

# Unidade 4

## Tabelas Hash e Busca com Retrocesso



# Introdução

Olá Aluno (a), nessa unidade de ensino serão abordadas mais duas novas estruturas de dados prosseguindo constantemente com nosso aprendizado. Pensando no fechamento da disciplina de Análise e Projeto de Algoritmos, as estruturas trabalhadas objetivam apresentar novas formas de solucionar problemas por meio de algoritmos.

Os Temas das duas aulas dessa Unidade de Ensino serão: Tabelas *Hash* ou ainda, Tabelas de Espalhamento e Busca com Retrocesso. Serão abordadas a ideia de resolução desses algoritmos, bem como, suas características e, acima de tudo, a utilização desses algoritmos dentro do cenário computacional.

Durante as aulas dessa Unidade de Ensino, serão construídos exemplos em vários contextos envolvendo Tabelas *Hash*. Esse algoritmo trabalha com índices para organização de sua estrutura. Esses índices por sua vez, são comparados ao controle e armazenamento por meio de um banco de dados. Os algoritmos de Busca com Retrocesso, também serão exemplificados utilizando cenários reais, com usabilidade dessa técnica de resolução de problemas.

Todo esse conteúdo será abordado considerando uma didática evolutiva, aliada a exemplos de aplicação dessas estruturas de dados, sempre buscando elucidar de diferentes formas as dúvidas que surgirão no caminho.

## Objetivos

- Conhecer as ferramentas para análise e projeto de algoritmos;
- Conhecer e identificar algoritmos de Tabelas de Espalhamento - *Hash*;
- Desenvolver algoritmos utilizando os Tabelas de Espalhamento - *Hash*;
- Conhecendo e compreendendo os algoritmos de Busca com Retrocesso.

## Conteúdo programático

**Aula 01** – Tabelas Hash.

**Aula 02** – Busca com Retrocesso.



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

Pode ser necessário instalar um aplicativo de leitura QRcode no

## Aula 01 – Tabelas Hash

### Introdução

O objetivo dessa aula é abordar as Tabelas *Hash*, trazendo o conceito e explicações práticas de como utilizar a lógica desse algoritmo. As explicações serão demonstradas utilizando exemplos didáticos para facilitar a absorção do conteúdo.

Partindo desse contexto, conceituando em ciência da computação, a tabela *hash*, também conhecida como tabela de espalhamento, é uma estrutura de dados que faz a associação de chaves a valores dos seus elementos. O objetivo dessa estrutura de dados, é a realização de buscas de valores dentro da estrutura, a partir de uma chave. A chave, é um tipo de índice que identifica os valores (CORMEN, 2012).

A lógica desse algoritmo também é utilizada para indexação de grandes volumes de informação, como por exemplo, uma base de dados (CORMEN, 2012).

A organização dos dados dentro de uma tabela de banco de dados é realizada também por meio de uma chave. A chave primária, que indica um identificador único do registro dentro da tabela. A Figura 1 exibe a tabela de Aluno referente a um banco de dados.

**Figura 1 – Tabela de Aluno**

Result Grid		Filter Rows:		
	mat	nome	cidade	uf
▶ 1		Macedo	Rio de Janeiro	RJ
2		Ricardo	Rio de Janeiro	RJ
3		Sérgio	Rio de Janeiro	RJ
4		Simone	Rio de Janeiro	RJ
5		Kleber	Londrina	PR
6		Adail	Londrina	PR
7		Bruno	Londrina	PR
	NULL	NULL	NULL	NULL

Fonte: elaborado pelo autor via MySQL Workbench (2021)

Na Figura 1, podemos observar que os campos que compõem a tabela de aluno são: mat, nome, cidade e uf. Entretanto, somente um campo é definido como chave primária. O campo em questão é o **mat**, que representa no caso a matrícula do aluno. A chave primária, ou PK que em inglês significa *Primary Key*, coloca uma regra de inserção de registros. Essa regra, restringe a inserção de duas matrículas com o mesmo valor.

Note que temos por exemplo o campo mat igual a “1” com o nome “Macedo”. Quando tentamos executar uma instrução SQL para inserção de mais um registro com o mesmo código, o banco de dados exibe uma mensagem e não realiza a operação. A instrução em SQL para inserção pode ser visualizada na Figura 2.

**Figura 2 – Instrução insert into (SQL)**

```
85  
86 • INSERT INTO ALUNO VALUES(7, 'Edson', 'Londrina', 'PR');  
87
```

Fonte: elaborado pelo autor via MySQL Workbench (2021)

O valor do campo mat que está sendo inserido é um valor que já pertence a um registro inserido no banco, observe Figura 1. A matrícula 7 identifica o aluno “BRUNO”. Dessa maneira, quando o banco tenta executar o comando um erro acontece. O erro pode ser verificado na Figura 3.

**Figura 3** – Erro de inserção de registro

Action Output				
	Time	Action	Response	Duration / Fetch Time
✖ 1	21:57:00	INSERT INTO ALUNO VALUES(7,'Edson','Londrina','PR')	Error Code: 1062. Duplicate entry '7' for key 'PRIMARY'	0.0019 sec

Fonte: elaborado pelo autor via MySQL Workbench (2021)

A mensagem apresentada como resposta a tentativa de execução do comando é: “*Error Code: 1062. Duplicate entry '7' for key 'PRIMARY'*”, que significa que existe uma entrada duplicada de valor “7” para a chave primária. Diante disso, para que o usuário técnico consiga inserir um novo registro, ele deve alterar o campo “mat” para um valor que não exista na tabela de aluno. Sendo assim, o registro será inserido normalmente na tabela.

## Realidade Profissional

Vamos contextualizar o exemplo de algoritmos de Tabela *Hash* por meio do exemplo de emergência hospitalar, brevemente comentado na Unidade 3 no conteúdo de Filas Prioritárias. No exemplo, trabalhamos com a classificação de risco pelo Protocolo de Manchester, figura 4.

**Figura 4** – Protocolo de Manchester

EMERGÊNCIA	Emergência: Caso gravíssimo, com necessidade de atendimento imediato e risco de morte.
MUITA URGÊNCIA	Muito urgente: Caso grave e risco significativo de evoluir para morte. Atendimento urgente.
URGÊNCIA	Urgente: Caso de gravidade moderada, necessidade de atendimento médico, sem risco imediato.
POUCA URGÊNCIA	Pouco Urgente: Caso para atendimento preferencial nas unidades de atenção básica.
NÃO URGÊNCIA	Não Urgente: Caso para atendimento na unidade de saúde mais próxima da residência. Atendimento de acordo com o horário de chegada ou serão direcionados às Estratégias de Saúde da Família ou Unidades Básicas de Saúde. Queixas crônicas; resfriados; contusões; escoriações; dor de garganta; ferimentos que não requerem fechamento e outros.

Fonte: Hospital Santa Cruz (2021)

A classificação quanto à prioridade do atendimento é realizada de acordo com as categorias presentes na Figura 4, porém mesmo assim, o atendimento precisa de uma organização a respeito da especialidade do atendimento. Alguns pacientes precisam ser direcionados para um ortopedista, outras para um cardiologista, pois os pacientes procuram o atendimento emergencial em um pronto socorro por diversos motivos.

Nesse contexto, os algoritmos de espalhamento ou tabela hash são efetivos, pois eles podem ajudar a distribuição desses pacientes que no caso são representados como dados para a tabela de espalhamento, sendo divididos pelas suas chaves.













Como se sabe, no momento da chegada do paciente ao hospital, é realizado o processo de triagem. Sem essa organização, o processo ficaria lento e ineficaz. Isso provocaria descontentamento e irritação dos pacientes.

Objetivando a organização nos atendimentos, vamos considerar que ele seja realizado por um enfermeiro ou um médico plantonista. Em nosso exemplo, vamos pensar

em vários pacientes com vários “problemas diferentes”, cada problema representa um índice

De acordo com esse contexto, vamos abordar algumas especialidades médicas apresentadas por meio do Quadro 1.

**Quadro 1** – Divisão de pacientes por problemas apresentados

Ortopedista	Cardiologista	Oftalmologista
 P1	 P2	 P3
 P4	 P5	 P6
 P7	 P8	 P9
 P10	 P11	 P12

Fonte: elaborada pelo autor (2021)

Como pode-se observar os pacientes estão identificados por P1, P2, que significa Paciente 1, Paciente 2 e assim sucessivamente. O Quadro 1 também mostra que os pacientes são organizados em novas filas por especialidade médica.

Seria muito trabalhoso os médicos irem em busca dos pacientes, portanto eles são organizados por índices, de acordo com as tabelas *hash*. Isso agiliza muito o atendimento.

Após o processo de triagem, temos todos os pacientes direcionados aos especialistas de acordo com o seu tipo de problema. Dessa forma, a tabela *hash* vem resolver essa situação apresentada, trazendo uma melhor organização e agilidade para esse cenário de atendimento hospitalar.



## Indicação de Leitura

O livro de Cormen “Algoritmos Teoria e Prática”, apresenta uma boa base dos diferentes tipos de estruturas existentes no cenário de Algoritmos. Dentre esses conteúdos, o capítulo 11 aborda sobre Tabelas de Espalhamento e explica os diferentes tipos e variações dessa estrutura de dados. A seção 11.1 aborda sobre Tabelas de Endereço Direto e a seção 11.2 traz o conteúdo de Tabelas de Espalhamento. Muitas outras estruturas, são abordadas neste livro e trazem características e exemplos de aplicação desses algoritmos. Vale a pena conferir!

Disponível em: <https://docero.com.br/doc/nxxnx8v>. Acesso em: 30 jul. 2021.

CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. Rio de Janeiro: Campus, 2012. 926 p.

Para melhor entendimento sobre o conceito das tabelas de espalhamento ou tabelas *hash*, segue uma indicação de vídeo com algumas explicações e detalhes de como funciona na prática esse algoritmo.

## Indicação de Vídeo

Confira o vídeo recomendado que aborda sobre as Tabelas *Hash*. No vídeo, é explicado alguns conceitos sobre o conteúdo e apresenta o exemplo de alocação das pessoas em suas filas corretas após a identificação por chaves.

Você pode assistir o vídeo quantas vezes quiser para entender o conceito. Vale a pena conferir.

Disponível em: [https://www.youtube.com/watch?v=Non0l\\_OSt9o](https://www.youtube.com/watch?v=Non0l_OSt9o). Acesso em: 30 jul. 2021.

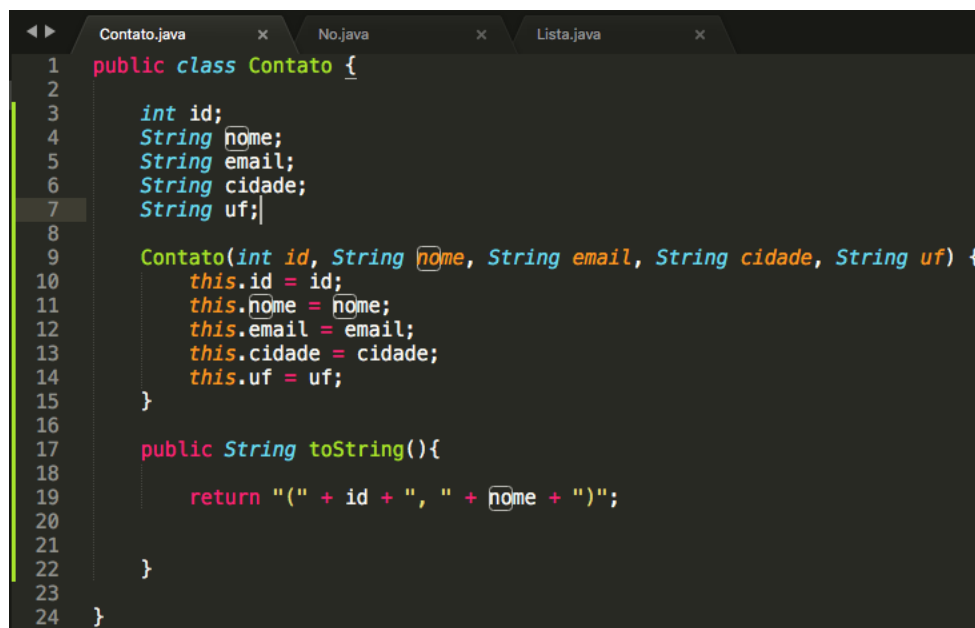


## Exemplo Prático

Vamos ao nosso exemplo prático. Vamos gerenciar um tipo específico de dado que será definido pelo usuário. Vamos gerenciar a inclusão de contatos inseridos dentro de uma tabela *hash*, um exemplo que é bastante trabalhado com tabelas de espalhamento. O nosso exemplo terá 4 classes. A classe Contato, a classe No, a Classe Lista e a Classe Hash.

Para a classe Contato, podemos definir alguns atributos como: id, nome, *e-mail*, cidade, e uf. Utilizaremos nesse exemplo o editor de texto Sublime versão 3.2.2, Observe a Figura 5.

Figura 5 – Classe Contato.java

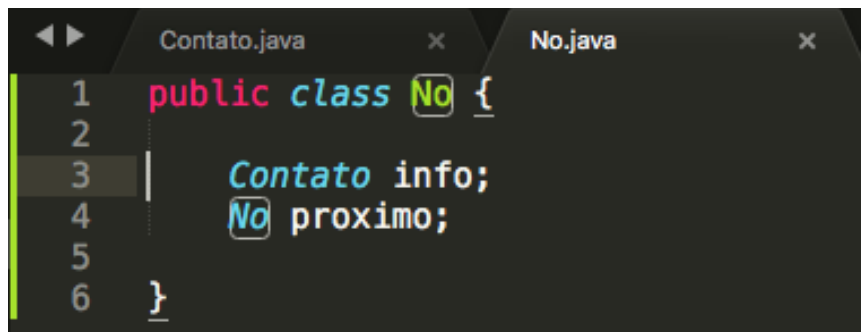


```
1 public class Contato {
2
3     int id;
4     String nome;
5     String email;
6     String cidade;
7     String uf;
8
9     Contato(int id, String nome, String email, String cidade, String uf) {
10         this.id = id;
11         this.nome = nome;
12         this.email = email;
13         this.cidade = cidade;
14         this.uf = uf;
15     }
16
17     public String toString(){
18         return "(" + id + ", " + nome + ")";
19     }
20
21 }
22
23
24 }
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

A Figura 5, cria a classe Contato com os atributos descritos e o método construtor fazendo a iniciação de valores no momento de sua instância.

**Figura 6 – Classe No.java**



```
1 public class No {  
2  
3     Contato info;  
4     No proximo;  
5  
6 }
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

A classe No, apresentada na Figura 6, faz a criação do atributo da classe Contato e outro do tipo No.

### Videoaula 1

Acompanhe à videoaula, ela vai apresentar a criação das duas classes apresentadas até aqui. A classe Contato e a classe No. O professor vai explicar o código na prática passo a passo para demonstrar todo o processo.

Assista esse vídeo para compreender a ideia da presença dessas duas classes no projeto.



#### Videoaula 1

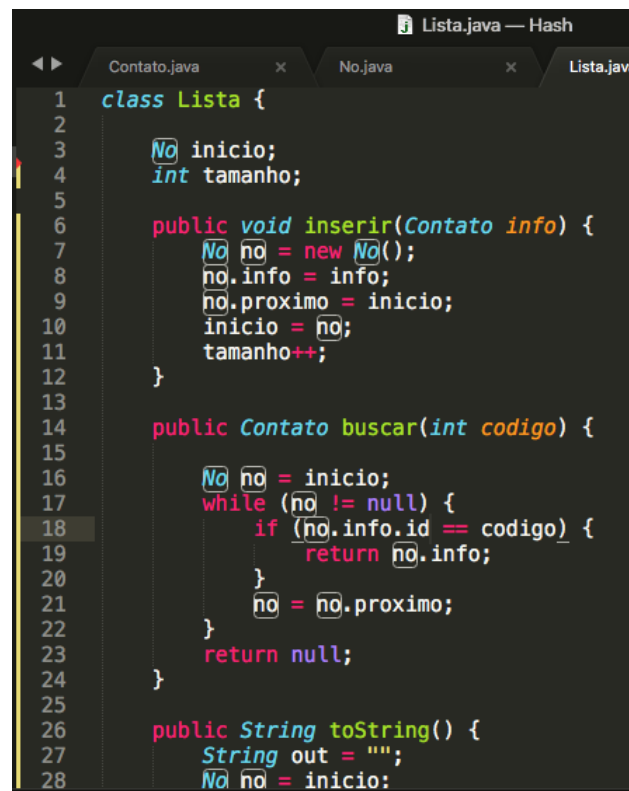
Utilize o QRcode para assistir!

Acompanhe à videoaula, ela vai apresentar a criação das duas classes apresentadas até aqui.



Vamos agora apresentar o restante da estrutura do nosso exemplo. Serão criadas mais duas classes. A classe *Lista* e a classe *Hash*. A classe *Lista* possui uma lista encadeada. Essa classe contém o método *inserir*, responsável por inserir elementos, um método para buscar elementos baseados no parâmetro código e outro método para mostrar elementos da lista. Observe a Figura 7.

Figura 7 – Classe *Lista.java*

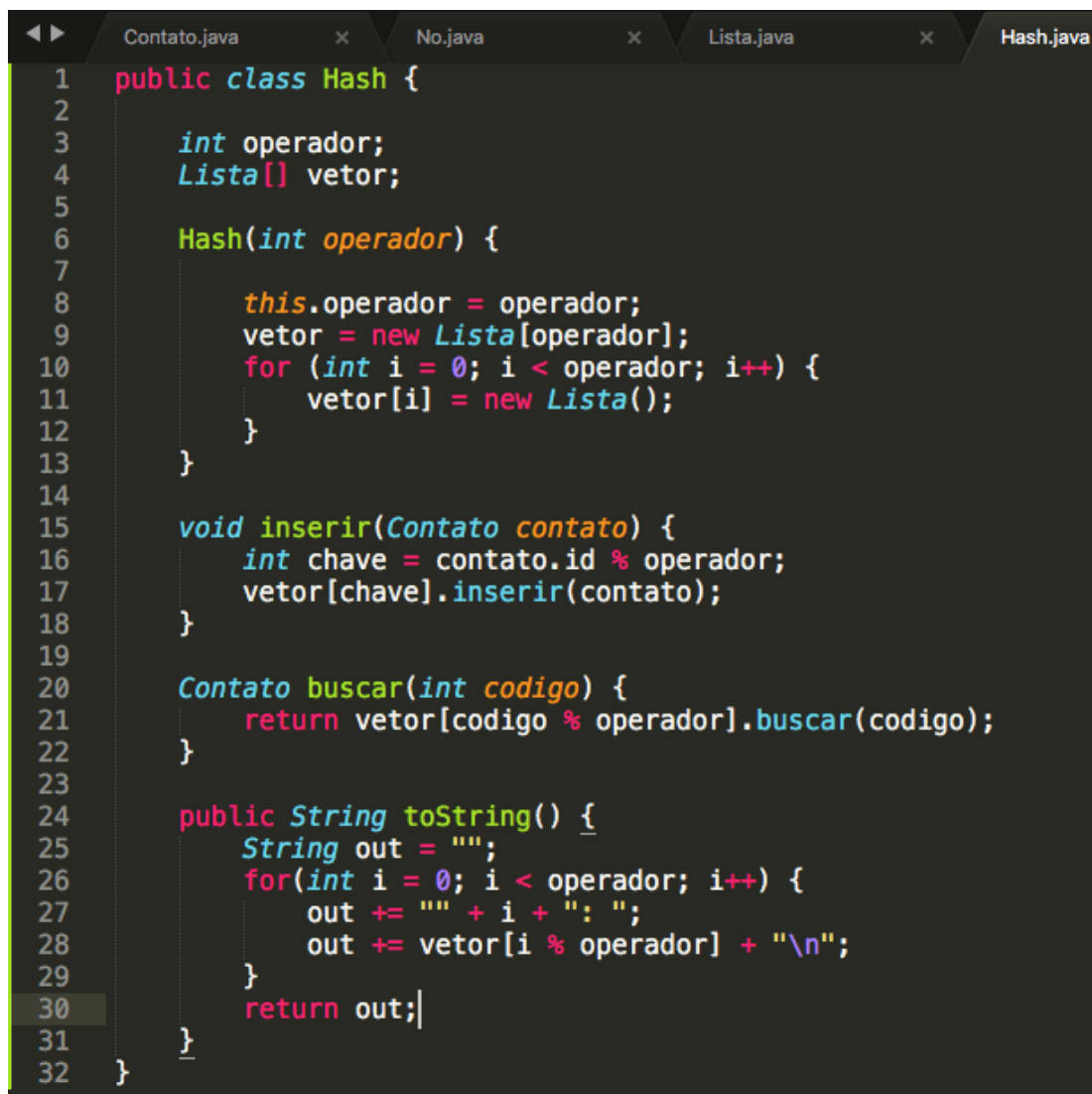


```
1 class Lista {
2
3     No inicio;
4     int tamanho;
5
6     public void inserir(Contato info) {
7         No no = new No();
8         no.info = info;
9         no.proximo = inicio;
10        inicio = no;
11        tamanho++;
12    }
13
14    public Contato buscar(int codigo) {
15
16        No no = inicio;
17        while (no != null) {
18            if (no.info.id == codigo) {
19                return no.info;
20            }
21            no = no.proximo;
22        }
23        return null;
24    }
25
26    public String toString() {
27        String out = "";
28        No no = inicio;
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

A Figura 8 apresenta o código da classe *Hash*. Nessa classe é implementada a Tabela *Hash*. Dentro da classe é criado um *Array* referente à classe *Lista*. Na classe de testes, será criado um vetor conforme o valor do operador. Com base no resto da divisão será definido a posição do vetor que será inserir o contato.

Figura 8 – Classe Hash.java



```
1 public class Hash {
2
3     int operador;
4     Lista[] vetor;
5
6     Hash(int operador) {
7
8         this.operador = operador;
9         vetor = new Lista[operador];
10        for (int i = 0; i < operador; i++) {
11            vetor[i] = new Lista();
12        }
13    }
14
15    void inserir(Contato contato) {
16        int chave = contato.id % operador;
17        vetor[chave].inserir(contato);
18    }
19
20    Contato buscar(int codigo) {
21        return vetor[codigo % operador].buscar(codigo);
22    }
23
24    public String toString() {
25        String out = "";
26        for(int i = 0; i < operador; i++) {
27            out += "" + i + ": ";
28            out += vetor[i % operador] + "\n";
29        }
30        return out;
31    }
32 }
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

Veja o vídeo para ver a criação dessas duas classes.

## Videoaula 2

A próxima videoaula, vai abordar a construção da classe Lista e da classe Java. O professor vai fazer na prática a construção das classes, por meio da linguagem de programação Java.

Assista o vídeo se possível replicando o exemplo em seu ambiente de desenvolvimento visando compreender o processo da construção das classes. Lembrando que para este exemplo está sendo trabalhado com o editor de texto Sublime versão 3.2.2.



### Videoaula 2

Utilize o QRcode para assistir!

A próxima videoaula, vai abordar a construção da classe Lista e da classe Java.



Para finalizar, vamos demonstrar a criação da classe de teste e fazer a execução do exemplo mostrando a impressão dos contatos na tabela *hash*.

## Videoaula 3

A próxima videoaula, vai abordar a criação da classe principal do projeto, realizando as instâncias necessárias e adicionando aleatoriamente alguns Contatos em nossa lista.

Durante a criação da classe, o professor vai abordar o exemplo e a inserção dos valores para teste. Acompanhe a compilação execução das classes do programa.

A exemplo das outras videoaulas, assista e replique os códigos para o seu ambiente para maximizar a fixação do conteúdo abordado.



### **Videoaula 3**

Utilize o QRcode para assistir!

A próxima videoaula, vai abordar a criação da classe principal do projeto, realizando as instâncias necessárias e adicionando aleatoriamente alguns Contatos em nossa lista.



## Aula 02 – Busca com Retrocesso

### Introdução

Nessa aula, o tema abordado será a técnica de busca por retrocesso. Antes de começarmos a falar sobre essa técnica cabe aqui uma pequena revisão sobre os algoritmos de busca em geral. O que chamamos de *OverView*.

A busca é uma rotina que realizamos todos os dias. Fazemos pesquisas na internet em busca de promoções ou até mesmo o endereço que precisamos estar presencialmente mais tarde.

Os tipos de busca mais conhecidas e utilizadas são: a SEQUENCIAL E BINÁRIA. Esses dois tipos de busca são os mais utilizados na resolução dos problemas de busca em algoritmos computacionais.

Na disciplina de Algoritmos e Estrutura de Dados, foram abordados esses dois tipos de algoritmos de busca, sendo abordados exemplos práticos de aplicação, bem como, a construção de exemplos codificados na linguagem de programação Java.

Em estrutura de dados, a 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. Em sua grande maioria, os algoritmos estudados por cientistas da computação que resolvem problemas são algoritmos de busca (DEITEL, 2010).

Na sequência, uma indicação de leitura com conteúdos sobre esse tipo de algoritmos.

#### Indicação de Leitura

Para melhor compreensão deste tópico, você pode estudar o livro de CORMEN. O capítulo 3 “Algoritmos para ordenar e buscar”, que apresenta a estrutura do algoritmo de busca binária. O capítulo aborda o conceito com exemplos e codificação. Vale a pena conferir.

CORMEN, Thomas H. **Desmistificando algoritmos**. Rio de Janeiro: Campus, 2012. 926 p.



## Backtracking

*Backtracking* é o termo em inglês para retrocesso. É importante você se familiarizar com o termo, pois se fizer pesquisas sobre o conteúdo, pode ser que ache artigos e trabalhos pelo termo em inglês e não pelo português.

Resumindo, *backtracking* ou retrocesso é uma técnica de programação que é implementada por meio da recursão. Essa técnica pode ser aplicada em problemas que aceitem candidatos para soluções parciais, isso é, quando não se chegou a solução correta e precisa de ajustes. Esse comportamento é uma das características desse tipo de algoritmos. Por exemplo: digamos que você tenha que colocar um elemento na posição (l,c) qualquer de uma matriz. Lembrando o l = representa a linha e o c = representa a coluna. O algoritmo de retrocesso vai testar as possibilidades para (l,c) que foram passadas como coordenadas. Desse modo, até que não se chegue na posição (l,c) desejada, o algoritmo vai fazendo alterações.

Outro fundamento da busca com retrocesso, é que ela não serve para problemas como ordenação de dados.

### Exemplo: Problema das 8 Rainhas

Para a explicação lógica da técnica de retrocesso, vamos demonstrar o exemplo das 8 rainhas em um tabuleiro de Xadrez. Para quem não sabe jogar Xadrez, esse jogo é um jogo de tabuleiro que contém 64 casas e 16 peças para cada jogador. São 2 jogadores que disputam a partida. Um fica com as peças pretas e outro com as peças brancas. Vence aquele que derruba o Rei do oponente. Essa jogada é conhecida como Xeque Mate.

Dentro do jogo temos peças como peão, cavalo, bispo, torre, rainha e o rei. São 8 peões, 2 cavalos, 2 bispos, 2 torres, 1 rainha e 1 rei para cada jogador, sendo que para cada tipo de peça possui um movimento característico. O jogador que está com as peças brancas começa o jogo.

A Rainha é uma peça chave no jogo de xadrez. É possível realizar várias jogadas com seus movimentos.

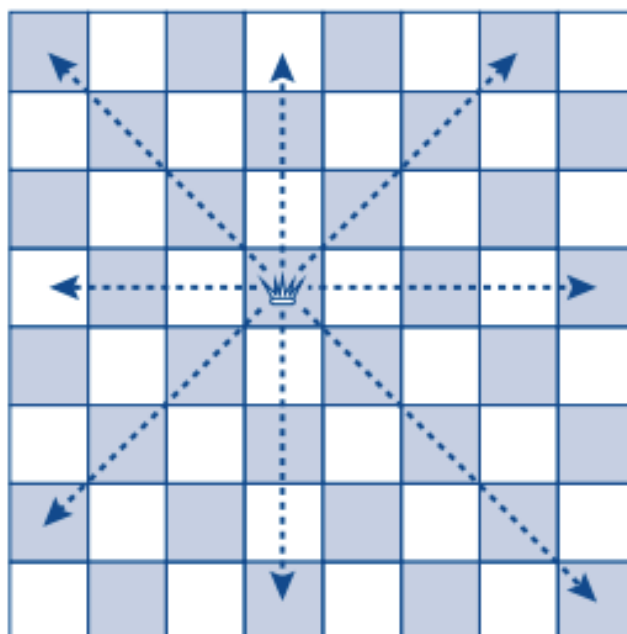
**Figura 1 – A Peça Rainha**



Fonte: pixabay (2021)

A Figura 1, exibe a peça Rainha. A questão dos movimentos de cada peça do tabuleiro, faz com que a Rainha seja a peça mais poderosa do Xadrez, pois ela pode atacar qualquer peça que esteja na mesma coluna, linha ou diagonal na qual ela está posicionada. Abaixo, a Figura 2 apresenta os seus movimentos.

**Figura 2 – Movimentos da Rainha**



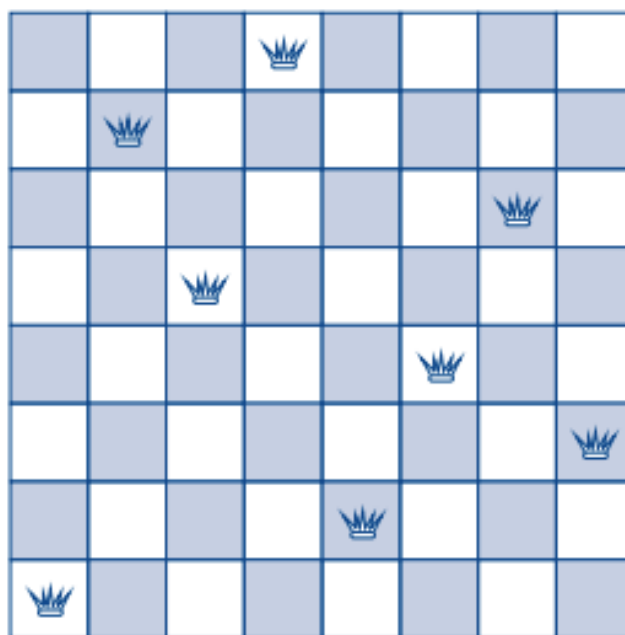
Fonte: adaptado de Oliveira (2019)

Pode-se perceber porque essa peça é a preferida entre os jogadores de Xadrez. Veja na Figura 2 o tamanho da área do tabuleiro que ela consegue cobrir.

Dentro desse contexto, vamos a explicação do problema das 8 rainhas. A resolução do problema das 8 rainhas, tem por objetivo colocar 8 rainhas em um tabuleiro de xadrez, posicionadas estrategicamente para que cada uma das 8 rainhas NÃO consiga atacar nenhuma outra. Sendo assim, a regra restringe que nenhum par de rainhas ocupe a mesma linha, coluna ou diagonal.

A Figura 3, apresenta uma solução visual do tabuleiro com as 8 rainhas corretamente posicionadas, uma em cada coluna e sem que estejam sob ataque nessa configuração.

**Figura 3** – Resolução do algoritmo



Fonte: adaptado de Oliveira (2019)

Como pode-se verificar na Figura 2, realmente nenhuma rainha consegue atacar uma outra, bem como, nenhuma está sob ataque. Foram identificadas 12 soluções diferentes para este tipo de problema dentro do tabuleiro.

Parece simples configurar o tabuleiro para que essa disposição seja criada, mas dentro dos algoritmos a solução não é tão trivial assim. Dessa maneira, trabalhamos com os algoritmos de retrocesso para resolver esse problema.

A ideia do algoritmo é a seguinte: enquanto não estiverem inseridas 8 rainhas no tabuleiro realizar as instruções a seguir:

1. Se na linha posterior existir alguma coluna que não esteja sob ataque de nenhuma rainha que já esteja no tabuleiro, inserir uma rainha nessa posição.
2. A condição acima sendo FALSA, ou seja, senão conseguir cumprir a instrução 1 com sucesso, voltar a linha anterior (*backtrack*);
3. Mover a rainha o mínimo de casas necessário sempre para a direita até que ela esteja livre de qualquer ataque.

Vamos acompanhar as próximas Figuras para identificar a forma de resolução apresentada.

**Figura 4** – Teste de Mesa da Execução do Algoritmo – parte 1

<b>R</b>							

Fonte: elaborado pelo autor (2021)

A Figura 4, apresenta a inserção da 1ª rainha na linha um e coluna um do tabuleiro. O algoritmo segue, respeitando as instruções 1, 2, 3 apresentadas acima, inserindo outras rainhas no mesmo tabuleiro, veja as próximas Figuras.

**Figura 5** – Teste de Mesa da Execução do Algoritmo – parte 2

<b>R</b>							
		<b>R</b>					

Fonte: elaborado pelo autor (2021)

Observe mais uma inserção para uma rainha na 3ª linha.

**Figura 6** – Teste de Mesa da Execução do Algoritmo – parte 3

<b>R</b>							
		<b>R</b>					
				<b>R</b>			

Fonte: elaborado pelo autor (2021)

Esse processo é repetido várias vezes até que todas as 8 rainhas sejam inseridas no mesmo tabuleiro, como mostra a Figura 6.

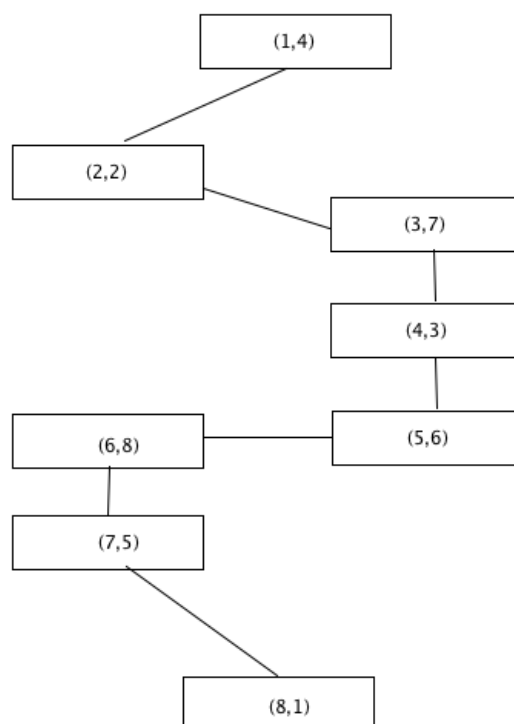
**Figura 7** – Teste de Mesa da Execução do Algoritmo – parte 4

			R				
	R						
						R	
		R					
					R		
							R
				R			
R							

Fonte: elaborado pelo autor (2021)

O esquema gráfico apresentado pela construção do tabuleiro na representação de um fluxograma pode ser observado na Figura 8.

**Figura 8** – Fluxograma do tabuleiro com as 8 rainhas



Fonte: elaborado pelo autor (2021)

Embora o tabuleiro de Xadrez tenha somente 64 casas, ou 8 linhas e 8 colunas, essa solução pode ser estendida para tabuleiros com N linhas e N colunas, porém para nossa explicação, vamos trabalhar com um pseudo tabuleiro de 4 x 4.

Vamos para a construção dessa estrutura algorítmica nessa primeira videoaula, demonstrando assim todos os passos para a conclusão desse processo.

## Videoaula 1

Nessa videoaula, será abordado o exemplo para a criação da estrutura do tabuleiro de Xadrez, trabalhando com um pseudo tabuleiro de 4 linhas e 4 colunas como explicado anteriormente nessa Aula. O professor vai implementar na prática a criação das estruturas de dados.

Acompanhe com atenção e não deixe de realizar no seu ambiente de desenvolvimento, todas as implementações, alterações no código e seguir as orientações de execução do algoritmo.



### Videoaula 1

Utilize o QRcode para assistir!

Nessa videoaula, será abordado o exemplo para a criação da estrutura do tabuleiro de Xadrez, trabalhando com um pseudo tabuleiro de 4 linhas e 4 colunas como explicado anteriormente nessa Aula.





## Videoaula 2

Nessa segunda videoaula, será abordada a construção do método que vai fazer o controle da inserção dos elementos, ou seja, das peças de xadrez dentro do tabuleiro. O professor vai implementar os códigos passo a passo e explicar a execução do algoritmo.

Acompanhe com atenção e não deixe de realizar esses exemplos no seu ambiente de desenvolvimento. Lembrando que você pode utilizar qualquer ambiente que permita o desenvolvimento na linguagem de Programação Java.

O teste de mesa de execução para esse método continua sendo imprescindível para que você compreenda em detalhes tudo o que está acontecendo no algoritmo.



### Videoaula 2

Utilize o QRcode para assistir!

Nessa segunda videoaula, será abordada a construção do método que vai fazer o controle da inserção dos elementos, ou seja, das peças de xadrez dentro do tabuleiro.



## Execução do Algoritmo

Para finalização do nosso exemplo, é executado todo o código para a exibição do tabuleiro 4 x 4, contendo a disposição dos seus elementos.

### Videoaula 3

Nessa terceira videoaula, será abordada a construção do método principal. Esse método efetivamente vai controlar a impressão das soluções encontradas para o problema das 8 rainhas. O professor vai construir a codificação para resolver o caso.

Acompanhe com atenção e não deixe de realizar esses exemplos no seu ambiente de desenvolvimento.



#### Videoaula 3

Utilize o QRcode para assistir!

Nessa terceira videoaula, será abordada a construção do método principal.



### Encerramento da Unidade

Chegamos ao fim desta Unidade de Ensino, após caminhar por conteúdos importantes para o seu desenvolvimento acadêmico até aqui. A Unidade 4 apresentou o conteúdo das tabelas *hash*, ou tabelas de espalhamento, trabalhando com um cenário real para distribuição dos elementos da tabela. Também foi abordado o algoritmo de busca com retrocesso, demonstrando a inserção de peças de xadrez no tabuleiro de acordo com o exemplo das 8 rainhas.

Após a assimilação dos conteúdos abordados nesta Unidade, você estará apto a prosseguir estudando um pouco mais sobre os conceitos e conteúdos dessa disciplina.

Tenha uma ótima leitura e não se esqueça de ler e estudar a Bibliografia e as vídeo aulas ministradas 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.



### **Videoaula Encerramento**

Utilize o QRcode para assistir!



## Referências

- CORMEN, T. H. **Desmistificando algoritmos**. Rio de Janeiro: Campus, 2012. 926 p.
- CORMEN, T. H. *et al.* **Algoritmos: teoria e prática**. Rio de Janeiro: Campus, 2012. 926 p.
- DEITEL, H. M.; DEITEL, P. J. **Java: como programar**. 8. ed. São Paulo: Pearson, 2010.
- FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**. 2. ed. São Paulo: Makron Books, 2000. 195 p.
- JANDL JUNIOR, P. **Java: guia do programador**. São Paulo: Novatec, 2007. 681 p.
- LAUREANO, M. **Estrutura de dados com algoritmos e C**. São Paulo: Brasport, 2012.
- OLIVEIRA, U. **Estrutura de Dados usando a linguagem C Volume 1: Fundamentos**. Ulysses de Oliveira, 2019. 644 p.
- PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados: com aplicações em Java**. São Paulo: Pearson, 2016.
- SANTANA, A. L. **Técnicas de Programação**: Curso Técnico em Informática. Colatina: IFES, 2011.



UNIFIL.BR