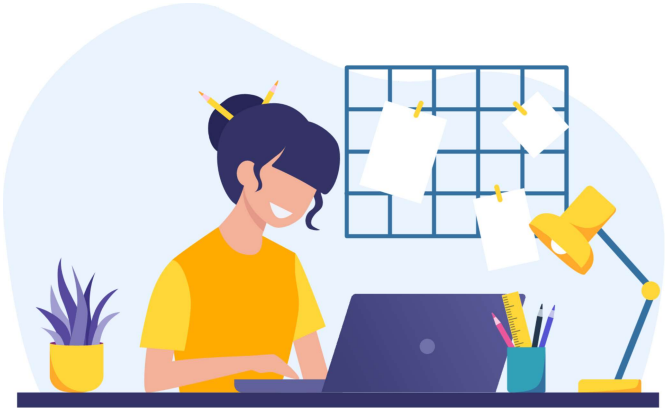
Fonte: elaborada pela autora.

Para testar o comportamento desse algoritmo vamos acessar o código com o Python Tutor.

Algoritmo de busca binária

Outro algoritmo usado para buscar um valor em uma sequência é o de busca binária.

A primeira grande diferença entre o algoritmo de busca linear e o algoritmo de busca binária é que neste os valores precisam estar ordenados.



Fonte: Shutterstock.

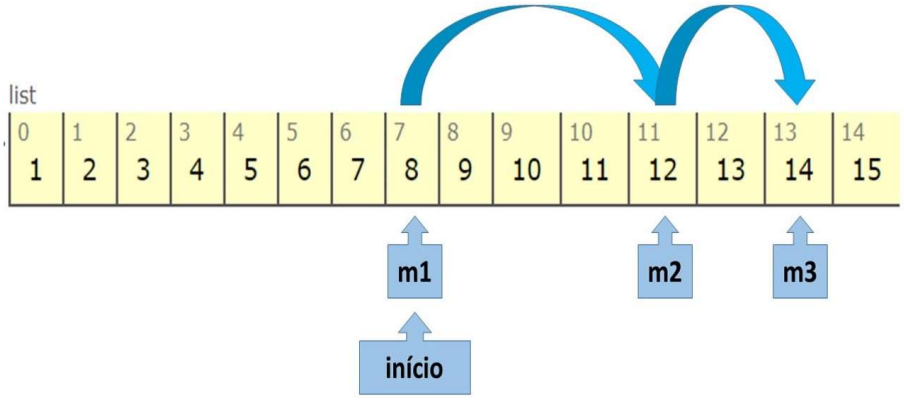
A lógica é a seguinte:

1. Encontra o item no meio da sequência.
2. Se o valor procurado for igual ao item do meio, a busca encerra.
3. Se não, verifica se o valor buscado é maior ou menor que o valor central.
4. Se for maior, então a busca acontecerá na metade superior da sequência (a inferior é descartada), se não for maior, a busca acontecerá na metade inferior da sequência (a superior é descartada).
5. Repete os passos: 1, 2, 3, 4.

Atenção: Veja que o algoritmo, ao encontrar o valor central de uma sequência, a divide em duas partes, o que justifica o nome de busca binária.

A figura seguinte ilustra o funcionamento do algoritmo, na busca pelo número 14 em uma certa sequência numérica.

Busca binária



Fonte: elaborada pela autora.

Veja que o algoritmo começa encontrando o valor central, o qual chamamos de m1. Como o valor buscado não é o central, e é maior que o elemento do meio da lista, a busca, então, passa a acontecer na metade superior. Dado o novo conjunto, novamente é localizado o valor central, o qual chamamos de m2, que também é diferente e menor do valor buscado. Mais uma vez a metade superior é considerada e, ao localizar o valor central (m3), agora sim o valor procurado, então o algoritmo encerra.

O código que representa a figura apresentada será escrito da seguinte maneira:

```
1  def busca_binaria(lista, elemento):
2          minimo = 0
3          maximo = len(lista)-1
4          encontrado = False
5
6          while minimo < =maximo and not encontrado:
7              meio_lista = (minimo + maximo)//2
8              if lista[meio_lista] == elemento:
9                  encontrado = True
10             else:
11                 if elemento < lista[meio_lista]:
12                     maximo = meio_lista-1
13                 else:
14                     minimo = meio_lista+1
15
16         return encontrado
17
18  testelista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
19  print(busca_binaria(testelista, 14)
```

Fonte: elaborada pela autora.

Para testar o comportamento desse algoritmo de **busca binária** vamos acessar o código com o Python Tutor.

Vamos a outro exemplo de aplicação do algoritmo de busca binária com utilização do mesmo vetor ordenado apresentado para a busca sequencial.

O código para esse exemplo será escrito da seguinte maneira:

```
1   def busca_binaria(lista, elemento):
2           minimo = 0
3           maximo = len(lista)-1
4           encontrado = False
5
6           while minimo< =maximo and not encontrado:
7               meio_lista = (minimo + maximo)//2
8               if lista[meio_lista] == elemento:
9                   encontrado = True
10              else:
11                  if elemento < lista[meio_lista]:
12                      maximo = meio_lista-1
13                  else:
14                      minimo = meio_lista+1
15
16          return encontrado
17
18   testelista = [1, 2, 8, 10, 13, 15, 18, 20]
19   print(busca_binaria(testelista, 5))
20   print(busca_binaria(testelista, 15))
```

Fonte: elaborada pela autora.

Para testar o comportamento desse algoritmo vamos acessar o código com o Python Tutor.

Agora vamos analisar alguns pontos importantes desse exemplo de aplicação de busca.

Linha 6

Na linha 6, temos a estrutura de repetição “while”, que será executada enquanto o primeiro elemento da lista (mínimo) for menor ou igual ao máximo (último elemento) e o elemento procurado não for encontrado.

Linha 7

Na linha 7, identificamos o índice associado à metade da lista.

Linha 8

Na linha 8, temos uma estrutura de condição “if” em razão da qual, basicamente, verifica-se que, se o elemento do meio da lista for o valor procurado, será retornado o True (linha 9).

Linha 10

Na linha 10, temos a condição “else”, que verifica se o elemento procurado é menor que o valor do meio da lista. Se for maior, então a busca acontecerá na metade superior da sequência (a inferior é descartada); se não for, a busca acontecerá na metade inferior da sequência (a superior é descartada).

Algoritmos de busca compõem o arsenal de algoritmos tradicionais da computação. Quando é conhecida a sequência de passos, ou seja, o pseudocódigo, basta escolher uma linguagem de programação e

implementá-la.

Pesquise mais

Todo algoritmo deve ter uma medida de complexidade – ou seja, a quantidade de recurso computacional é necessária para executar tal solução. A análise da complexidade do algoritmo fornece mecanismos para medir o desempenho de um algoritmo em termos de "tamanho do problema versus tempo de execução" (TOSCANI; VELOSO, 2012). A análise da complexidade é feita em duas dimensões: espaço e tempo. Embora ambas as dimensões influenciem na eficiência de um algoritmo, o tempo que ele leva para executar é tido como a característica mais relevante.

Fonte: Shutterstock.

Para aprofundamento, faça a leitura das páginas 13, 14 e 15 do Capítulo 2 da seguinte obra, disponível na Biblioteca Virtual:

TOSCANI, L. V; VELOSO, P. A. S. **Complexidade de algoritmos**: análise, projeto e métodos. 3. ed. Porto Alegre: Bookman, 2012.