

Algoritmos e Estruturas de Dados

Caros alunos, as vídeo aulas desta disciplina encontram-se no AVA (Ambiente Virtual de Aprendizagem).

| Unidade 2

Algoritmos de Busca e Ordenação de Vetores

Introdução da Unidade

Olá aluno. Nesta unidade vamos começar a aprofundar nosso trabalho com *Arrays*. Vamos compreender o motivo da utilização dessa estrutura como recurso e meio para outras tarefas envolvendo a área de desenvolvimento de sistemas. Uma dessas tarefas é uma rotina que realizamos várias vezes por dia. Essa tarefa é a busca, e podemos citar vários exemplos, como, a busca por arquivos no computador, por um documento físico ou digital, por um e-mail importante, pelo comprovante de votação, enfim, todo e qualquer tipo de busca pode envolver um sistema, uma rotina de código que otimizada será muito útil para aumentar a eficácia e velocidade desse processo.

Vamos estudar alguns dos tipos de busca mais conhecidas e que são amplamente utilizadas na solução de problemas computacionais e no desenvolvimento de aplicações de todas as áreas. Nesta unidade, iremos abordar os tipos de busca SEQUENCIAL e BINÁRIA e ao final, já poderemos utilizar os recursos desses tipos de busca para resolução de muitos algoritmos que envolvem essa ideia.

Iremos estudar também o processo de ordenação de elementos, presente em algumas técnicas algorítmicas existentes. Será explicado por meio de conceitos e exemplos práticos como e porque realizar a ordenação dentro de um *Array*.

O que vamos estudar nesta unidade será de suma importância para trabalhar com *Arrays* ordenados, além de conhecer e aplicar o processo de busca de elementos dentro dessa estrutura.

Os conhecimentos prévios para esta unidade são os conteúdos que foram abordados na Unidade 1 dessa disciplina. Espero que tenha estudado e realizado os exercícios desenvolvidos nas videoaulas apresentadas na Unidade 1. Contudo, este conteúdo será apresentado por meio de uma linguagem dialógica facilitando assim a sua compreensão. Para esta unidade, vamos continuar desenvolvendo os exercícios utilizando a linguagem de programação Java, mas vale lembrar, que os conteúdos estudados são aplicáveis a qualquer outra linguagem que você queira utilizar.

Objetivos

- Conhecer e compreender os problemas de busca em um *Array*;
- Conhecer e aplicar os métodos de Busca com a estrutura de um *Array* Unidimensional;
- Conhecer e aplicar as técnicas de Ordenação de Vetores;
- Aplicar o processo de busca em *Arrays* já ordenados.

Conteúdo programático

Aula 01 – Algoritmos de Busca

Aula 02 – Ordenação de Vetores



Você poderá também assistir às **videoaulas** em seu celular! Basta apontar a câmera para os **QR Codes** distribuídos neste conteúdo.

Aula 1 - Algoritmos de Busca

Por que estudamos Algoritmos de Busca?

Que os Algoritmos de busca são fundamentais em sistemas computacionais já se sabe. Mas por que motivo temos que aprender algoritmos de busca ou ainda quais suas principais funcionalidades? É isso que será abordado nesta Aula, desta unidade. Vamos ao nosso estudo!

Em ciência da computação, um algoritmo de busca, é um algoritmo que toma um problema como entrada e retorna a solução para o problema, geralmente após resolver um número possível de soluções. Porém, se considerarmos dentro da estrutura de dados, um algoritmo de busca é um algoritmo projetado para encontrar um item com um valor especificado dentro de um conjunto de itens. Esses itens podem estar armazenados individualmente dentro de um *Array* ou como registros em um banco de dados. Na maioria dos casos, os algoritmos estudados por cientistas da computação que resolvem problemas são algoritmos de busca (DEITEL, 2010).

Os objetos para aplicação dos algoritmos de busca, que começou a surgir na década de 1970, podiam ser dados cifrados, utilizados durante as duas primeiras guerras. Os algoritmos de busca foram se desenvolvendo também com a construção dos primeiros computadores e da internet (DEITEL, 2010).

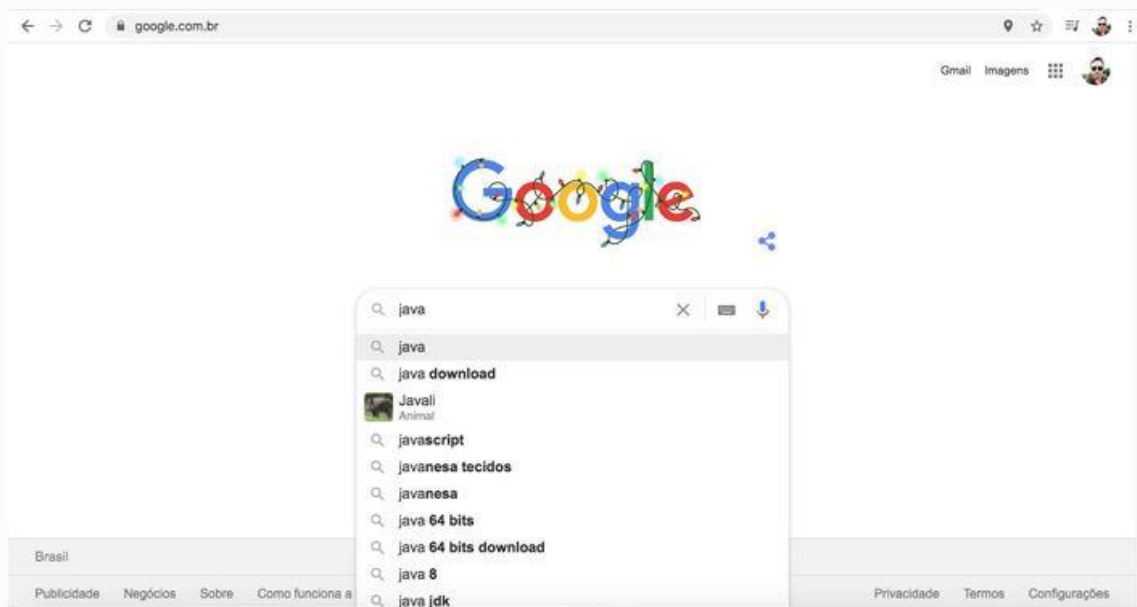
Na atualidade, os algoritmos de busca são aplicados em sistemas computacionais de todas as áreas, inclusive no mercado financeiro e principalmente nos grandes portais de busca na *web*. É comum por exemplo, utilizarmos esses serviços quando quisermos uma recomendação para um serviço, saber mais sobre um produto, conhecer a idade de uma figura pública, ou até mesmo, encontrar um restaurante ou shopping mais próximo de onde estamos. Um outro exemplo, no mercado financeiro, seria a busca por uma corretora que cobre um valor menor para operar o seu capital na bolsa e ofereça alguns benefícios interessantes a longo prazo. Realmente é ilimitado o número de exemplos quando pensamos em algoritmos de busca.

Indicação de Vídeo

Veja este vídeo sobre Algoritmos de Busca do canal Programação Dinâmica. No vídeo é apresentado uma ideia geral do “problema de busca”: Disponível em: <<https://www.youtube.com/watch?v=M719mDyirW0>>. Acesso em: 30 nov. 2020.

A busca eficiente é muito mais importante do que parece ser. Imagine o tempo de busca em um sistema de saúde para leito de UTI de um paciente que está tendo um AVC. Questão de segundos podem definir a vida ou a morte deste paciente. Em outro exemplo, podemos citar a busca de um conteúdo em um portal de busca na *internet*. Existem vários portais de busca, porém alguns creio que você nunca ouviu falar, portanto vamos apresentar os dois portais de busca mais conhecidos. O *Google* e o *Bing*. O *Google* é conhecido como o buscador universal, é o preferido por toda a população. Uma tela básica desse portal de busca é exibida na Figura 1.

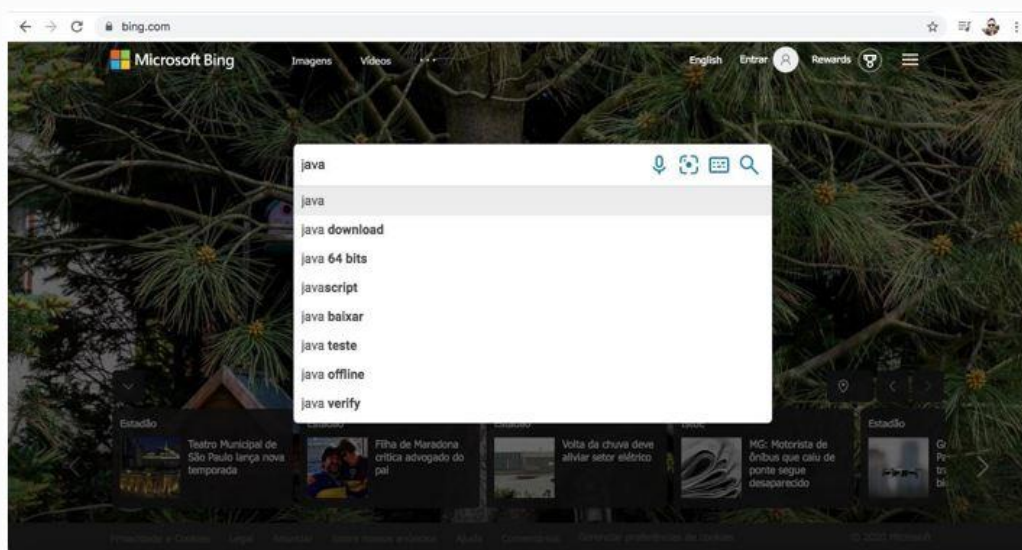
Figura 1 – Site de Pesquisa *Google*



Fonte: Google (2020)

O *Bing* é o buscador oficial da Microsoft e principal concorrente do *Google*. Se o computador que você comprou é da Microsoft, o navegador padrão é o *Edge* e o motor de busca é o *Bing*. A tela principal do *Bing* é exibida na Figura 2.

Figura 2 – Site de Pesquisa Bing



Fonte: Bing (2020)

Focando no exemplo, imagine que você está fazendo uma busca sobre a linguagem de programação Java nos dois portais de busca: *Google* e *Bing*. O algoritmo de busca utilizado por um portal pode não ser o mesmo utilizado pelo outro, isso fará com que uma busca possa ser mais rápida que a outra em algumas situações. O tempo de busca depende inclusive do conteúdo que se está buscando e onde esse conteúdo existe. Esses, entre outros fatores, determinarão o tempo de busca e o resultado das informações retornadas ao navegador.

Videoaula 1

Nesta videoaula, o professor irá explicar de forma introdutória a importância da busca em algoritmos. No vídeo, você verá que existem vários algoritmos de busca com o intuito de resolver o problema da busca.



Videoaula 1

Utilize o QRcode para

No vídeo, você verá que existem vários algoritmos de busca com o intuito de resolver o problema da busca.



Continuando nosso conteúdo, é muito comum pensarmos que as únicas maneiras de se fazer busca em informações são em tabelas de Banco de Dados. Mas na verdade, TABELA é um termo relacionado, ou seja, um termo genérico que pode ser qualquer estrutura de dados utilizada inclusive para armazenamento em memória. E é exatamente neste cenário que os algoritmos de busca são mais eficazes.

Dentro de uma TABELA, é possível realizar operações com seus elementos. As operações possíveis em uma tabela são:

- **Inserção:** operação utilizada para adicionar um novo elemento à tabela;
- **Remoção:** operação utilizada para retirar um elemento existente da tabela;
- **Recuperação ou Pesquisa:** busca um elemento escolhido na tabela;

Entre as maneiras que uma Tabela pode ser apresentada, estão: um *Array*, uma lista encadeada ou uma árvore. Além disso, a Tabela pode estar totalmente na memória (busca interna), totalmente no armazenamento auxiliar (busca externa) ou até mesmo dividida em ambas.

Vamos apresentar nesta aula duas técnicas de busca em memória interna. São elas: SEQUENCIAL E BINÁRIA.

Busca Sequencial

O objetivo da busca sequencial é pesquisar um elemento dentro de um conjunto de elementos. Essa busca recebe esse nome, pois ele faz a varredura de forma sequencial, um após o outro. Vamos considerar para o nosso estudo um *Array* Unidimensional, já abordado na Unidade de Ensino 1.

A busca sequencial é uma das maneiras mais simples de se buscar um elemento em uma tabela como um vetor ou uma lista. O vetor, também conhecido como *Array* Unidimensional, é uma das matérias primas no desenvolvimento de algoritmos. Ele pode ser uma lista, uma fila, uma pilha ou ainda simplesmente utilizado como uma variável no código.

A busca sequencial é o que fazemos quando temos que encontrar um livro específico em uma pilha de livros, por exemplo. Um a um você tem que retirar até encontrar o livro que está procurando. Imagine que você tenha uma pilha de quinze livros para procurar um título específico. No melhor caso, você pode se deparar com o livro que você está procurando no topo da lista, fazendo com que você consiga realizar a busca rapidamente. No pior caso, você pode ter que sequencialmente procurar o livro até o último livro da pilha. Como pode-se perceber, neste caso a pesquisa demorou o tempo máximo necessário. Esse tempo pode ser bem pior considerando uma pilha de trinta ou cinquenta livros.

Um outro exemplo cotidiano do cenário acadêmico, se retrata quando o professor está corrigindo as provas dos alunos da sua disciplina. Se o professor tiver 40 alunos na sala e não colocar as provas em ordem alfabética, ele terá que fazer uma busca sequencial para entregar a prova para cada aluno. Não seria mais fácil colocar as provas em ordem alfabética primeiro? A resposta para esta questão é SIM. A busca sequencial é mais efetiva quando temos os elementos ordenados. Abordaremos a ordenação de vetores na Aula 2 dessa unidade de ensino.

Apresentamos um vetor a seguir, de 10 elementos, que será utilizado para demonstração da busca sequencial.

Figura 3 – Array Unidimensional de 10 posições

0	1	2	3	4	5	6	7	8	9
10	12	45	21	65	32	78	89	13	22

Fonte: elaborado pelo próprio autor (2020)

Na Figura 3, podemos perceber que o vetor está devidamente preenchido com elementos numéricos. Na busca sequencial, definimos o valor a ser pesquisado no vetor, também conhecido como chave. Se o valor for encontrado no vetor, ele é retornado. Já se o valor não for encontrado, mesmo após percorrido todos os elementos do vetor, é exibida uma mensagem ao usuário informando que o valor não foi encontrado. Isso significa que o valor não se encontra no vetor.

Dentro desse vetor, vamos realizar uma busca sequencial para encontrar o **valor = 65**, também chamado de “chave”.

Figura 4 – Busca Sequencial – valor a ser pesquisado

0	1	2	3	4	5	6	7	8	9
10	12	45	21	65	32	78	89	13	22

Fonte: elaborada pelo próprio autor (2020)

A partir da posição zero (0) do vetor iniciamos o processo de busca sequencial. O objetivo é fazer a busca até que o valor 65 seja encontrado, que neste caso, como a Figura 4, exibe o número 65 na posição 4, o valor será encontrado.

Para tal algoritmos, precisamos de um comando de laço de repetição que percorra o vetor do início ao fim, ou até encontrar o elemento procurado pela busca sequencial. O Algoritmo vai precisar de um apontador para a coluna do vetor, que será iniciado na posição 0 (zero), indo até o final da estrutura. Para a pesquisa, a cada novo elemento percorrido, deve-se verificar por meio de uma estrutura de seleção, se o valor procurado é igual ao valor encontrado, neste caso o valor será retornado e depois exibido na tela do Algoritmo. A Figura 5, representa a chamada

do método “**buscaSequencial()**”, passando como parâmetro de entrada o valor 65. O método armazena o retorno na variável “ret”, e depois exibe a variável na tela. Observe:

Figura 5 – Função de busca sequencial sendo executada

```
25 | | | int ret = buscaSequencial(65);  
26 | | | System.out.println(ret);
```

Fonte: elaborada pelo próprio autor (2020)

A linha 25, da Figura 5, é responsável pela execução do método, armazenando o retorno na variável. A Figura 6, apresenta a construção e a lógica implementada no algoritmo da função.

Figura 6 – Construção da função de busca sequencial

```
30 | public static int buscaSequencial(int valorPesquisado){  
31 |  
32 |     int[] vetor = new int[]{10,12,45,21,65,32,78,89,13,22};  
33 |  
34 |     for(int i = 0; i <= 9; i++){  
35 |  
36 |         if(vetor[i] == valorPesquisado){  
37 |             return vetor[i];  
38 |         }  
39 |  
40 |     }  
41 |     return 0;  
42 | }
```

Fonte: elaborada pelo próprio autor (2020)

O método construído em Java, figura 6, aplica no vetor a busca sequencial com o objetivo de varrer a estrutura em busca do valor passado por parâmetro. A linha 30, cria o método do tipo inteiro e um parâmetro do mesmo tipo, para que seja atribuído ao método no momento de sua chamada. A linha 32, cria um *Array* Unidimensional com o nome de vetor que recebe os elementos na mesma ordem da Figura 3.

A partir da linha 34, o método começa a varrer o vetor do início ao fim, buscando o valor passado por parâmetro no vetor. Se o valor for encontrado, a linha 37 retorna o valor do vetor, caso contrário o método retorna 0(zero), significando que o valor procurado não foi encontrado no vetor.

Videoaula 2

Nesta videoaula, será abordado um exemplo de busca sequencial em memória interna. Vale lembrar que a busca sequencial também pode ser aplicada em memória externa ou dividida entre ambas. No vídeo, será desenvolvido na prática um algoritmo que implemente este algoritmo da busca.



Videoaula 2

Utilize o QRcode para

No vídeo, será desenvolvido na prática um algoritmo que implemente este algoritmo da busca.



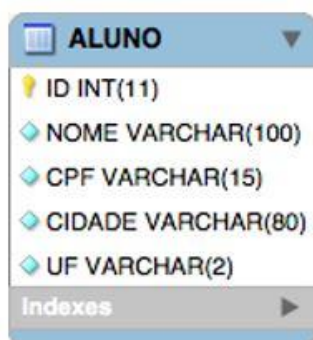
Busca Binária

Outra forma de se buscar um elemento em um *Array*, é a busca binária. Esse tipo de busca realiza um processo bem diferente de busca na estrutura. A busca binária pode ser usada com a organização de tabela sequencial indexada. Mas o que é uma tabela indexada?

A ideia de indexação, está presente no Banco de Dados, onde são armazenadas todas as informações de uma organização. Os sistemas computacionais são responsáveis por validar e verificar todas as informações antes dela ser mantida efetivamente no Banco. As informações são mantidas no Banco mediante às Tabelas, organizadas por meio de várias características que definem o tipo de dado, a quantidade de campos (colunas) e os relacionamentos entre as tabelas (CHEN, 1976).

Uma das características de uma Tabela do Banco de Dados é o índice. Um índice é uma classificação relativa a um campo de uma tabela. Por exemplo: em uma tabela de Alunos contendo os campos de (**id, nome, cpf, cidade e uf**), podemos ter o índice de chave primária (*primary key*). Esse tipo de índice acontece automaticamente quando definimos o campo chave de uma tabela. Observa a Figura 7.

Figura 7 – Tabela de Banco de Dados



The image shows a software interface for defining a database table named 'ALUNO'. The fields are listed as follows:

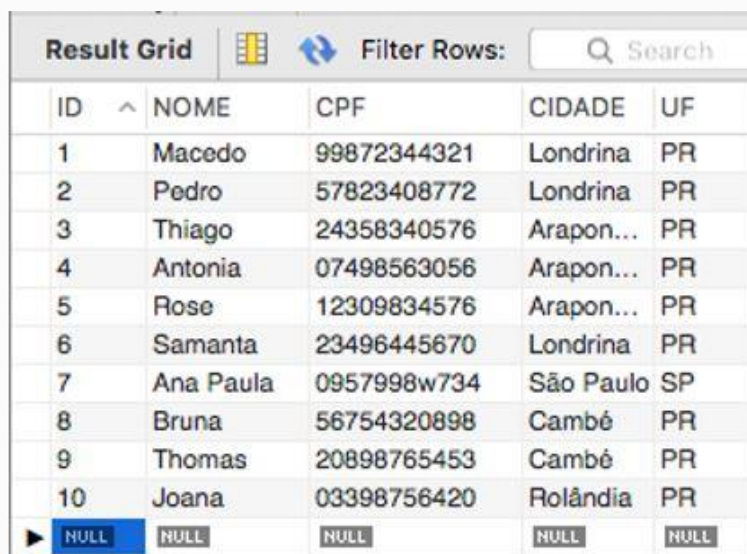
Field	Data Type
ID	INT(11)
NOME	VARCHAR(100)
CPF	VARCHAR(15)
CIDADE	VARCHAR(80)
UF	VARCHAR(2)

Below the fields, there is a section labeled 'Indexes' with a right-pointing arrow.

Fonte: elaborada pelo próprio autor (2020)

A Figura 7 apresenta a Tabela de Aluno dentro do Diagrama Entidade-Relacionamento (DER) (CHEN, 1976). Observe que no campo ID, existe uma identificação de uma chave amarela à esquerda do campo, indicando que esse campo é chave primária, portanto, um campo indexado. Na prática esse campo será responsável por uma ordem dos dados dentro da tabela.

Figura 8 – Registros da Tabela Aluno



The image shows a 'Result Grid' with columns: ID, NOME, CPF, CIDADE, and UF. The records are ordered by the primary key ID. The last row shows NULL values for all fields.

ID	NOME	CPF	CIDADE	UF
1	Macedo	99872344321	Londrina	PR
2	Pedro	57823408772	Londrina	PR
3	Thiago	24358340576	Arapon...	PR
4	Antonia	07498563056	Arapon...	PR
5	Rose	12309834576	Arapon...	PR
6	Samanta	23496445670	Londrina	PR
7	Ana Paula	0957998w734	São Paulo	SP
8	Bruna	56754320898	Cambé	PR
9	Thomas	20898765453	Cambé	PR
10	Joana	03398756420	Rolândia	PR
NULL	NULL	NULL	NULL	NULL

Fonte: elaborada pelo próprio autor (2020)

Na Figura 8, são apresentados os dados da Tabela de Aluno com o campo ID, sendo chave primária da Tabela. A ordem natural de inserção dos registros é o campo ID, ou seja, os registros respeitam a ordem da chave primária e não a ordem alfabética, pois a indexação presente é por meio da chave. Se quiséssemos ordenar os dados dessa tabela para aplicar a busca binária a fim de encontrar um aluno pelo campo nome, temos que criar um índice para este campo.

Agora que você já compreendeu o que é a indexação, vamos novamente focar em nosso *Array Unidimensional* para explicar esse tipo de busca.

A Figura 9, apresenta o nosso *Array* Unidimensional, porém indexado, ordenado. Esta ordenação compreende nos mesmos elementos apresentados anteriormente, porém desta vez ordenados do menor para o maior e da esquerda para a direita. Observe ainda que o início e fim da estrutura se dá por meio do índice e não pelo elemento em si.

Figura 9 – Array Unidimensional Ordenado

0	1	2	3	4	5	6	7	8	9
10	12	13	21	22	32	45	65	78	89

Fonte: elaborada pelo próprio autor

Na busca binária, o elemento buscado é comparado ao elemento do meio do *Array*. Se o elemento for igual, então busca bem-sucedida, caso contrário verifica-se outras situações. Em primeiro lugar, se o elemento buscado é menor que o elemento do meio do *Array*, busca-se na metade inferior do *Array*, se o elemento for maior, busca-se na metade superior do *Array*.

Vamos realizar uma busca pelo número 13, um valor que se encontrado no índice dois (2) do *Array*. Pela busca binária, é encontrado o meio do *Array* no elemento 22, índice 4. Como o valor 13 é menor que 22, determina-se a busca na metade inferior do *Array*.

Figura 10 – Busca Binária

Buscando pelo elemento: 13									
Inf=0	1	2	3	meio	5	6	7	8	Sup=9
10	12	13	21	22	32	45	65	78	89
13 < 22									

Fonte: elaborada pelo próprio autor (2020)

Como o valor 13 é menor que o meio do *Array*, a busca binária agora determina que só seja buscado os índices de 0 a 3, figura 11.

Figura 11 – Busca Binária

Buscando pelo elemento: 13									
Inf=0	1	2	3	meio	5	6	7	8	Sup=9
10	12	13	21	22	32	45	65	78	89
13 < 22									

Fonte: elaborado pelo próprio autor (2020)

Mais uma vez repete-se o processo. Para o meio do *Array*, podemos determinar o elemento de número 13, sendo, portanto, exatamente o valor buscado e o algoritmo irão retornar o valor com sucesso.

Figura 12 – Continuação da Busca Binária

Buscando pelo elemento: 13									
Inf=0	1	meio	Sup=3	4	5	6	7	8	9
10	12	13	21	22	32	45	65	78	89
13 = 13									

Fonte: elaborado pelo próprio autor (2020)

A Figura 12 retrata a busca binária tendo sucesso, isso é, quando o algoritmo encontra o valor buscado no *Array*. Podemos concluir também que a Busca Binária torna o processo de busca mais eficiente, visto que a cada comparação, reduz o número de possíveis candidatos por um fator de 2.

Pode-se destacar uma das vantagens na busca binária: a eficiência e a simplificada de implementação do algoritmo. Como desvantagens se destacam: que nem todo *Array* está ordenado, e que a inserção e remoção são ineficientes nesta busca.

A seguir, na Figura 13, veja um exemplo do código implementando a Busca Binária.

Figura 13 – Construção da função de busca binária

```
49 public static int buscaBinaria(int v[], int n, int x){
50
51     int inf = 0, sup = n - 1, meio;
52
53     while (inf <= sup){
54
55         meio = (inf + sup) / 2;
56
57         if(x == v[meio])
58             return v[meio];
59         else if(x < v[meio])
60             sup = meio - 1;
61         else inf = meio + 1;
62     }
63
64     return 0;
65 }
```

Fonte: elaborada pelo próprio autor (2020)

Videoaula 3

Nesta videoaula, será abordado um exemplo de busca binária em memória interna. No vídeo, será desenvolvido na prática um algoritmo que implemente este algoritmo da busca.



Videoaula 3

Utilize o QRcode para assistir!

Nesta videoaula, será abordado um exemplo de busca binária em memória interna. No vídeo, será desenvolvido na prática um algoritmo que implemente este algoritmo da busca.



Nesta aula você pode identificar e compreender as características dos algoritmos de busca em estruturas de lista. Foi apresentado duas técnicas de busca, foram elas: busca sequencial e busca binária. Cada técnica com suas diferenças de conceitos, aplicações e comportamentos.

Na próxima aula, você verá o processo de ordenação de elementos dentro de um *Array* Unidimensional.

Aula 2 - Ordenação de Vetores

Introdução

Vamos iniciar nossa segunda aula falando um pouco sobre o processo de organização e estruturação dos dados. Já percebemos que ser uma pessoa organizada ou manter o hábito de se colocar objetos e ou arquivos em ordem, traz muitos benefícios. Por exemplo, o comprovante de votação é um documento que você precisa para fazer a inscrição em um concurso público ou a matrícula em uma Universidade pública. Saber exatamente onde este documento está armazenado, mesmo que você não o veja há 2 anos vai economizar muito tempo.

Na organização dos dados, isso também acontece. Se mantermos nossos dados organizados ou ordenados, quando precisamos encontrar algo o processo é mais simples e mais rápido. Por isso estudamos os Algoritmos de Busca na Aula 1 dessa Unidade de Ensino, porém quando a estrutura, no caso o *Array*, está ordenado a busca é realizada em alguns casos com a metade do tempo.

Neste momento é hora de abordar justamente o processo de ordenação dessas estruturas. A ordenação pode acontecer em estruturas com valores caracteres e numéricas. As Figuras 1 e 2 mostram respectivamente um *Array* Unidimensional desordenado e ordenado.

Figura 1 – Array Unidimensional Desordenado

0	1	2	3	4	5	6	7	8	9
10	12	45	21	65	32	78	89	13	22

Fonte: elaborada pelo próprio autor (2020)

Figura 13 – Array Unidimensional Ordenado

0	1	2	3	4	5	6	7	8	9
10	12	13	21	22	32	45	65	78	89

Fonte: elaborada pelo próprio autor (2020)

Como já abordado na Aula 1, podemos ver que a busca binária, por exemplo, necessita que a estrutura em questão esteja ordenada. Já a busca sequencial, pode ser realizada com a estrutura desordenada também, embora ordenada, a busca sempre acontecerá mais rapidamente.

Indicação de Leitura

Para melhor compreensão deste tópico, você pode estudar o livro do Manzano. O capítulo 7, “Classificação dos Elementos de uma Matriz”, especificamente apresenta um processo passo a passo de ordenação de elementos. Vale a pena conferir!

MANZANO, Jose Augusto N. G. **Estudo Dirigido:** algoritmos. São Paulo: Érica, 1997. 265 p.

A ordenação de informações é inerente aos sistemas de informações. Por exemplo, na preparação de um relatório de notas de uma turma, pode-se, em dado momento, exibi-lo ordenado por notas decrescentes. Para tal, podemos utilizar os vetores como matéria-prima neste processo, atuando como a estrutura que contém os elementos a serem ordenados.

De acordo com Laureano (2012), uma das técnicas de ordenação de vetores, denominada *Insertion Sort*, consiste em realizar a ordenação de elementos posicionando os menores no início do *Array* comparando os dois elementos. Esse processo é realizado por meio de laços de repetições varrendo os elementos do vetor do início até o final da estrutura quantas vezes forem necessárias até que toda a estrutura esteja corretamente ordenada.

Indicação de Vídeo

Veja este vídeo sobre a Ordenação de Vetores. No vídeo, por meio de um vetor humano, é realizada o teste de mesa de todo o processo de ordenação do *Array*. Pause o vídeo quantas vezes for necessário e faça anotações para que você compreenda o processo realizado na implementação via código. Disponível em: <<https://www.youtube.com/watch?v=ROaIU379l3U&t=75s>>. Acesso em: 15 nov. 2020.

O vídeo retrata em detalhes todas as comparações possíveis para ordenar a estrutura. Realizando o teste de mesa, é possível identificar também a técnica de ordenação utilizada. Por exemplo, quando percebemos por meio do teste de mesa que os elementos menores estão sendo deslocados para a esquerda do vetor, primariamente podemos classificar essa técnica como: *Insertion Sort*. Por outro lado, se os elementos de maior valor no *Array* estão sendo deslocados para a direita ou topo da estrutura, também primariamente, podemos classificar essa técnica como *Bubble Sort*.

Para obtermos a ordenação completa do vetor, há duas alternativas: a primeira é inserir os elementos diretamente na estrutura, respeitando a ordenação (ordenação garantida por construção), ou a partir de uma estrutura já composta de todos os elementos e aplicar uma das técnicas de ordenação de vetores.

Outro fator importante a ser considerado é como será construído o algoritmo de ordenação no código. Nesse caso, pode ser colocado o código do algoritmo dentro de uma função sem retorno

do tipo *void()* e chamar esta função dentro do método *main()*. Uma vez que o vetor esteja ordenado, basta varrê-lo do início ao fim exibindo-o na tela do programa.

Por fim, é preciso lembrar que o objetivo das técnicas de ordenação de vetores é colocar a estrutura na ordem predefinida previamente. Ela pode ser crescente ou decrescente dependendo do requisito do algoritmo.

A Figura 14 exibe uma comparação clássica em um *Array* Unidimensional, testando se os elementos dentro do *Array* estão em lugares que necessitam ser trocados visando a ordenação da estrutura.

Figura 14 – Código de comparação

```
42 while (j>=0 && aux < vetDes[j]){  
43     vetDes[j+1] = vetDes[j];  
44     j--;  
45 }  
46 vetDes[j+1] = aux;
```

Fonte: elaborada pelo próprio autor (2020)

A linha 42 da Figura 14 faz a comparação de dois elementos do *Array* “vetDes”, e retornando verdadeiro, o algoritmo inicia o processo de troca desses elementos.

O processo de ordenação é utilizado em todos os sistemas computacionais. As informações são armazenadas no banco de dados ordenados (CHEN, 1976). A partir disso, os *softwares* acessam e manipulam a informação do banco por meio de validações e verificações que garantem a integridade dos dados.

O Algoritmo que será apresentado, trabalha em laços de repetições internos que faz a verificação dos elementos dois a dois no vetor. Essa técnica tem a característica de levar os menores valores para a esquerda do vetor, isso considerando o vetor na posição horizontal. Se o vetor tiver na posição vertical, os valores são levados para o que chamamos de base da estrutura.

Essa técnica de ordenação conhecida como *Insertion Sort*, será a primeira a ser apresentada nesta aula.

A Figura 15 exibe uma prévia do algoritmo em Java para esta técnica. Observe que o algoritmo está sendo implementado por meio de uma função que faz o retorno da estrutura após sua execução.

Figura 15 – Técnica *Insertion Sort*

```
int i, j, atual;
for (i = 1; i < vetor.Length; i++)
{
    atual = vetor[i];
    j = i;
    while ((j > 0) && (vetor[j - 1] > atual))
    {
        vetor[j] = vetor[j - 1];
        j = j - 1;
    }
    vetor[j] = atual;
}
return vetor;
```

Fonte: elaborada pelo próprio autor (2020)

Na Figura 15, no primeiro momento são criadas as variáveis que servirão de apontadores e auxiliares no processo de ordenação. Em seguida, são necessários dois laços de repetições para aplicar o algoritmo de ordenação *Insertion Sort* no *Array*.

Videoaula 1

Nesta videoaula, será abordada a técnica de ordenação de vetores *Insertion Sort*. Acompanhe no vídeo este processo. Também será demonstrada a execução do código implementado em Java.



Videoaula 1

Utilize o QRcode para assistir!

Nesta videoaula, será abordada a técnica de ordenação de vetores *Insertion Sort*. Acompanhe no vídeo este processo. Também será demonstrada a execução do código implementado em Java.



Outra técnica bem interessante no processo de ordenação de um *Array* é a *Bubble Sort*. Essa técnica que quer dizer Ordenação por Flutuação, literalmente “por bolha”, também é bem simples.

Sua ideia é varrer o *Array* o número de vezes necessária e a cada passagem fazer flutuar para o topo o maior elemento do *Array*.

Veja o código desse vetor na Figura 16, apresentada a seguir:

Figura 16 – Técnica *Bubble Sort*

```
int tamanho = vetor.Length;
int comparacoes = 0;
int trocas = 0;

for (int i = tamanho - 1; i >= 1; i--)
{
    for (int j = 0; j < i; j++)
    {
        comparacoes++;
        if (vetor[j] > vetor[j + 1])
        {
            int aux = vetor[j];
            vetor[j] = vetor[j + 1];
            vetor[j + 1] = aux;
            trocas++;
        }
    }
}

return vetor;
```

Fonte: elaborada pelo próprio autor (2020)

Indicação de Vídeo

Veja este vídeo sobre a Ordenação de Vetores. Por meio de um vetor humano, é realizada o teste de mesa de todo o processo de ordenação do *Array*. Pause o vídeo quantas vezes for necessário e faça anotações para que você compreenda o processo realizado na implementação via código.

O vídeo tem o objetivo de apresentar o teste de mesa da técnica *Bubble Sort* de ordenação de um *Array*.

Disponível em: <<https://www.youtube.com/watch?v=lyZQPjUT5B4>>. Acesso em: 15 nov. 2020.

Outro vídeo muito interessante que apresentado em forma de dança, mostra na prática o processo de ordenação de um *Array* de 10 elementos por meio da técnica de ordenação *Bubble Sort*.

Videoaula 2

Nesta videoaula, será abordada a técnica de ordenação de vetores *Bubble Sort*. Acompanhe no vídeo este processo. Também será demonstrado a execução do código implementado em Java.



Videoaula 2

Utilize o QRcode para assistir!

Nesta videoaula, será abordada a técnica de ordenação de vetores Bubble Sort. Acompanhe no vídeo este processo. Também será demonstrado a execução do código implementado em Java.



Existem outras técnicas de ordenação presentes, vale a pena você pesquisar e tentar implementar os algoritmos no Java fazendo o teste de mesa. A técnica *Quick Sort* é conhecida como mais rápida. Outra técnica também existente é a *Merge Sort*, que trabalha fazendo uma ordenação por mistura, tal ordenação é realizada por comparação do tipo dividir-para-conquistar.

Indicação de Vídeo

Veja este vídeo sobre a Ordenação de Vetores. Ele apresenta mais uma variação do vetor humano realizando o teste de mesa de todo o processo de ordenação do *Array*. Esta técnica é a *Quick Sort*. Oriente você a assistir o vídeo com calma, e utilizando papel e caneta para fazer o teste de mesa e compreender o processo realizado na implementação do algoritmo.

O vídeo tem o objetivo de apresentar o teste de mesa da técnica *Quick Sort* de ordenação de um *Array*.

Disponível em: <<https://www.youtube.com/watch?v=ywWBy6J5gz8>>. Acesso em: 15 nov. 2020.

Indicação de Vídeo

A técnica apresentada neste é a *Merge Sort*. Proceda da mesma forma que nos outros vídeos.

O vídeo tem o objetivo de apresentar o teste de mesa da técnica *Merge Sort* de ordenação de um *Array*.

Disponível em: <https://www.youtube.com/watch?v=XaqR3G_NVoo>. Acesso em: 15 nov. 2020.

Os algoritmos de ordenação de vetores estão relacionados com a área dos algoritmos de busca. Vamos abordar algumas características envolvendo essas duas áreas da estrutura de dados. Uma maneira de fazer com que o algoritmo de busca fique mais eficiente é utilizar um SENTINELA.

Este método consiste em adicionar um elemento de valor “x” no final da tabela. Ele garante que o elemento será encontrado. Esse fator elimina um teste, melhorando assim a performance do algoritmo.

Algumas considerações sobre a complexidade da busca sequencial. Se o elemento pesquisado for encontrado no primeiro índice, foi feita apenas uma comparação. Se o elemento pesquisado estiver no último índice, foram feitas N comparações. Sendo provável que o elemento pesquisado esteja em qualquer índice do *Array*, em média são feitas $(n + 1) / 2$ comparações. Caso não seja encontrado o elemento pesquisado, a busca foi malsucedida e foram feitas N comparações. Logo, conclui-se que, no pior caso, a busca sequencial é $O(n)$.

Na busca sequencial, quando o *Array* não está ordenado, a inserção é realizada sempre no final da estrutura. Portanto, na remoção de um elemento, é necessário a realocação dos elementos que estão acima, ou após o elemento removido.

Para que seja aumentada a eficiência é pertinente que o *Array* seja reordenado continuamente e os registros mais acessados sejam deslocados para o início. Em um exemplo do cotidiano, um comerciante de vendas de eletrodomésticos de uma loja pode utilizar essa estratégia, ou seja, como ele já sabe quais são os produtos mais vendidos, pode colocar eles à frente da loja ou do departamento específico que ele se enquadra. Você já tinha parado para pensar que isso pode ser uma técnica de Busca Sequencial?

Existe muito conteúdo interessante a ser estudado sobre Busca Sequencial, como exemplo: inserção e remoção de elementos no *Array*, índices secundários, entre outros.

Indicação de Vídeo

A técnica apresentada neste é a *Shell Sort*. Proceda da mesma forma que nos outros vídeos.

O vídeo tem o objetivo de apresentar o teste de mesa da técnica *Shell Sort* de ordenação de um *Array*.

Disponível em: <<https://www.youtube.com/watch?v=CmPA7zE8mx0>>. Acesso em: 15 nov. 2020.

Curiosidades

O livro de FORBELLONE e EBERSPÄCHER, apresenta tópicos interessantes sobre este conteúdo da disciplina de Algoritmos e Estrutura de Dados.

O capítulo 5, a partir da página 116, apresenta o conceito do “Arquivo Direto Acessado Sequencialmente”. O Tópico também aborda exercícios de fixação do conteúdo.

Este livro consta na seção de Bibliografia desta Unidade de Ensino e apresenta com profundidade o universo do desenvolvimento por meio de algoritmos, apresentando as diversas estruturas com conceitos, características e exercícios práticos.

Em nossa Biblioteca Digital você vai encontrar estes e demais livros para aprofundar seus conceitos na disciplina de Algoritmos e Estrutura de Dados e outras disciplinas correlatas.

Link para Biblioteca Digital: Disponível em: <<https://web.unifil.br/pergamum/biblioteca/index.php>>. Acesso em: 14 nov. 2020.

Agora é hora de aplicar os exercícios de algoritmos de busca sequencial e binária em *Arrays* que já estão de fato ordenados. Observe e exercite a sua capacidade de teste de *software* verificando se realmente as buscas em estruturas ordenadas são efetivamente mais rápidas.

Videoaula 3

Agora, assista ao vídeo do professor, onde é abordado a aplicação de métodos dentro dos algoritmos de busca sequencial e binária. No vídeo, você verá o processo para construção e também para testar o método com valores fixos e também fornecidos pelo usuário.



Videoaula 3

Utilize o QRcode para assistir!

Agora, assista ao vídeo do professor, onde é abordado a aplicação de métodos dentro dos algoritmos de busca sequencial e binária. No vídeo, você verá o processo para construção e também para testar o método com valores fixos e também fornecidos pelo usuário.



Encerramento

Chegamos ao fim desta Unidade de Ensino após caminhar por conteúdos relevantes para o seu conhecimento e desenvolvimento acadêmico até aqui. A Unidade 2 apresentou detalhadamente as técnicas dos algoritmos de busca, exemplificando o seu funcionamento com aulas práticas de acordo com o conteúdo ministrado. Foi detalhado processos, como: a criação de um método com parâmetros obrigatórios de entrada, a manipulação de seus elementos e configuração dos retornos dos elementos encontrados no *Array*.

A Unidade 2 também apresentou com clareza o processo de ordenação de um *Array* de acordo com duas técnicas explicadas na Aula 2: *Insertion Sort* e *Bubble Sort*. Além da Unidade ser enriquecida com diversas indicações de vídeos que mostram na prática como os algoritmos de ordenação funcionam.

Após a assimilação dos conteúdos abordados nesta Unidade, você estará apto a prosseguir estudando as estruturas abordadas nas próximas Unidades de Ensino.

Tenha uma ótima leitura e não se esqueça de ler e estudar a Bibliografia e as aulas gravadas pelo professor.

Obrigado pela oportunidade de ser o mediador entre os conteúdos específicos abordados e o conhecimento adquirido por você por meio da didática e abordagem desta Unidade.

Referências

CHEN, P. P. S. **The entity-relationship model—toward a unified view of data**. ACM Transactions on Database System, v. 1, p. 9–36, 1976.

DEITEL, Harvey M.; DEITEL, Paul J. **Java: como programar**. 8. ed. São Paulo: Pearson, 2010.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação**. 2. ed. São Paulo: Makron Books, 2000. 195 p.

LAUREANO, M. **Estrutura de dados com algoritmos e C**. São Paulo: Brasport, 2012.

MANZANO, Jose Augusto N. G. **Estudo Dirigido: algoritmos**. São Paulo: Érica, 1997. 265 p.

Esperamos que este guia o tenha ajudado compreender a organização e o funcionamento de seu curso. Outras questões importantes relacionadas ao curso serão disponibilizadas pela coordenação.

Grande abraço e sucesso!

