

Algoritmos e Estruturas de Dados I  
Aula16

# Algoritmos de Busca e Ordenação

---

**Prof. MSc. Adalto Selau Sparremerberger**

 assparremerberger@senacrs.com.br

  @adaltoss  
 

 /assparremerberger

# Algoritmos de Busca

- Todos nós fazemos buscas, consciente e inconscientemente, todos os dias.
- Fazemos isso quando, por exemplo, procuramos um restaurante onde possamos almoçar, ou uma roupa para usar ao longo do dia, ou determinado livro em uma ou mais estantes de uma biblioteca.
- Também fazemos buscas mentais, internamente, em nosso cérebro, ao procurar um rosto conhecido ou o significado de uma palavra que aprendemos anteriormente.

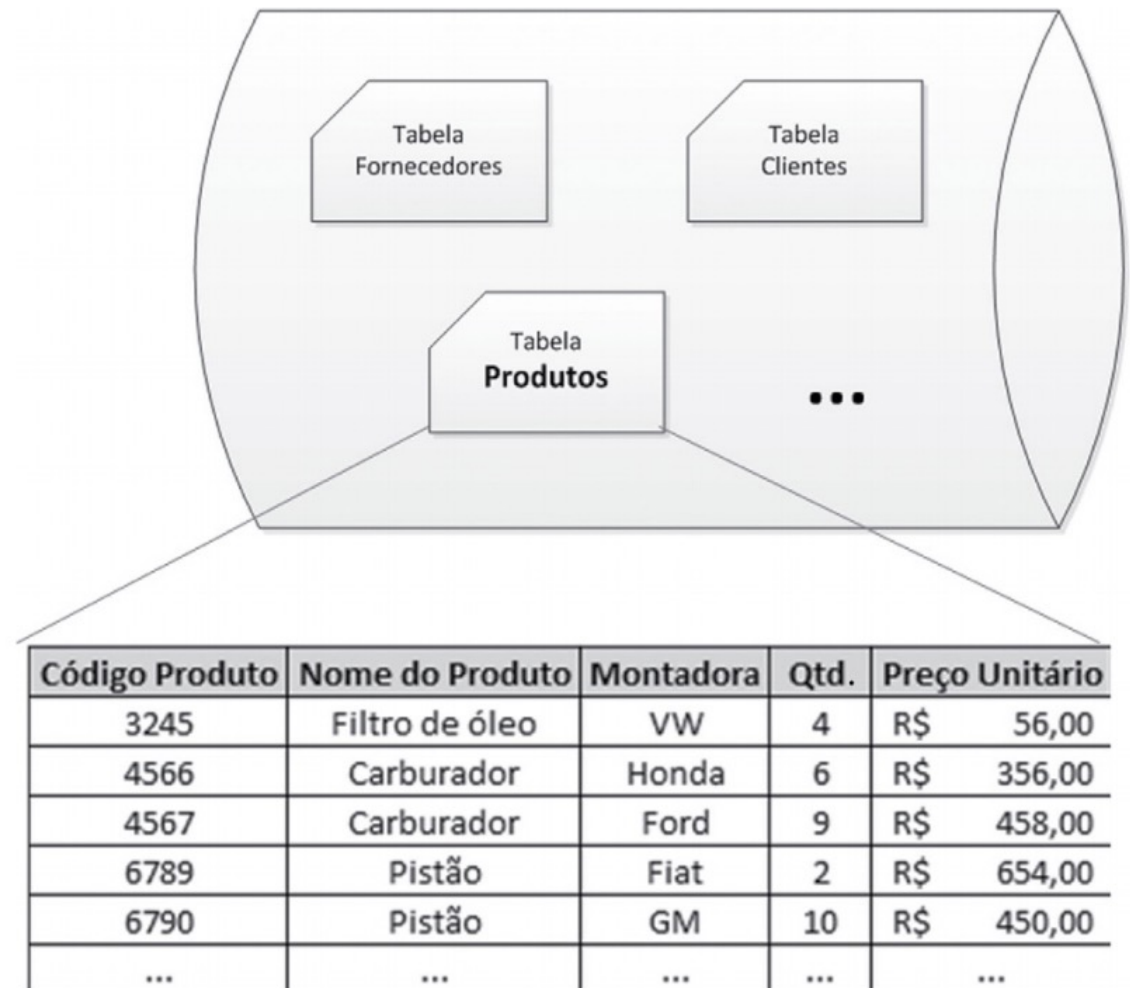
# Algoritmos de Busca

- Atualmente, guardar informações não é mais um problema.
- A capacidade das máquinas cresceu muito, e serviços de cloud computing, por exemplo, têm auxiliado ainda mais nessa tarefa.
- O grande problema está no processo de recuperação dessas informações, principalmente quando precisamos recuperar uma informação de qualidade, algo que possamos realmente utilizar da forma como planejamos ou necessitamos.

# Algoritmos de Busca

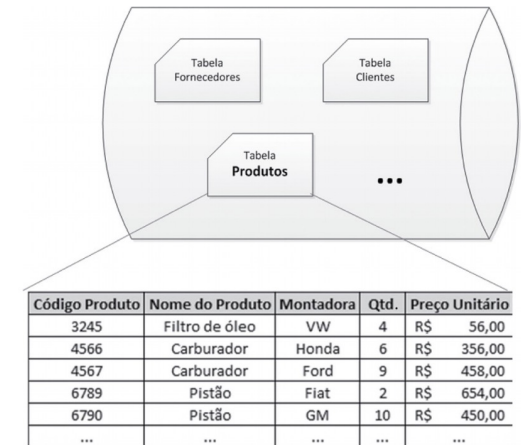
- Ilustração da organização da base de dados, com a tabela de produtos, seu registro e o campo de cada um.
- Na Figura as colunas “Código do produto”, “Nome do produto”, “Montadora”, “Qtd. e Preço unitário” são os campos. Cada linha corresponde às informações sobre um produto. A esse conjunto de informações damos o nome de registro. Assim, cada linha é um registro. O conjunto de todos os registros dos produtos forma a tabela de produtos. Essa tabela e outras ficam alocadas dentro de uma base de dados.

(BIANCHI, 2013)



# Algoritmos de Busca

- Voltando ao balcão: quando o cliente pergunta sobre determinado produto – por exemplo, “carburador da Ford” –, o funcionário vai até um sistema, busca o nome ou código do produto e retorna ao cliente, dizendo: “Sim, temos! E o preço é R\$ 458,00”.
- Aqui surgem algumas dúvidas: como o funcionário encontrou o produto especificado? Quanto tempo se passou até que o funcionário o encontrasse e respondesse ao cliente?
- Possivelmente, o funcionário acessou a base de dados através de um sistema automatizado que contava com uma ferramenta de busca. Assim, inseriu a informação passada pelo cliente em um local específico no sistema, e o produto, com suas respectivas informações, foi encontrado e exibido na tela do terminal de consulta.
- Isso faz com que perguntemos: como o sistema encontrou essa informação? Possivelmente, utilizando um algoritmo de busca. E como pode ser implementado um algoritmo de busca?



(BIANCHI, 2013)

# Algoritmos de Busca

- Bem, pensemos em um conjunto de informações mais simples, como um vetor de 10 elementos (números inteiros), conforme ilustrado na Figura 4.2.

4	78	12	3	65	21	34	77	98	11
---	----	----	---	----	----	----	----	----	----

**FIGURA 4.2:** Um vetor com 10 números inteiros. (BIANCHI, 2013)

- Se alguém lhe perguntasse se no vetor da Figura 4.2 existe o valor 34, você certamente responderia que sim. E poderia ser detalhista e completar dizendo que ele está na sétima posição.
- Você conseguiria descrever o método que utilizou para encontrar o valor 34 no vetor?
- A resposta mais frequente é: “Corri os olhos pelo vetor, da esquerda para a direita, verificando cada um dos valores, até encontrar o valor procurado: 34.”

# Algoritmos de Busca

## Algoritmo (em português estruturado)

Supondo um vetor  $v[]$ , um total de elementos,  $n$ , e o valor a ser encontrado,  $k$ , teríamos:

- percorra o vetor (da primeira à última posição);
- para cada elemento do vetor, verifique se é igual ao valor a ser encontrado;
- se for igual, retorne com a posição do vetor onde se encontra esse valor;
- caso contrário, continue procurando.

Se você chegou ao final do vetor e não encontrou o valor, retorne o valor -1.

# Algoritmo de Busca Sequencial

- O algoritmo de busca sequencial é utilizado para localizar um dado dentro de um vetor e este vetor pode estar ordenado ou não.
- Para iniciar o algoritmo, o dado que se deseja localizar é digitado pelo usuário, e um laço com as comparações é efetuado na quantidade de vezes igual ao número de dados do vetor ou até encontrar o dado desejado.



# Algoritmo de Busca Sequencial

- Como mencionado, para exemplificar essa forma de busca, vamos utilizar uma estrutura de dados do tipo vetor. Para tanto, o algoritmo de busca sequencial vai percorrer todo o vetor, do início ao fim, comparando o valor que se busca com cada elemento do vetor.
- Se durante a busca a comparação for positiva, ou seja, se o valor for encontrado, a posição do vetor será retornada.
- Caso a comparação chegue ao final do vetor, isso significa que o valor buscado não foi encontrado. Como uma estratégia nesse caso, o valor retornado geralmente é -1, pois o vetor não tem uma posição com esse valor.

# Algoritmo de Busca Sequencial

- Uma ou possibilidade de implementação pode ser a descrita no código a seguir.
- Implementação de uma função que receba como parâmetros um vetor  $v[]$  de números inteiros, um número inteiro  $n$ , que indica a quantidade de elementos nesse vetor, e por fim outro valor inteiro  $k$ , que seria o valor que se busca encontrar no vetor  $v[]$ . A função retorna a posição do elemento do vetor  $v[]$  igual a  $k$  (se existir) ou  $-1$ , se o valor não for encontrado. Vamos ao código!

```
int buscaSeq (int v[], int n, int k)
{
    int i;
    for (i=0; i<n; i++) {           // percorre todo o vetor v[]
        if (v[i]==k) return i;      // se encontra um valor igual, retorna a posição i
    }
    return -1;                     // caso o valor não seja encontrado, retorna o valor -1
}
```

(BIANCHI, 2013)

# Algoritmo de Busca Binária

- O algoritmo de busca binária é utilizado para localizar um dado dentro de um vetor ordenado e sem repetições.
- Para iniciar o algoritmo, o dado que se deseja localizar é digitado pelo usuário e um laço com as comparações é efetuado. A seguir, o algoritmo de busca binária é ilustrado com um vetor que possui dez posições e o dado procurado é o número 9.

# Algoritmo de Busca Binária

**Execução número 1 do laço:**

início = 1


fim = 10

posi = (início + fim) / 2

posi = (1 + 10) / 2 = 11 / 2 = 5,5

parte inteira de posi = 5

Vet	2	4	7	9	12	15	18	20	24	30
	1	2	3	4	5	6	7	8	9	10



Compara-se o número da posição 5 (número 12) com o número procurado (número 9). Se forem iguais, a execução do laço será encerrada; do contrário, uma nova posição será calculada. Como o número procurado é menor que o número da posição 5, temos um novo fim.

# Algoritmo de Busca Binária

**Execução número 2 do laço:**

$\text{inicio} = 1$

$\text{fim} = \text{posi} - 1 = 5 - 1 = 4$

$\text{posi} = (\text{inicio} + \text{fim}) / 2$

$\text{posi} = (1 + 4) / 2 = 5 / 2 = 2,5$

parte inteira de  $\text{posi} = 2$

Vet	2	4	7	9	12	15	18	20	24	30
	1	2	3	4	5	6	7	8	9	10
		↑								

Compara-se o número da posição 2 (número 4) com o número procurado (número 9). Se forem iguais, a execução do laço será encerrada; do contrário, uma nova posição será calculada. Como o número procurado é maior que o número da posição 2, então temos um novo início.

# Algoritmo de Busca Binária

**Execução número 3 do laço:**

$\text{inicio} = \text{posi} + 1 = 2 + 1 = 3$

$\text{fim} = 4$

$\text{posi} = (\text{inicio} + \text{fim}) / 2$

$\text{posi} = (3 + 4) / 2 = 7 / 2 = 3,5$

parte inteira de  $\text{posi} = 3$

Vet	2	4	7	9	12	15	18	20	24	30
	1	2	3	4	5	6	7	8	9	10
			↑							

Compara-se o número da posição 3 (número 7) com o número procurado (número 9). Se forem iguais, a execução do laço será encerrada; do contrário, uma nova posição será calculada. Como o número procurado é maior que o número da posição 3, então temos um novo início.

# Algoritmo de Busca Binária

**Execução número 4 do laço:**

$\text{inicio} = \text{posi} + 1 = 3 + 1 = 4$

$\text{fim} = 4$

$\text{posi} = (\text{inicio} + \text{fim}) / 2$

$\text{posi} = (4 + 4) / 2 = 8 / 2 = 4$

parte inteira de  $\text{posi} = 4$

Vet	2	4	7	9	12	15	18	20	24	30
	1	2	3	4	5	6	7	8	9	10

↑

Compara-se o número da posição 4 (número 9) com o número procurado (número 9). Se forem iguais, a execução do laço será encerrada. Como o número da posição 4 é igual ao número procurado, a busca será encerrada.

As comparações serão realizadas até o número procurado ser encontrado ou até a variável FIM ser menor que a variável INICIO.

# Algoritmo de Busca Binária

- Esse método só funciona se os elementos da estrutura de dados (no caso, vetor) estiverem ordenados. Ele utiliza a ideia de “dividir para conquistar”: a divisão acontece sempre comparando o valor que se busca com o elemento localizado no meio do vetor.
- Ao se realizar essa comparação, há três possibilidades: (1) o valor desse elemento central é igual à chave de busca; (2) o valor é menor do que a chave de busca; ou (3) o valor é maior do que a chave de busca.
- No primeiro caso, basta retornar à posição central e encontramos o valor. No segundo caso, se os elementos do vetor estiverem em ordem crescente e o elemento central for menor do que a chave de busca, isso significa que o valor dessa chave, se existir, estará na parte superior do vetor e a parte inferior (da metade ao início) será totalmente ignorada.
- O mesmo princípio se aplica ao terceiro caso, mas desta vez o elemento que se busca, se existir, estará na parte inferior do vetor e a parte superior (da metade até o final do vetor) será totalmente ignorada.



# Algoritmo de Busca Binária

- Depois dessa primeira divisão, o novo vetor passará a conter apenas os elementos da parte selecionada, e uma nova busca será feita com o elemento central. Ocorrerá uma nova divisão e assim sucessivamente, até que se encontre o valor procurado ou se retorne um valor de divisão igual a zero (ou seja, não existem mais elementos no processo de divisão do vetor).
- Nesse caso, não existe valor igual à chave de busca. Uma possibilidade de implementação desse método de busca pode ser o do código, a seguir.

# Algoritmo de Busca Binária

```
int buscaBin (int v[], int n, int k)
{
    int inicio=0;                // início do vetor ou de parte do vetor (primeiro elemento)
    int fim=n-1;                 // final do vetor ou de parte do vetor (último elemento)
    int centro;                  // posição central do vetor
    while (inicio <= fim) {      // enquanto existir elementos no vetor...
        centro=inicio+(fim-inicio)/2; // recebe a posição central do vetor
        if (k == v[centro]) return centro; // caso (1) – encontrou o valor
        else if (k > v[centro]) // caso (2) – o valor do elemento central é menor que k
            inicio=centro+1; // nesse caso (2) o novo vetor passa a ser a parte superior.
        else // caso (3) – o valor do elemento central é maior que k
            fim=centro-1; // nesse caso (3) o novo vetor passa a ser a parte inferior.
    }
    return -1;                  // caso o valor não seja encontrado, retorna o valor -1
}
```

(BIANCHI, 2013)

# Métodos de Ordenação

- Refere-se ao consumo de tempo (ou seja, o "tempo de resposta") do algoritmo.
- Em todas as definições,  $n$  é o tamanho de uma instância do problema que o algoritmo resolve.
- Um algoritmo é:
  - **Quadrático** se consome tempo proporcional a  $n^2$ ,
  - **Logarítmico** se consome tempo proporcional a  $\log n$ ,
  - **Linear** se consome tempo proporcional a  $n$ ,
  - **Cúbico** se consome tempo proporcional a  $n^3$ ,
  - **Exponencial** se consome tempo proporcional a  $cn$ , sendo  $c$  uma constante

# **Métodos de Ordenação Quadráticos**

- Bubble Sort
- Insertion Sort
- Selection Sort

# Complexidade n-Logarítmica

- A quantidade de operações básicas necessárias (ou o tempo) é proporcional a **Log  $n$** , onde  **$n$**  é um parâmetro que indica o tamanho do problema.
- E o que é **Log  $n$** ? É o logaritmo de  $n$ , por exemplo, na base 2 (Binário).
- Por exemplo: Para encontrar um elemento em uma árvore binária perfeitamente balanceada, a complexidade do algoritmo é **Log  $n$** .
- Isso quer dizer o seguinte: Para achar um elemento em uma árvore binária com 4 elementos ( $2^2$ ) é necessário fazer no máximo **2 comparações** (ou iterações), ou seja, **log de 4 na base 2 é 2**.
- Para fazer a mesma consulta em uma árvore com 1024 elementos ( $2^{10}$ ) é necessário no máximo fazer **10 comparações** (ou iterações), ou seja, **Log de 1024 na base 2 é 10**.

# Métodos de Ordenação $n$ -Logarítmicos

- Heap Sort
- Merge Sort
- Quick Sort

# Métodos de Ordenação Lineares

- Count Sort
- Radix Sort
- BucketSort

# Desafio

- Construa um algoritmo de busca contendo um vetor de números inteiros de 20 posições.
- O algoritmo deve ter duas funções, uma para busca sequencial e outra para busca binária.
- Coloque um contador em cada função para contar as iterações de cada função.
- Peça ao usuário que informe o valor que deseja procurar.
- Então informe ao usuário se este valor existe no vetor e quantas iterações foram necessárias em cada função.



# Referências

- PUGA, Sandra; RISSETTI, Gerson. **Lógica de Programação e estruturas de dados, com aplicações Java**. Person, 2016.
- DOBRUSHKIN, Vladimir A. **Métodos para Análise de Algoritmos**. LTC, 03/2012.
- BIANCHI, Francisco. **Estruturas de Dados e Técnicas de Programação**. Elsevier, 2013.
- MARTINS, Paulo Roberto. **Algoritmos e estrutura de dados\_ análise e desenvolvimento de sistemas**. Pearson, 2009.