

ALGORITMOS E ESTRUTURA DE DADOS

Unidade 1

Tipos Abstratos De Dados E Listas Ligadas

Aula 1

Tipos Abstratos de Dados (TAD)

Tipos abstratos de dados (TAD)



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe boas-vindas a esta aula e esperamos que este conteúdo tenha valor agregado na construção do seu conhecimento.

A partir deste momento, vamos conhecer um pouco sobre a área de algoritmos que aprofunda a maneira como eles são desenvolvidos, na medida em que os problemas resolvidos por soluções algorítmicas também se aprofundam. Estamos falando das estruturas de dados.

No decorrer da disciplina, serão apresentadas diversas estruturas de dados que podem ser utilizadas no desenvolvimento de uma aplicação ou resolução de um problema pontual. Para tanto, abordaremos nesta aula os tipos abstratos de dados, conhecidos também pela sigla TAD.

ALGORITMOS E ESTRUTURA DE DADOS

Atualmente, a automação de tarefas está presente no cotidiano das pessoas e dentro das grandes empresas. Seja com o auxílio da inteligência artificial ou não, existe um apelo consumista em produtos e dispositivos que trazem conforto, comodidade e agilidade para todos os usuários adeptos a estas tecnologias. Apresentaremos, por meio dos TADs, como isso pode ser perfeitamente implementado em uma linguagem de programação com o intuito de tirar essa ideia do papel e colocá-la em prática.

Portanto, vamos abordar um exemplo prático que apresenta uma demanda para um sistema de controle gerencial. Imagine que você trabalha em uma empresa de varejo que precisa de um sistema para gerir o seu estoque de produtos. Cada produto possui informações como código, descrição, quantidade disponível e preço por unidade. Este sistema deve permitir ações como: incluir novos produtos, excluir produtos, atualizar a quantidade de produtos disponíveis e calcular o valor total do estoque. Dessa forma, a empresa poderá gerenciar seu estoque de maneira eficiente.

Convidamos você a conhecer essa estrutura de dado e entender como ela pode auxiliar na sua atividade como um desenvolvedor de sistemas.

Vamos Começar!

O objetivo desta aula consiste em discutir os conceitos de tipos abstratos de dados, oferecendo uma definição e demonstrações práticas sobre como aplicar a lógica dos TADs no algoritmo que está sendo desenvolvido. Para esclarecer o conteúdo, utilizaremos exemplos didáticos, tornando a compreensão mais acessível.

A essência desse algoritmo se encontra na minimização das informações necessárias para a construção do foco principal do programa. É importante destacar que, em muitos casos, uma linguagem de programação possui um arquivo ou classe central encarregada de executar as outras sub-rotinas. Uma linguagem de programação dentro do paradigma orientado é o Java, que trabalha seguindo esse modelo.

Considerando um objeto “cliente” como exemplo, sem que haja a utilização de uma estrutura algorítmica TAD, o cliente teria que ser controlado por meio do algoritmo através de variáveis soltas dentro do código (“id”, “nome”, “cidade” e “uf”). Essas variáveis seriam manipuladas de forma isolada e sem uma associação por um tipo ou classe qualquer.

No sentido conceitual, adotando a abordagem de um tipo abstrato de dado, o programa foi concebido de maneira conjunta, resultando na eliminação da necessidade de campos individuais como “id”, “nome”, “cidade” e “uf”. Em vez disso, foi introduzido o tipo “cliente”, que engloba todos os atributos relacionados ao cliente. Ele é definido como uma classe, na qual cada atributo é atribuído a um tipo específico, funcionando como variáveis comuns, porém associadas ao tipo “cliente”.

ALGORITMOS E ESTRUTURA DE DADOS

De acordo com a literatura da área, uma classe é dividida em três partes: seu nome, atributos e operações. Logo, o tipo “cliente” também incorpora operações que são implementadas dentro da classe.

Definição de TAD

Para iniciarmos a definição dessa estrutura de dado, o Quadro 1 a seguir apresenta uma comparação entre as abordagens de algoritmo. Enquanto na coluna “Tipo Comum” os dados são gerenciados por meio de variáveis individuais, na coluna “Tipo Cliente”, opta-se por utilizar um tipo abstrato de dados para agrupar as informações dentro do tipo “cliente”.

Número	Tipo Comum	Tipo Cliente
1	Id	cliente: { int id }
2	Nome	nome: {String nome }
3	E-mail	email: {String email}
4	Endereço	endereco: { String endereco}
5	Cidade	cidade: { String cidade}
6	UF	uf: { String uf}
7	Operações dentro do programa sem ligação direta.	Operações que vinculam os atributos com os métodos do tipo Cliente.

Quadro 1 | Comparativo entre as formas de abordagem.

A utilização de modelos por meio do TAD resultou em uma compreensão aprimorada dos algoritmos. No entanto, essa abordagem também introduziu um aumento na sofisticação das soluções, tornando, assim, imprescindível a criação de uma estrutura de modelagem de dados em qualquer sistema computacional.

Essa estrutura de modelagem de dados nos direciona ao paradigma da orientação a objetos. Dessa forma, a programação orientada a objetos, também conhecida como *object oriented programming* (OOP), desempenha um papel fundamental na progressão dos tipos abstratos de

ALGORITMOS E ESTRUTURA DE DADOS

dados, permitindo que esses algoritmos sejam desenvolvidos e controlados de acordo com as políticas estabelecidas para a criação e implementação de classes e outros objetos dentro desse paradigma.

A proposta subjacente aos tipos abstratos de dados é que os tipos de dados primitivos ou simples usados para representar e manipular informações mais complexas são, na verdade, insuficientes. Portanto, é necessário a utilização de novos tipos de dados capazes de atender às demandas algorítmicas dos modernos modelos de desenvolvimento de sistemas computacionais (Cormen, 2013).

Como criar um TAD

Antes de iniciarmos a etapa prática, é relevante que contextualizemos o cenário de aplicação deste tipo abstrato de dado.

A automação residencial é uma área que encanta os adeptos desse setor, o qual avança em produtos e serviços oferecidos. A automatização de diversos eletrodomésticos, bem como da iluminação e de portões eletrônicos, está presente nas residências de inúmeras pessoas ao redor do mundo. A conveniência de ligar ou desligar remotamente um aparelho através de um aplicativo ou por meio de comandos de voz, utilizando assistentes como a Alexa, atrai consideravelmente aqueles que apreciam esse tipo de tecnologia.

Além dos aspectos relacionados ao conforto e ao entretenimento, algumas automatizações assumem um papel fundamental quando se trata de segurança. Por exemplo, existem sistemas de automação residencial programados para prevenir acidentes decorrentes de vazamentos de gás. Mesmo que os moradores não estejam em casa, um vazamento de gás pode resultar em um incidente grave. Assim, o sistema assume a responsabilidade de interromper o fornecimento de gás, abrir as janelas para ventilação e ainda avisar o proprietário por meio de notificações em seu smartphone.

Além disso, no contexto do armazenamento de informações, é viável implementar um sistema computacional que registre o histórico dos eventos ocorridos na residência. Com esses dados disponíveis em um banco de dados, torna-se possível criar relatórios com indicadores e gráficos que apresentam percentuais de irregularidades por dispositivos, entre outras diversas possibilidades.

Para criar uma TAD, vamos propor a automatização de um eletrodoméstico utilizado rotineiramente pela maioria das pessoas em sua residência. Estamos falando da televisão, ou TV, como é conhecida popularmente. Os comandos disponíveis em um controle remoto, como a alteração de canais, ajuste do volume ou a escolha de um aplicativo específico, como um serviço de streaming, podem ser automatizados por sistemas computacionais. Diante deste contexto, iniciaremos destacando os atributos do dispositivo que pretendemos definir em nosso exemplo. O Quadro 2 ilustra esses atributos.

ALGORITMOS E ESTRUTURA DE DADOS

Número	Tipo de dado	Nome do atributo
1	string	marca
2	boolean	ligado
3	int	volume
4	int	canal
5	string	local

Quadro 2 | Atributos que serão controlados.

O Quadro 2 apresenta os atributos da TV que serão controlados em nosso exemplo. O exemplo será criado por meio da linguagem de programação Java, no editor de códigos Sublime, ou seja, não necessita de nenhuma ferramenta para desenvolvimento, basta ter o Java Development Kit (JDK) instalado. O JDK é o pacote de desenvolvimento Java; trata-se de um conjunto de utilitários que permitem criar sistemas de software para a plataforma Java. Na sequência, a classe é apresentada na Figura 1.



```
1 public class Televisao{  
2  
3     String marca;  
4     boolean ligado;  
5     int volume;  
6     int canal;  
7     String local;  
8  
9     public Televisao(){  
10        ligado = false;  
11    }  
12  
13}
```

Figura 1 | Classe Televisao.java.

ALGORITMOS E ESTRUTURA DE DADOS

A classe “Televisao” será encarregada de gerenciar as operações que serão executadas pelo dispositivo eletrônico, a TV.

A partir da classe de operações devidamente criada, com todos os seus atributos e métodos, a construção de cada método deve ser feita. Portanto, a próxima etapa da construção do nosso exemplo se encarrega da construção de alguns métodos de controle para a nossa TV automatizada.

Siga em Frente...

Exemplos de utilização de TADs

Os procedimentos que serão gerenciados no âmbito do programa incluem a mudança de canal e o ajuste do volume da televisão. O método para a mudança de canal será denominado como “defineCanal()” e solicitará o canal como um parâmetro. Com base nesse valor, o programa verificará se a televisão está ligada e, em seguida, atualizará o canal para o valor fornecido como parâmetro.

```

56     public void defineCanal(int pcanal){
57
58         if(ligado){
59
60             canal = pcanal;
61             System.out.println("Marca: " + marca + ": canal definido para "
62
63         }
64         else{
65             System.out.println("Ligue a TV " + marca);
66         }
67     }
68 }
```

Figura 2 | Função que define um novo canal.

A Figura 2 exibe a construção da função “defineCanal()”. O atributo “ligado” que é do tipo lógico, ou seja, verdadeiro ou falso. Quando a TV é ligada, ele recebe o valor verdadeiro; desta forma, o código faz uma verificação antes de “setar” o canal para o atributo.

O método responsável por aumentar o volume será designado como “aumentarVolume()”. Este método não requer parâmetros e é do tipo “void”. A verificação da condição da TV é realizada da mesma maneira que no método “defineCanal()”. Após essa verificação, o método aumenta o volume da TV em incrementos de passo 1. Isso significa que, a cada vez que o comando é executado, o volume da TV é aumentado em uma unidade. Por exemplo, se o volume atual estiver em 15, após a execução do método “aumentarVolume()”, ele passará a 16. Logo a seguir, a Figura 3, exibe a construção da função “aumentarVolume()”.

ALGORITMOS E ESTRUTURA DE DADOS

```
44     public void aumentarVolume(){  
45  
46         if(ligado){  
47             volume++;  
48             System.out.println("Marca: " + marca + ": volume definido para: "  
49         }  
50         else{  
51             System.out.println("Ligue a TV " + marca);  
52         }  
53     }  
54 }
```

Figura 3 | Função que define um novo volume.

Conforme a sua criatividade, outras funcionalidades podem ser implementadas no projeto, por exemplo, outra função para diminuir o volume da TV.

Para terminar a criação do nosso exemplo prático de automação, falta apenas a criação de uma classe para execução dos métodos que foram construídos na classe Televisao.java.

Uma informação importante: se você quiser reproduzir este exemplo, crie a próxima classe no mesmo diretório a fim de concentrar em só local os arquivos compilados, ou seja, os arquivos com extensão “.class”.

A classe criada receberá o nome de UtilizarTV.java; dentro dela, como padrão, será criado o método principal conhecido como main(). Na sequência, dentro do método principal, serão instanciados dois aparelhos de TV, representando que podem ser instanciados mais aparelhos se necessário. Realizada a instância, os métodos criados na classe Televisao.java poderão ser executados.

Na Figura 4, é apresentado um exemplo de execução dos métodos para definir um novo canal e um novo volume, observe:

ALGORITMOS E ESTRUTURA DE DADOS



```
UtilizarTV.java
public class UtilizarTV{
    public static void main(String args[]){
        Televisao t1 = new Televisao();
        Televisao t2 = new Televisao();
        t1.defineMarca("Samsung");
        t1.defineCanal(3);
        t1.local("SALA");
        t1.ligar();
        t1.defineCanal(3);
        t1.aumentarVolume();
    }
}
```

Figura 4 | Executando métodos na classe UtilizarTV.java.

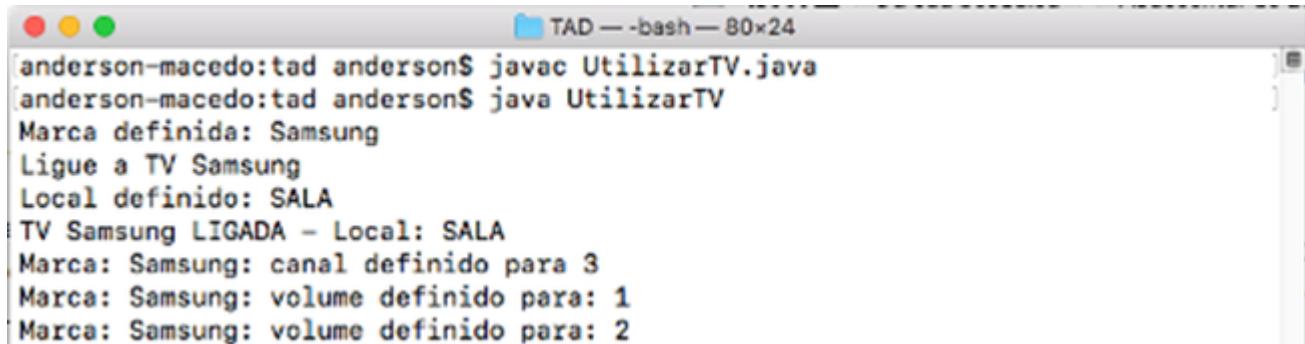
Na Figura 5 a seguir, você pode verificar a estrutura de diretórios deste exemplo. Ela foi organizada em um diretório chamado “TAD”, e dentro dele, foram adicionados todos os arquivos relacionados à prática. Os arquivos “UtilizarTV.java” e “Televisao.java” contêm o código com os métodos, instâncias e outros elementos necessários. Os arquivos “.class” são gerados como resultado do processo de compilação quando executamos o comando correspondente. Consulte a Figura 5 para visualizar esses arquivos.

ALGORITMOS E ESTRUTURA DE DADOS



Figura 5 | Diretório referente aos arquivos do projeto.

Para compilar os arquivos e gerar os arquivos “.class” correspondentes, você deve utilizar o comando “javac” seguido do nome do arquivo e sua extensão. Veja o exemplo na Figura 6.



```
anderson-macedo:tad anderson$ javac UtilizarTV.java
anderson-macedo:tad anderson$ java UtilizarTV
Marca definida: Samsung
Ligue a TV Samsung
Local definido: SALA
TV Samsung LIGADA - Local: SALA
Marca: Samsung: canal definido para 3
Marca: Samsung: volume definido para: 1
Marca: Samsung: volume definido para: 2
```

Figura 6 | Compilando os arquivos.

Outra maneira de compilar arquivos em Java ou em qualquer outra linguagem de programação, fora de um ambiente de desenvolvimento integrado, é utilizando comandos no prompt do sistema operacional. Na Figura 6, você pode observar o comando para compilar o código em Java. Após a compilação, o código é executado por meio do comando: “java UtilizarTV”.

Vamos Exercitar?

O objetivo deste estudo é a aplicação de conceitos de tipos abstratos de dados, focando na criação de uma estrutura que represente as informações do sistema que deverá ser desenvolvido para uma empresa que necessita gerenciar seu estoque de maneira eficiente. Este sistema deverá gerenciar o estoque de produtos, com ações como: incluir novos produtos, excluir produtos, atualizar a quantidade de produtos disponíveis e calcular o valor total do estoque. Para resolver esta situação-problema, você deve:

- **Definição do TDA Produto:** criar uma estrutura que represente as informações essenciais de um produto, como código, descrição, quantidade em estoque e preço unitário.

ALGORITMOS E ESTRUTURA DE DADOS

- **Adicionar produto ao estoque:** criar um procedimento para adicionar um novo produto ao estoque, considerando a atualização do total de produtos.
- **Remover produto do estoque:** estabelecer um procedimento para remover um produto do estoque com base no código do produto fornecido.
- **Atualizar quantidade em estoque:** implementar um procedimento para atualizar a quantidade em estoque de um produto específico, identificado pelo código.
- **Calcular valor total do estoque:** desenvolver uma função que calcule o valor total do estoque, considerando a multiplicação da quantidade pelo preço unitário para cada produto.
- **Exemplo de utilização:** no contexto do programa principal, realizar operações como adição, remoção e atualização de produtos no estoque, além de calcular o valor total do estoque.

Este processo proporciona um controle eficaz do estoque, facilitando a manipulação de produtos na empresa de varejo. Cada passo contribui para a compreensão e implementação adequada do sistema de controle de estoque.

Saiba mais

Para saber mais sobre os tipos abstratos de dados no que diz respeito à construção de algoritmos com esta técnica, acesse a dissertação intitulada como: "*Descoberta e composição de serviços web semânticos através de algoritmo genético baseado em tipos abstratos de dados*", no repositório da Universidade Federal de Alagoas – Instituto de Computação.

SOARES, E. A. [Descoberta e composição de serviços web semânticos através de algoritmo genético baseado em tipos abstratos de dados](#). 2009. 79 f. Dissertação (Mestrado em Modelagem Computacional de Conhecimento) - Universidade Federal de Alagoas, Maceió, 2009.

Bons estudos!

Referências

CORMEN, T. **Algoritmos** – Teoria e Prática. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

ALGORITMOS E ESTRUTURA DE DADOS

Aula 2

Definição e Elementos de Listas Ligadas

Definição e elementos de listas ligadas

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe boas-vindas a esta aula e esperamos que este conteúdo tenha valor agregado na construção do seu conhecimento.

Prosseguindo no conteúdo de estruturas de dados, você vai conhecer e compreender como a estrutura de uma lista ligada funciona, para conseguir desenvolver soluções para os mais diversos nichos do mercado que utilizam a estrutura de lista como base do seu funcionamento. Poderá também optar pela criação de novos processos automatizados baseados no conceito de listas ligadas.

No decorrer desta aula, serão abordados os conceitos necessários a essa estrutura de dados, com o objetivo de serem utilizados no desenvolvimento aplicações ou na construção de ideias inovadoras, provendo soluções para o mercado tecnológico de sistemas. Portanto, serão abordados, nesta aula, a definição e elementos de listas ligadas.

Vamos para um exemplo prático! Alguns serviços comuns de manutenção de veículos são realizados constantemente pelos seus proprietários com o objetivo de prevenir acidentes e

ALGORITMOS E ESTRUTURA DE DADOS

minimizar as chances de uma pane no meio das ruas e rodovias. Assim, será apresentada uma sugestão de solução com o auxílio da estrutura de listas para este problema.

Portanto, estudando os conceitos e colocando em prática os procedimentos que vão ser discutidos nesta aula, podemos concluir que o material vai contribuir com seu aprendizado no que diz respeito ao avanço nos conteúdos abordados na disciplina.

Convidamos você a conhecer essa estrutura de dados e a entender como ela pode auxiliar na sua atividade de desenvolvimento de sistemas e na liderança de equipes com propostas técnicas algorítmicas inovadoras para solução de problemas computacionais.

Vamos Começar!

Estruturas de dados são métodos de organizar e distribuir informações, otimizando a busca e a manipulação de dados por meio de algoritmos. O estudo de estruturas de dados desempenha um papel fundamental no desenvolvimento de programas e algoritmos, oferecendo uma variedade de opções para diferentes aplicações em sistemas específicos.

Conforme mencionado por Cormen (2013), o uso de vetores para representar conjuntos de dados é uma abordagem primitiva, na qual um tamanho máximo de elementos é pré-definido no vetor (vet). Podemos observar um exemplo na Figura 1.

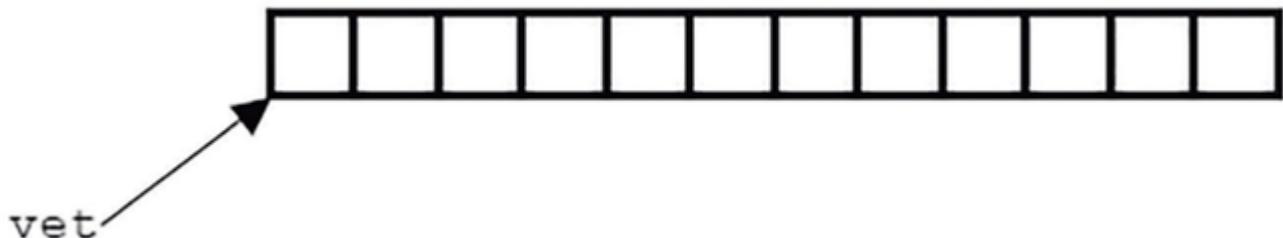


Figura 1 | Exemplo de vetor. Fonte: adaptada de Cormen (2013).

A utilização de vetores na construção de algoritmos é comum, visto que eles se tornam matéria-prima para se trabalhar com listas dos mais variados tipos. Por exemplo, é possível trabalhar com vetores para implementar pilhas, filas e os diversos tipos de listas existentes.

Apesar dessa característica dos vetores, ele não é uma estrutura flexível quando o seu tamanho precisa ser redefinido. Quando o número de elementos ultrapassa a capacidade do vetor, é necessário expandir a sua dimensão para acomodar esses elementos adicionais. Podemos fazer uma analogia com uma bancada de computadores que originalmente tem capacidade para cinco computadores. Se comprarmos mais dois, será preciso providenciar uma bancada maior para receber os novos.

ALGORITMOS E ESTRUTURA DE DADOS

Para resolver este conflito dentro da estrutura de dados, precisamos de uma estrutura que possibilite o crescimento do vetor ou a sua redução de forma automática, isto é, dinâmica. Estamos falando então da estrutura de dado “listas ligadas”.

Definição de listas ligadas

Uma lista ligada é uma coleção representada por $L: [a_1, a_2, \dots, a_n]$, onde $n > 0$. Sua característica principal é a organização dos elementos em sequência, baseando-se apenas na relação de posição relativa entre eles.

De acordo com Cormen (2013), também é conhecida como lista encadeada. Ela é composta por um conjunto de dados que estão dispostos em nós sequenciais. A relação de sucessão entre esses elementos é determinada por meio de ponteiros que indicam a posição do próximo elemento, podendo a lista estar ou não ordenada.

Observe a Figura 2. Nela, pode ser visualizado o modelo de uma lista ligada. A estrutura da lista é composta por uma sequência de elementos encadeados, conhecidos como nós da lista. Cada nó contém duas informações: o elemento que está armazenado e um ponteiro que aponta para o próximo elemento na lista.

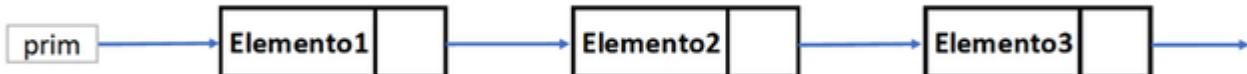


Figura 2 | Modelo de lista ligada. Fonte: adaptada de Cormen (2013).

A Figura 2 exemplifica que os Elementos1, Elementos2, e assim sucessivamente; é o valor do item que se encontra armazenado na lista. O espaço em branco à direita de cada elemento receberá o endereço do próximo elemento da lista. Existe neste caso uma diferença da estrutura de um vetor convencional, em que o armazenamento é feito de forma contígua ou linear; já a lista ligada estabelece uma sequência de forma lógica e concreta.

Ao lidar com listas encadeadas, é essencial definir um ponto inicial ou um ponteiro que aponte para o início da lista. A partir desse ponto, podemos realizar diversas operações, como inserção, remoção e busca de elementos na lista.

As operações básicas de manipulação de listas, conforme descritas por Cormen (2013), incluem:

- Criação ou definição da estrutura de uma lista.
- Inicialização da lista.
- Inserção com base em um endereço como referência.
- Alocação de um endereço de nó para inserção na lista.
- Remoção do nó com base em um endereço como referência.

ALGORITMOS E ESTRUTURA DE DADOS

- Deslocamento do nó removido da lista.

Quando uma lista está completamente sem nós, ela é denominada vazia ou nula. Dessa forma, o valor do ponteiro também será nulo. Observe a Figura 3.

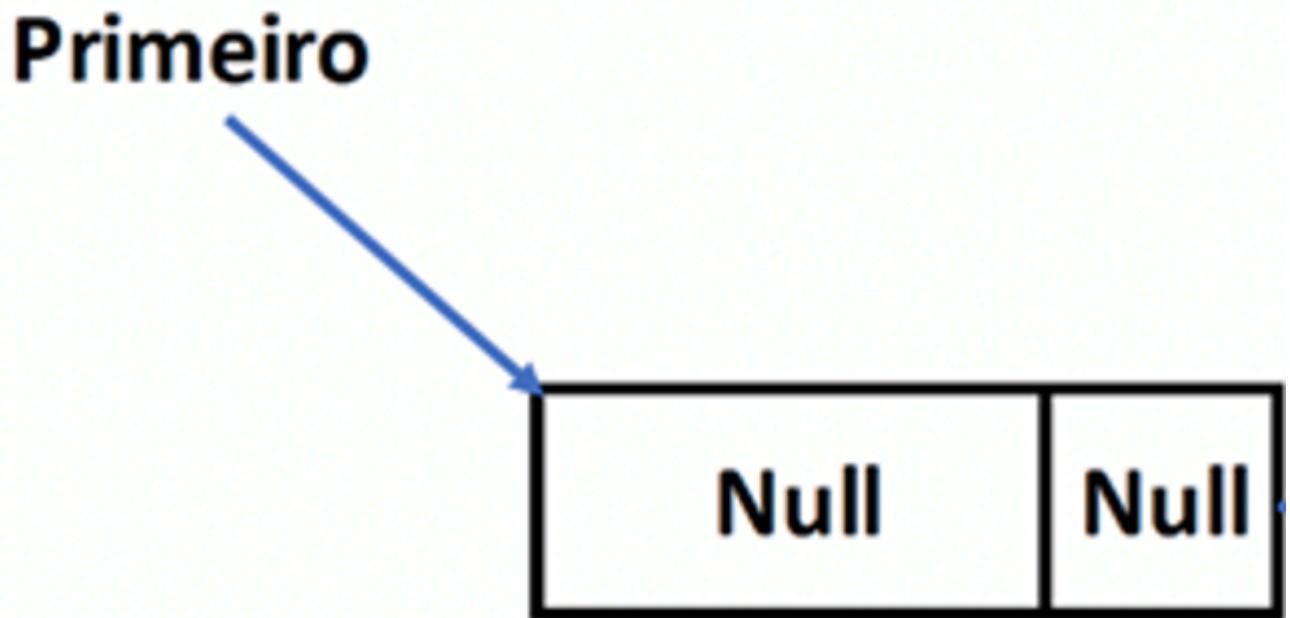


Figura 3 | Lista vazia com ponteiro nulo.

Implementação de listas ligadas

As listas ligadas são consideradas fundamentais em programação de computadores. Elas são compostas por nós que contêm dados e um ponteiro que aponta para o próximo nó na sequência. Existem dois tipos principais de listas ligadas: lista com cabeça e lista sem cabeça. Abordaremos os detalhes e um exemplo de cada uma dessas listas.

- **Lista ligada com cabeça:** uma lista ligada com cabeça é uma lista em que o primeiro nó (cabeça) não contém dados reais, mas é usado apenas para simplificar a manipulação da lista. É útil ao realizar operações de inserção e remoção, pois evita a necessidade de tratar o primeiro nó de forma especial. A Figura 4 apresenta um exemplo de código em C para criar uma lista ligada com cabeça.

ALGORITMOS E ESTRUTURA DE DADOS

```
struct Node {
    int data;
    struct Node* next;
};

struct LinkedList {
    struct Node* head;
};

int main() {
    struct LinkedList myList;
    myList.head = (struct Node*)malloc(sizeof(struct Node));
    myList.head->next = NULL;

    // Inserção de elementos
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = 42;
    newNode->next = myList.head->next;
    myList.head->next = newNode;

    // Remoção de elementos
    struct Node* temp = myList.head->next;
    myList.head->next = temp->next;
    free(temp);

    return 0;
}
```

Figura 4 | Lista ligada com cabeça.

- **Lista ligada sem cabeça:** uma lista ligada sem cabeça é uma lista que não possui um nó de cabeça adicional. A manipulação dessa lista pode ser um pouco mais complexa, já que os primeiros nós precisam ser tratados de forma especial. A Figura 5, apresenta um exemplo de código em C para criar uma lista ligada sem cabeça.

ALGORITMOS E ESTRUTURA DE DADOS

```
struct Node {  
    int data;  
    struct Node* next;  
};  
  
int main() {  
    struct Node* myList = NULL;  
  
    // Inserção de elementos no início  
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));  
    newNode->data = 42;  
    newNode->next = myList;  
    myList = newNode;  
  
    // Remoção do primeiro elemento  
    struct Node* temp = myList;  
    myList = myList->next;  
    free(temp);  
  
    return 0;  
}
```

Figura 5 | Lista ligada sem cabeça.

Os códigos apresentados são exemplos simples de listas ligadas na linguagem de programação C, e a escolha entre listas com ou sem cabeça depende dos requisitos do seu programa. Ambas as abordagens têm suas vantagens e desvantagens, e a seleção adequada depende do contexto da sua aplicação.

Citaremos alguns dos benefícios que essa estrutura pode oferecer para o seu programa. Vamos a elas:

- **Flexibilidade:** listas ligadas são dinâmicas, o que significa que você pode facilmente adicionar ou remover elementos sem a necessidade de pré-alocação de memória. Isso as torna adequadas para situações em que o tamanho da coleção de dados pode variar.

ALGORITMOS E ESTRUTURA DE DADOS

- **Inserções e remoções eficientes:** inserir ou remover elementos no início ou no meio de uma lista ligada pode ser uma operação muito eficiente, especialmente em comparação com outras estruturas de dados, como *arrays* (vetores), que podem ser operações custosas.
- **Uso eficiente de memória:** como a memória é alocada apenas para os elementos que compõem a lista, listas ligadas podem economizar memória quando comparadas a estruturas de dados estáticas, como *arrays*.
- **Ordenação dinâmica:** listas ligadas podem ser facilmente reorganizadas, tornando-as adequadas para a implementação de algoritmos de ordenação.
- **Implementação de outras estruturas de dados:** as listas ligadas são componentes fundamentais em muitas outras estruturas de dados, como pilhas, filas e árvores.
- **Facilidade de implementação:** listas ligadas podem ser implementadas de maneira relativamente simples, mesmo por programadores iniciantes.
- **Versatilidade:** existem vários tipos de listas ligadas, como listas simplesmente encadeadas, listas duplamente encadeadas e listas circulares, que se adequam a diferentes necessidades de programação.
- **Conexões flexíveis:** cada nó de uma lista ligada possui um ponteiro para o próximo nó (e, em listas duplamente encadeadas, para o nó anterior), o que permite a criação de estruturas de dados complexas e relacionadas.
- **Acessibilidade:** com ponteiros para os próximos elementos, é possível percorrer a lista de forma eficiente, seja para buscar, processar ou manipular os dados.

No entanto, é importante observar que as listas ligadas também têm desvantagens, como a necessidade de mais memória (devido aos ponteiros) e acesso não tão eficiente a elementos individuais (em comparação com *arrays*). Portanto, a escolha de usar listas ligadas depende dos requisitos específicos de cada problema de programação.

Siga em Frente...

Aplicações de listas ligadas

A utilização de listas no cotidiano é mais frequente do que imaginamos. As listas podem ser aplicadas em várias situações diárias, muitas vezes passando despercebidas. Um exemplo comum é no ambiente de trabalho, onde frequentemente precisamos criar listas de tarefas para organizar nosso dia de trabalho. Após a elaboração da lista, é possível atribuir prioridades a cada tarefa, como ilustrado na Figura 6.

Isso permite numerar as tarefas restantes com base em sua importância ou grau de prioridade. Dessa forma, as listas se tornam uma ferramenta útil para gerenciar nossas atividades diárias de maneira mais eficiente.

ALGORITMOS E ESTRUTURA DE DADOS



Figura 6 | Lista de prioridade de tarefas. Fonte: Shutterstock.

As listas ligadas são estruturas de dados extremamente versáteis e úteis em programação, encontrando aplicações em uma ampla variedade de cenários. Outra aplicação das listas ligadas ocorre quando a quantidade de dados que será armazenada não é conhecida antecipadamente e precisa ser alocada dinamicamente.

Em conclusão, as listas ligadas oferecem uma solução eficaz para gerenciar dados dinâmicos em diversas aplicações, desde listas de tarefas a bancos de dados complexos. Sua capacidade de alocar memória conforme necessário e de permitir operações de inserção e exclusão eficientes faz com que elas se tornem uma escolha valiosa na programação.

Portanto, dominar o conceito de listas ligadas é fundamental para qualquer desenvolvedor que deseja criar aplicativos robustos e eficientes.

Vamos Exercitar?

O objetivo deste estudo é a aplicação de conceitos de estruturas de dados baseadas em listas ligadas, focando no desenvolvimento de um sistema de atendimento para uma oficina de alinhamento e balanceamento de pneus.

O nosso exemplo prático apresentado inicialmente, permite aplicar os conceitos de listas ligadas

ALGORITMOS E ESTRUTURA DE DADOS

em um cenário real, desenvolvendo um sistema de atendimento para uma oficina de alinhamento e balanceamento de pneus. Isso demonstra a utilidade das estruturas de dados na resolução de problemas do mundo real e a importância de uma fila organizada em um ambiente de atendimento. Para resolver esta situação-problema, você deve:

- **Definir a estrutura do nó:** crie uma estrutura (*struct*) que representará um cliente na fila de atendimento. Essa estrutura pode incluir informações como nome, placa do veículo e tipo de serviço (alinhamento ou balanceamento).
- **Inicialização da lista ligada:** crie um ponteiro que apontará para o início da lista ligada, indicando que a fila está vazia.
- **Adição de clientes à fila:** implemente a lógica para adicionar clientes à fila. Isso envolve a criação de um novo nó, preenchendo-o com informações relevantes e conectando-o à lista ligada.
- **Atendimento dos clientes:** desenvolva a lógica para atender os clientes na ordem correta. Isso requer a remoção do cliente que está no início da fila e a atualização do ponteiro da lista ligada.
- **Exibição de informações:** crie funções para listar os clientes que aguardam atendimento. Isso pode ser útil para os mecânicos visualizarem a lista de clientes.
- **Teste e validação:** implemente casos de teste para verificar se o sistema de atendimento está operando conforme o esperado. Teste cenários de fila vazia, adição de clientes, atendimento e exibição de informações.

Conclua o estudo de caso refletindo sobre a aplicação dos conceitos de listas ligadas à resolução de um problema prático. Reflita também como a estrutura de dados ajuda a gerenciar o atendimento de clientes de forma organizada.

Saiba mais

Para saber mais sobre as listas ligadas no que diz respeito à construção de algoritmos com esta técnica, acesse o artigo intitulado como: "*Listas Encadeadas*", no repositório da Universidade de São Paulo – USP.

FEOFILOFF, P. [Listas encadeadas](#). Universidade de São Paulo, 2018.

Bons estudos!

Referências

CORMEN, T. **Algoritmos** – Teoria e Prática. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

ALGORITMOS E ESTRUTURA DE DADOS

SZWARCFITER, J L., MARKENZON, L. **Estruturas de dados e seus algoritmos.** 3^a ed. Rio de Janeiro: LTC, 2020.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 3

Operações com Listas Ligadas

Operações com listas ligadas

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe boas-vindas a esta aula e esperamos que esse conteúdo agregue valor em seu constante aprendizado na disciplina de algoritmos e estrutura de dados.

Agora que você já conhece um pouco sobre a definição e aplicação da estrutura de listas ligadas, vamos continuar aprofundando nosso estudo com as operações com listas ligadas.

Você vai conhecer e compreender que é possível realizar as operações de inserção e remoção de elementos em uma lista ligada. Fique atento ao conteúdo aqui abordado, pois trataremos dessas operações de forma prática, com exemplos que vão tornar mais fácil o seu aprendizado.

ALGORITMOS E ESTRUTURA DE DADOS

Para exemplificar, vamos utilizar um cenário da realidade gastronômica. Nos dias atuais, a entrega de *fast food* (que, em português, significa comida rápida) é um serviço que muitas pessoas utilizam diariamente para se alimentar, pois não têm tempo hábil disponível para se locomover a um restaurante ou para preparar uma refeição em suas próprias casas. Muitas pessoas estão ocupadas com inúmeras atividades para cumprir diariamente.

O restaurante, por sua vez, precisa de um sistema eficiente para controle e entrega desses pedidos. Assim, apresentaremos uma sugestão de solução com o auxílio da estrutura de listas ligadas, utilizando suas operações.

Convidamos você a se aprofundar na estrutura de listas ligadas e conhecer um pouco mais suas operações, além de entender como ela pode auxiliar tarefas de criação de sistemas computacionais com maior eficiência dos seus algoritmos.

Vamos Começar!

O objetivo desta aula é abordar as três operações importantes dentro da estrutura de listas ligadas. Vale ressaltar que, em algumas fontes, é possível encontrar essa mesma estrutura com outro nome: “listas encadeadas”. Contudo, para esta disciplina, vamos designá-las como listas ligadas.

As três operações em listas ligadas que abordaremos são:

- Inserir elementos.
- Percorrer a lista.
- Remover elementos.

Uma lista nula ou (vazia) é uma lista sem nenhum nó; desta forma, o ponteiro para o próximo elemento possui valor nulo (Tenenbaum, 1995).

Uma lista é uma estrutura de dados que demonstra dinamismo, pois a quantidade de seus nós pode sofrer alterações significativas à medida que elementos são adicionados ou removidos. Essa característica dinâmica de uma lista está em contraste com a natureza estática de um vetor, cujo tamanho permanece invariável. Por isso, para a programação de vários sistemas precisamos de listas ligadas, haja vista que determinadas listas são atualizadas a todo momento (Cormen, 2022).

Inserir elementos

Para adicionar um elemento a uma lista ligada, é essencial reservar dinamicamente espaço na memória para armazenar o elemento e conectá-lo à lista já existente (Cornem, 2022).

ALGORITMOS E ESTRUTURA DE DADOS

A inserção de elementos em uma lista ligada é uma operação fundamental na manipulação dessa estrutura de dados. Vamos explorar os principais cenários de inserção: no início, no meio e no final da lista.

Inserção no início da lista

A inserção de elementos no início de uma lista é um processo relativamente simples. Isso envolve criar um novo nó para armazenar o elemento que desejamos adicionar e, em seguida, ajustar os ponteiros para que o novo nó se torne o primeiro da lista. A principal vantagem da inserção de elementos no início de uma lista ligada está relacionada à eficiência do processo. Inserir um elemento no início da lista é uma operação de tempo constante ($O(1)$), o que significa que o tempo necessário para realizar a inserção não depende do tamanho da lista (Cormen, 2022). A Figura 1, exibe um exemplo da inserção de um elemento no início da lista.

```
struct Node* newNode = criarNovoNode(dado);
newNode->proximo = primeiroNode;
primeiroNode = newNode;
```

Figura 1 | Inserção no início da lista.

Inserção no meio da lista

Para inserir um elemento no meio da lista, precisamos encontrar a posição desejada e ajustar os ponteiros para incluir o novo nó. A Figura 2, exibe um exemplo da inserção de um elemento no meio da lista.

```
struct Node* newNode = criarNovoNode(dado);
newNode->proximo = nodeAnterior->proximo;
nodeAnterior->proximo = newNode;
```

Figura 2 | Inserção no meio da lista.

ALGORITMOS E ESTRUTURA DE DADOS

A inserção de elementos no meio de uma lista ligada envolve algumas ações específicas, tais como:

- **Encontrar a posição de inserção:** antes de inserir um elemento no meio da lista, é necessário encontrar a posição desejada. Isso pode ser feito percorrendo a lista a partir do início e contando os nós ou usando algum critério específico, como a comparação de valores em cada nó.
- **Criar um novo nó:** após identificar a posição desejada, você deve alocar espaço na memória para um novo nó que conterá o elemento a ser inserido.
- **Atualizar ponteiros:** a próxima etapa é ajustar os ponteiros para “costurar” o novo nó na lista. Isso envolve modificar os ponteiros do nó anterior e do novo nó. O ponteiro do nó anterior deve apontar para o novo nó, e o ponteiro do novo nó deve apontar para o nó que originalmente ocupava essa posição.
- **Manter a coesão da lista:** certifique-se de que os ponteiros da lista permaneçam consistentes após a inserção, garantindo que nenhum nó seja perdido ou quebrado durante o processo.
- **Verificar limites:** certifique-se de que você não está tentando inserir o elemento além dos limites da lista. Caso contrário, você corre o risco de acessar posições inválidas na lista.

Inserir elementos no meio da lista é uma operação que requer mais cálculos e manipulações de ponteiros do que a inserção no início. No entanto, essa abordagem é útil quando a ordem dos elementos na lista é importante ou quando você deseja manter os elementos em uma ordem específica.

A complexidade de tempo da inserção no meio de uma lista ligada é geralmente $O(n)$, onde n é o número de elementos na lista, devido à necessidade de percorrer a lista para encontrar a posição de inserção. Portanto, essa operação pode ser mais lenta à medida que a lista cresce (Cormen, 2022).

Inserção no final da lista

Inserir um elemento no final da lista requer percorrer a lista até o último nó e, em seguida, anexar o novo nó. A Figura 3 exibe um exemplo da inserção de um elemento no final da lista.

```
struct Node* newNode = criarNovoNode(dado);
newNode->proximo = NULL;
ultimoNode->proximo = newNode;
ultimoNode = newNode;
```

ALGORITMOS E ESTRUTURA DE DADOS

Figura 3 | Inserção no final da lista.

Portanto, a inserção em uma lista ligada pode ser adaptada para atender às necessidades de diferentes situações, seja no início, meio ou final da lista, tornando-a uma estrutura de dados flexível e poderosa para várias aplicações.

Siga em Frente...

Percorrer a lista

Um aspecto essencial ao trabalhar com listas ligadas é a capacidade de percorrer ou buscar elementos dentro delas. Isso envolve navegar pela lista para localizar um item específico ou para realizar operações como contar elementos, encontrar um valor desejado ou qualquer outra tarefa que envolva acesso aos dados da lista.

Para realizar esse processo, é necessário seguir alguns passos bem definidos. Neste texto, explicaremos detalhadamente como percorrer elementos em uma lista ligada, passo a passo.

- **Passo 1 – iniciar na cabeça da lista:** o primeiro passo ao percorrer uma lista ligada é posicionar um ponteiro no início da lista, geralmente na cabeça (ou nó cabeça). Essa cabeça não contém dados reais, mas é usada como um ponto de partida para a navegação na lista. Ela aponta para o primeiro elemento real da lista.
- **Passo 2 – usar um *loop* para navegar:** o processo de percorrer uma lista é normalmente realizado usando um *loop*, como *while* ou *for*, que continua até que o elemento desejado seja encontrado ou até que o final da lista seja alcançado. A cada iteração do *loop*, você deve mover o ponteiro para o próximo elemento da lista.
- **Passo 3 – verificar o elemento atual:** em cada iteração do *loop*, você deve verificar o elemento atual (para saber se é o que você está procurando) e, se necessário, realizar operações com ele. Por exemplo, você pode comparar o elemento atual com o valor desejado, contar os elementos ou realizar outra ação com base nas necessidades do seu programa.
- **Passo 4 – avançar para o próximo elemento:** para avançar para o próximo elemento na lista, você precisa atualizar o ponteiro para apontar para o próximo nó ou elemento. Esse avanço é essencial para garantir que você esteja sempre inspecionando o elemento correto.
- **Passo 5 – repetir o processo:** o processo de verificação, avanço e repetição deve ser continuado até que você encontre o elemento desejado ou até que alcance o final da lista, o que é indicado quando o ponteiro aponta para um nó vazio ou NULL.

A Figura 4 e a Figura 5 exibem um exemplo do processo de busca de um elemento em uma lista ligada.

ALGORITMOS E ESTRUTURA DE DADOS

```
#include <stdio.h>
#include <stdlib.h>

// Estrutura do nó da lista ligada
struct Node {
    int data;
    struct Node* next;
};

// Função para percorrer a lista e buscar um elemento
struct Node* buscarElemento(struct Node* head, int elemento) {
    struct Node* current = head;

    while (current != NULL) {
        if (current->data == elemento) {
            return current;
        }
        current = current->next;
    }

    return NULL; // Elemento não encontrado
}
```

Figura 4 | Busca de um elemento.

A Figura 5, por sua vez, exibe a codificação da função *main* referente ao mesmo exemplo de busca de um elemento na lista.

ALGORITMOS E ESTRUTURA DE DADOS

```
int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;

    second->data = 2;
    second->next = third;

    third->data = 3;
    third->next = NULL;

    int elementoBuscado = 2;
    struct Node* resultado = buscarElemento(head, elementoBuscado);

    if (resultado != NULL) {
        printf("Elemento encontrado: %d\n", resultado->data);
    } else {
        printf("Elemento não encontrado.\n");
    }

    return 0;
}
```

Figura 5 | Função main.

ALGORITMOS E ESTRUTURA DE DADOS

Neste exemplo, o código define uma lista ligada simples com três elementos e, em seguida, busca o elemento com o valor desejado (2). Se o elemento for encontrado, ele é impresso na tela; caso contrário, é exibida uma mensagem informando que o elemento não foi encontrado.

O processo de busca em uma lista ligada é fundamental em muitas aplicações e fornece uma maneira flexível de acessar e manipular dados em memória. É importante entender os princípios envolvidos na navegação de listas ligadas para aproveitar ao máximo essa estrutura de dados.

Remover elementos

Remover elementos de uma lista ligada é uma operação essencial quando se trabalha com essa estrutura de dados. Isso permite liberar memória e manter a lista organizada e eficiente.

Discutiremos detalhadamente como remover elementos de uma lista ligada, passo a passo:

- **Passo 1 – iniciar na cabeça da lista:** o processo de remoção começa com o posicionamento de um ponteiro no início da lista. Assim como ao percorrer a lista, o ponteiro normalmente começa na cabeça da lista, que é um nó fictício usado para facilitar a navegação.
- **Passo 2 – encontrar o elemento a ser removido:** antes de remover um elemento, você deve localizá-lo. Para fazer isso, você precisa navegar pela lista usando um *loop* (como *while* ou *for*) e comparar os valores de cada elemento com o valor que deseja remover.
- **Passo 3 – atualizar os ponteiros:** uma vez que você localizou o elemento a ser removido, é hora de realizar a operação de exclusão. Para fazer isso, você precisa atualizar os ponteiros no nó anterior ao que você está removendo. Esses ponteiros são fundamentais para manter a integridade da lista. O ponteiro *next* do nó anterior deve apontar para o seguinte ao nó que será removido. O nó a ser removido deve ser liberado da memória (desalocado).
- **Passo 4 – lidando com o primeiro elemento:** remover o primeiro elemento da lista é uma operação especial, pois o anterior ao primeiro elemento é o próprio nó cabeça. Portanto, a operação de remoção começa da seguinte maneira: atualize o ponteiro *next* do nó cabeça para apontar para o segundo elemento da lista. Libere o primeiro elemento.
- **Passo 5 – lidando com a remoção de elementos no meio ou no fim:** para remover um elemento no meio ou no final da lista, o processo é semelhante. Localize o elemento a ser removido. Atualize o ponteiro *next* do nó anterior para apontar para o seguinte ao nó que está sendo removido. Libere o nó a ser removido.
- **Passo 6 – lidar com elementos não encontrados:** é importante verificar se o elemento que você deseja remover realmente existe na lista. Se o elemento não for encontrado, você deve lidar com essa situação adequadamente, seja retornando uma mensagem de erro ou tomando a ação apropriada de acordo com os requisitos do seu programa.

A Figura 6 exibe um exemplo do processo de remoção de um elemento em uma lista ligada.

ALGORITMOS E ESTRUTURA DE DADOS

```
struct Node {
    int data;
    struct Node* next;
};

// Função para remover um elemento da lista
void removerElemento(struct Node* head, int elemento) {
    struct Node* temp = head;
    struct Node* prev = NULL;

    if (temp != NULL && temp->data == elemento) {
        head = temp->next;
        free(temp);
        return;
    }

    while (temp != NULL && temp->data != elemento) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) return;

    prev->next = temp->next;
    free(temp);
}
```

Figura 6 | Remoção de elementos na lista.

A Figura 7 exibe a função que faz a impressão da lista.

ALGORITMOS E ESTRUTURA DE DADOS

```
void imprimirLista(struct Node* node) {  
    while (node != NULL) {  
        printf("%d ", node->data);  
        node = node->next;  
    }  
}
```

Figura 7 | Impressão da lista.

Por fim, a Figura 8 exibe o método principal encarregado de executar o método de remoção de um elemento solicitado por parâmetro e, após isso, imprimir novamente a lista.

ALGORITMOS E ESTRUTURA DE DADOS

```
int main() {
    struct Node* head = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1;
    head->next = NULL;

    struct Node* second = (struct Node*)malloc(sizeof(struct Node));
    second->data = 2;
    second->next = NULL;
    head->next = second;

    struct Node* third = (struct Node*)malloc(sizeof(struct Node));
    third->data = 3;
    third->next = NULL;
    second->next = third;

    printf("Lista ligada original: ");
    imprimirLista(head);

    int elementoARemover = 2;
    removerElemento(head, elementoARemover);

    printf("\nLista ligada após remoção do elemento %d: ", elementoARemover)
    imprimirLista(head);

    return 0;
}
```

Figura 8 | Método principal.

Neste exemplo, temos uma lista ligada simples com três elementos, e estamos removendo o elemento com o valor 2. A função “removerElemento()” realiza a remoção do elemento, atualizando os ponteiros conforme discutido anteriormente.

A remoção de elementos em listas ligadas é fundamental para manter a organização e eficiência das estruturas de dados. Trata-se de uma operação comum em muitas aplicações da

ALGORITMOS E ESTRUTURA DE DADOS

programação. Entender os princípios envolvidos na remoção de elementos é essencial para trabalhar com eficácia com listas ligadas.

Vamos Exercitar?

O cenário do nosso exemplo prático é um restaurante de *fast food* que oferece um serviço de entrega. Eles precisam de um sistema eficiente para registrar pedidos, organizar o fluxo de preparação e entrega, e manter um histórico dos pedidos para futura referência. Para materialização dessa ideia, pede-se que seja utilizado no sistema conceitos de operações com listas ligadas.

A seguir veja as necessidades do sistema:

- O sistema deve criar uma lista ligada vazia para representar pedidos.
- Quando um pedido é recebido, um novo nó é criado com as informações do pedido.
- O novo nó é inserido na lista ligada.
- O status do pedido é atualizado para refletir seu progresso (aguardando preparação, pronto para entrega, entregue).
- A lista ligada permite que o restaurante mantenha um histórico de pedidos anteriores para referência.

A sugestão dos passos para realização do sistema de pedido são:

1. Inicialização da lista ligada.
2. Recebimento do pedido.
3. Criação de um novo nó.
4. Inserção na lista ligada.
5. Atualização de status.
6. Preparação e entrega.
7. Registro de entrega.
8. Histórico de pedidos.

Esse sistema de pedidos com operações em listas ligadas ajuda o restaurante a gerenciar eficientemente os pedidos dos clientes, garantindo que sejam preparados e entregues no prazo, e mantendo um registro para análise futura.

Saiba mais

Para saber mais sobre as listas ligadas e como construir um programa contendo a codificação das funções de inserir e remover informações em uma lista ligada, acesse em sua Biblioteca

ALGORITMOS E ESTRUTURA DE DADOS

Virtual no livro intitulado “*Estrutura de Dados e seus algoritmos*, o Capítulo 2 – “Listas Lineares”, a partir da página 15 em diante.

SZWARCFITER, J. L.; MARKENZON, L. [Estruturas de Dados e seus algoritmos](#). 3^a edição. Grupo GEN, 2010.

Bons estudos!

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 4

Listas Duplamente Ligadas

Listas duplamente ligadas

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

ALGORITMOS E ESTRUTURA DE DADOS

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe boas-vindas a mais esta aula; que você continue progredindo no estudo da estrutura de dados de listas ligadas! Agora vamos estudar as listas duplamente ligadas, ou listas duplamente encadeadas, como também são conhecidas. Temos certeza de que este conteúdo será desafiador e, ao mesmo tempo, gratificante para a construção de seu aprendizado.

Nesta estrutura, você vai aprender como armazenar informações valiosas de ações realizadas dentro do sistema.

Suponha que você é um desenvolvedor de software encarregado de criar um aplicativo de edição de imagens. Os usuários desejam a capacidade de desfazer e refazer várias alterações que elaboram em suas imagens, como ajustar cores, aplicar filtros, recortar e redimensionar. Para gerenciar esse histórico de alterações, você decidiu usar estruturas de dados chamadas listas duplamente ligadas.

Seu aplicativo possui uma tela de edição na qual os usuários podem carregar imagens e aplicar várias edições. Você precisa criar um sistema que permita aos usuários desfazer (*undo*) e refazer (*redo*) alterações em suas imagens; para isso, você usará listas duplamente ligadas para armazenar o histórico das alterações.

Nesta aula, serão apresentadas essas operações e seus conceitos. Você aprenderá como são implementadas em uma linguagem de programação.

Convidamos você a conhecer o algoritmo da lista duplamente ligada e a trabalhar com suas operações. Bons estudos!

Vamos Começar!

Nesta aula sobre a estrutura de listas duplamente ligadas, o objetivo é abordar as duas operações mais utilizadas dentro da própria estrutura. Vale ressaltar que, em algumas fontes, é possível também encontrar essa mesma estrutura com outro nome: listas duplamente encadeadas. Contudo, nesta disciplina, usaremos a designação “listas duplamente ligadas”.

As duas operações abordadas serão:

- Inserção de elementos na lista.

ALGORITMOS E ESTRUTURA DE DADOS

- Remoção de elementos da lista.

Também contextualizaremos e definiremos as listas duplamente ligadas. Até o momento, você explorou o conceito de listas ligadas, que são estruturas de dados baseadas em encadeamento simples. Em uma lista ligada, os elementos são organizados sequencialmente, de modo que cada nó contém um ponteiro apontando para o próximo elemento na lista, criando uma espécie de relação em cadeia, semelhante a uma lista de prioridades (Cormen, 2022).

Uma característica importante das listas ligadas é que elas não permitem uma travessia reversa, ou seja, não é possível percorrer os elementos na direção inversa, começando pelo último elemento e indo em direção ao primeiro. Além disso, a remoção de um elemento em uma lista ligada simples requer um esforço adicional, uma vez que é necessário percorrer toda a lista para localizar o elemento anterior. Isso acontece porque, dado um ponteiro para um elemento, você tem acesso apenas ao próximo na sequência, não ao anterior.

Definição de lista duplamente ligada

Dando prosseguimento nesta aula, vamos explorar as listas duplamente ligadas, um tipo de estrutura de dados que difere das listas ligadas simples. Em uma lista duplamente ligada, cada nó contém não apenas um elemento de informação, mas também dois ponteiros: um que aponta para o próximo elemento e outro que aponta para o elemento anterior (Cormen, 2022).

Essa característica permite que você acesse tanto o próximo elemento quanto o elemento anterior na lista. Dessa forma, você pode percorrer a lista em ambas as direções, da extremidade final até o início ou vice-versa. O primeiro elemento da lista aponta para NULL (valor nulo) como seu elemento anterior, assim como o último elemento também tem um ponteiro NULL, conforme ilustrado na Figura 1 (Cormen, 2022).

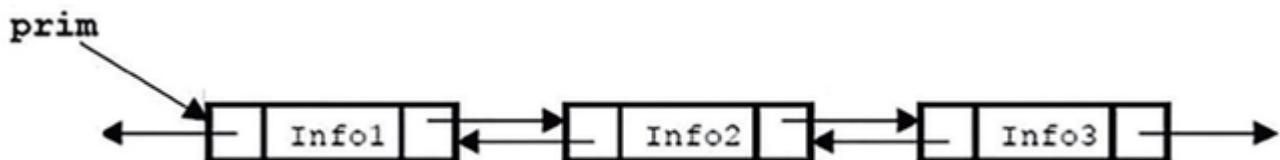


Figura 1 | Lista duplamente ligada. Fonte: adaptada de Cormen (2022).

De acordo com Tenenbaum (1995), um nó em uma lista duplamente ligada consiste na criação de três campos:

- Um campo-elemento para a informação.
- Um ponteiro direcionado para o próximo elemento.
- Um ponteiro direcionado para o elemento anterior.

ALGORITMOS E ESTRUTURA DE DADOS

Ao criar uma lista duplamente ligada, é necessário definir não apenas o tipo de dados que será usado na sua implementação e o ponteiro que aponta para o próximo elemento, mas também um ponteiro que faz referência ao elemento anterior na lista, seguindo o modelo de implementação apresentado na Figura 2:

```
struct lista {  
    int info;  
    struct lista* ant;  
    struct lista* prox;  
};
```

Figura 2 | Codificando uma lista duplamente ligada.

Da mesma maneira que ocorre com a lista ligada simples, a lista duplamente ligada requer inicialização antes de seu uso após a declaração. Uma maneira comum de realizar essa inicialização é criando uma função que retorne a lista com um valor nulo. Observe a Figura 3.

```
Lista* inicializa(void) {  
    return NULL;  
}
```

Figura 3 | Inicialização (retorna lista vazia).

Inserir elementos

Inserir elementos em uma lista duplamente ligada é uma operação fundamental em estruturas de dados. Essa operação permite adicionar elementos a uma lista que pode ser percorrida em ambas as direções, ou seja, a partir do início até o final e vice-versa. Uma lista duplamente ligada é uma estrutura que contém elementos conhecidos como nós, e cada nó contém dois ponteiros: um que aponta para o próximo elemento da lista e outro que aponta para o elemento anterior (Cormen, 2022).

ALGORITMOS E ESTRUTURA DE DADOS

A inserção de elementos em uma lista duplamente ligada pode ocorrer em diferentes posições da lista: no início, no meio ou no final. Cada um desses casos possui um procedimento específico. Vamos apresentar em primeiro lugar a inserção no início da lista.

Inserção no início da lista duplamente ligada

Para inserir um elemento no início de uma lista duplamente ligada, você precisa criar um novo nó que contenha as informações do elemento a ser adicionado. Em seguida, ajusta-se os ponteiros do novo nó para que o próximo elemento aponte para o antigo primeiro nó e o ponteiro do nó anterior a ele aponte para o novo (Cormen, 2022). A Figura 4 apresenta um exemplo.

```
void inserirNoInicio(ListaDupla* lista, int valor) {
    Nodo* novoNodo = criarNodo(valor);
    if (lista->inicio == NULL) {
        lista->inicio = lista->fim = novoNodo;
    } else {
        novoNodo->prox = lista->inicio;
        lista->inicio->ant = novoNodo;
        lista->inicio = novoNodo;
    }
}
```

Figura 4 | Início da lista.

Inserção no meio da lista duplamente ligada

Para inserir um elemento no meio da lista, é necessário encontrar a posição correta, criando um novo nó e ajustando os ponteiros do nó anterior e posterior a ele para que apontem para o novo nó. A Figura 5 apresenta um exemplo.

ALGORITMOS E ESTRUTURA DE DADOS

```
void inserirNoMeio(ListaDupla* lista, int valor, Nodo* nodoAnterior) {
    Nodo* novoNodo = criarNodo(valor);
    novoNodo->prox = nodoAnterior->prox;
    nodoAnterior->prox->ant = novoNodo;
    nodoAnterior->prox = novoNodo;
    novoNodo->ant = nodoAnterior;
}
```

Figura 5 | Meio da lista.

Inserção no final da lista duplamente ligada

Para inserir um elemento no final da lista, você deve criar um novo nó e ajustar os ponteiros do novo nó e do nó que atualmente é o último para que apontem um para o outro. A Figura 6 apresenta um exemplo.

```
void inserirNoFinal(ListaDupla* lista, int valor) {
    Nodo* novoNodo = criarNodo(valor);
    if (lista->inicio == NULL) {
        lista->inicio = lista->fim = novoNodo;
    } else {
        novoNodo->ant = lista->fim;
        lista->fim->prox = novoNodo;
        lista->fim = novoNodo;
    }
}
```

Figura 6 | Final da lista.

Esses são os procedimentos básicos para inserir elementos em uma lista duplamente ligada. A capacidade de inserção em diferentes posições torna essa estrutura de dados versátil e útil para uma variedade de aplicações.

ALGORITMOS E ESTRUTURA DE DADOS

Siga em Frente...

Remover elementos

A operação de eliminação em uma lista duplamente ligada possibilita a remoção de um elemento da lista, desde que seja conhecido o ponteiro para esse elemento específico. Para tornar esse processo mais eficaz, pode-se utilizar a função de busca para localizar o elemento desejado na lista. Uma vez encontrado, realiza-se o ajuste no encadeamento da lista, garantindo que o elemento seja devidamente retirado da alocação de memória.

Um trecho da função de busca pode ser implementado pelo código exibido na Figura 7.

```
Lista* busca(Lista* l, int v) {  
    Lista* p;  
    for (p = l; p != NULL; p = p->prox) {  
        if (p->info == v)  
            return p;  
    }  
    return NULL;  
}
```

Figura 7 | Trecho da função de busca.

Dessa forma, encontrado o elemento, basta apontar o anterior para o próximo e o próximo para o anterior, permitindo que o elemento no meio da lista seja removido do encadeamento.

As listas duplamente ligadas oferecem a vantagem de permitir o acesso tanto aos elementos anteriores quanto aos posteriores a um determinado nó, o que torna a remoção mais flexível em comparação com as listas simplesmente ligadas. A seguir será detalhado o processo de remoção em alguns passos:

- **1º passo – localizar o elemento:** a primeira etapa é encontrar o elemento que desejamos remover. Para isso, percorremos a lista de um nó para o outro até localizar o elemento

ALGORITMOS E ESTRUTURA DE DADOS

desejado.

- **2º passo – ajustar os ponteiros:** uma vez que localizamos o elemento, precisamos ajustar os ponteiros do elemento anterior e posterior a ele para que eles não apontem mais para o nó que será removido. Isso envolve a atualização dos ponteiros “*prox*” do nó anterior ao elemento “*ant*” do nó posterior ao elemento.
- **3º passo – liberar memória:** por fim, após ajustar os ponteiros, é fundamental liberar a memória alocada para o nó que estamos removendo. A liberação é crucial para evitar vazamento de memória.

Vamos agora ver uma implementação em C que demonstra o processo de remoção de elementos em uma lista duplamente ligada. A Figura 8 apresenta a criação do No.

ALGORITMOS E ESTRUTURA DE DADOS

```
#include <stdio.h>
#include <stdlib.h>

struct No {
    int info;
    struct No* ant;
    struct No* prox;
};

typedef struct No Lista;
```

Figura 8 | Criação da struct No.

ALGORITMOS E ESTRUTURA DE DADOS

No código da Figura 9, a função “removeElemento()” recebe a lista e o valor a ser removido como argumentos. Ela percorre a lista até encontrar o elemento desejado, ajusta os ponteiros “*ant*” e “*prox*” dos nós vizinhos e libera a memória do nó removido.

ALGORITMOS E ESTRUTURA DE DADOS

```
Lista* removeElemento(Lista* l, int valor) {
    Lista* p = l;

    while (p != NULL && p->info != valor) {
        p = p->prox;
    }

    if (p == NULL) {
        // Elemento não encontrado
        return l;
    }

    if (p->ant != NULL) {
        p->ant->prox = p->prox;
    }

    if (p->prox != NULL) {
        p->prox->ant = p->ant;
    }

    if (p == l) {
        // Se o elemento a ser removido é o primeiro da lista
        l = l->prox;
    }

    free(p); // Liberar memória

    return l; // Retorna a lista atualizada
}
```

Figura 9 | Criação da struct No.

ALGORITMOS E ESTRUTURA DE DADOS

A Figura 10 apresenta o método *main*, no qual o método “removeElement()” é chamado à execução, realizando a remoção de um elemento na lista duplamente ligada.

```
int main() {
    Lista* minhaLista = NULL;

    // Inserindo elementos na lista (implementação não mostrada)

    // Removendo o elemento com valor 5 da lista
    minhaLista = removeElement(minhaLista, 5);

    // Resto do código para exibir ou manipular a lista
    return 0;
}
```

Figura 10 | Execução do método “removeElement()”.

A remoção de elementos em uma lista duplamente ligada envolve a localização do elemento, o ajuste dos ponteiros e a liberação da memória. Essa operação é essencial para manter a integridade e eficiência da lista. É essencial entender como realizar essas operações para tirar o máximo proveito de listas duplamente ligadas em projetos de programação e estruturas de dados.

Vamos Exercitar?

O cenário do exemplo prático é desenvolvimento de um aplicativo que possui uma tela de edição na qual os usuários possam carregar imagens e aplicar várias edições. Você precisa criar um sistema que permita aos usuários desfazer (*undo*) e refazer (*redo*) alterações em suas imagens; para isso, você está usando listas duplamente ligadas para armazenar o histórico das alterações. Veja a seguir uma sugestão para resolução da situação-problema.

- **Início da edição:** cada vez que um usuário inicia uma nova edição em uma imagem, você cria uma nova entrada na lista duplamente ligada. Cada entrada contém informações sobre a operação realizada (por exemplo, “ajustar cores”, “aplicar filtro”, “recortar”, etc.) e os parâmetros específicos da operação.
- **Desfazer (*undo*):** se um usuário deseja desfazer a última alteração, você navega para o nó anterior na lista e aplica a operação inversa. Por exemplo, se a última alteração foi

ALGORITMOS E ESTRUTURA DE DADOS

“aumento de brilho”, a operação de desfazer seria “diminuição de brilho”.

- **Refazer (*redo*):** se um usuário deseja refazer uma alteração que foi desfeita, você navega para o próximo nó na lista (se existir) e reaplica a operação original.
- **Limites do histórico:** você pode definir limites para o histórico, ou seja, quantas operações de *undo/redo* são possíveis. Quando o histórico atinge o limite, as operações mais antigas são descartadas.

Usando listas duplamente ligadas, você pode criar um sistema eficaz de controle de edições para o aplicativo de edição de imagens. Os usuários poderão desfazer e refazer várias etapas, permitindo-lhes experimentar livremente enquanto mantêm o controle sobre as mudanças realizadas em suas imagens. A estrutura de dados das listas duplamente ligadas torna essa funcionalidade possível, fornecendo um histórico eficiente e flexível para todas as operações de edição.

Saiba mais

Para saber mais sobre as listas ligadas e como construir um programa contendo a codificação das funções para inserir e remover informações em uma lista ligada, acesse em sua Biblioteca Virtual no livro intitulado como: *Estruturas de Dados e Seus Algoritmos*, o Capítulo 2 – no tópico Listas Lineares em Alocação Encadeada, as Seções 2.7.2 “Pilhas e Filas” e 2.7.4 “Listas Duplamente Encadeadas”.

SZWARCFITER, J. L.; MARKENZON, L. [Estruturas de Dados e Seus Algoritmos](#). 3^a edição. Grupo GEN, 2010.
Bons estudos!

Referências

CORMEN, T. **Algoritmos** – Teoria e Prática. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

ALGORITMOS E ESTRUTURA DE DADOS

Aula 5

Encerramento da Unidade

Videoaula de Encerramento



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Chegada

Olá, estudante! Para desenvolver a competência desta Unidade, que é “compreender os conceitos fundamentais relacionados a tipos abstratos de dados (TADs) e listas ligadas, incluindo o que são, como funcionam e por que são usados”, você deverá primeiramente conhecer os conceitos fundamentais abordados ao longo da Unidade de ensino.

Uma das competências essenciais é a capacidade de entender e identificar as situações em que os tipos abstratos de dados (TADs) e as listas ligadas são apropriados.

Os TADs permitem a criação de abstrações de dados personalizadas, encapsulando detalhes de implementação e facilitando a modularidade do código. Já as listas ligadas são estruturas de dados dinâmicas que permitem o armazenamento e a recuperação eficiente de informações.

Com o avanço do conteúdo, foram expostos situações e problemas do mundo real em que TADs e listas ligadas são apropriados. Você aprendeu como avaliar as aplicações dessas estruturas em diversos contextos, desde a criação e manipulação de elementos em listas de vários tipos.

ALGORITMOS E ESTRUTURA DE DADOS

Essas competências vão auxiliar você a não apenas desenvolver algoritmos mais eficientes, mas também fornecerão uma base sólida para enfrentar desafios complexos no desenvolvimento de software. Ao compreender quando e como usar TADs e listas ligadas, você estará bem-preparado para criar soluções robustas e escaláveis em suas futuras carreiras na área de tecnologia da informação.

É Hora de Praticar!



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Convidamos você a colocar em prática o conhecimento adquirido nesta Unidade de ensino. Para isso, vamos abordar um Estudo de caso com o cenário descrito a seguir:

Imagine que você está desenvolvendo um sistema de gerenciamento de contatos pessoais. Esse sistema permitirá que os usuários armazenem informações sobre pessoas, como seus nomes, números de telefone e endereços. Para implementar esse sistema, você decide usar tipos abstratos de dados (TADs) e, especificamente, uma estrutura de dados de lista ligada.

TAD “Contato”:

O TAD “Contato” é a representação de uma pessoa em sua lista de contatos. Ele contém campos que armazenam as informações relevantes, como o nome da pessoa, o número de telefone e o endereço.

TAD “No”:

O TAD “No” é um elemento individual em sua lista ligada. Cada nó representa um único contato e armazena uma instância do TAD “Contato”. Além disso, ele possui uma referência ao próximo nó na lista, o que cria o encadeamento dela.

TAD “ListaLigada”:

O TAD “ListaLigada” é a própria lista que contém todos os contatos. Ele é composto por nós encadeados, e o primeiro nó (primeiro contato) é a entrada principal para a lista.

A criação deste sistema fará com que os usuários possam adicionar, remover e buscar contatos em sua lista de contatos. Por exemplo, eles podem adicionar novos contatos, remover contatos antigos ou procurar informações de contato de alguém quando necessário.

Esse cenário ilustra como os TADs e as listas ligadas podem ser aplicados em um contexto do mundo real para criar um sistema de gerenciamento de contatos eficaz. A criação de tais estruturas de dados e sua aplicação prática são habilidades essenciais para qualquer programador ou desenvolvedor de software.

ALGORITMOS E ESTRUTURA DE DADOS

- Considerando a definição de TADs como estruturas de dados que encapsulam dados e operações em um único objeto, reflita sobre como a utilização de TADs pode simplificar o processo de desenvolvimento de software. Você pode dar exemplos de situações em que TADs seriam especialmente úteis?
- A seleção da estrutura de dados correta é fundamental ao desenvolver algoritmos eficientes. Reflita sobre como a escolha entre listas ligadas e *arrays* (vetores) pode afetar o desempenho de um programa. Em que situações as listas ligadas superam os *arrays*, e vice-versa? Você pode dar exemplos dessas situações?

Sugestão de solução

- **Passo 1: definição do TAD “Contato”** : primeiro, vamos criar um TAD chamado “Contato” para representar informações de contato. O TAD “Contato” pode incluir campos como nome, telefone e endereço. Considerando a linguagem de programação Java, ele ficaria assim:

```
class Contato {
    String nome;
    String telefone;
    String endereco;

    public Contato(String nome, String telefone, String endereco) {
        this.nome = nome;
        this.telefone = telefone;
        this.endereco = endereco;
    }
}
```

Figura 1 | TAD “Contato” em Java.

- **Passo 2: implementação da lista ligada:** segue a implementação da lista ligada que armazenará os contatos. É necessária uma lista ligada simples, na qual cada nó conterá um contato. Considerando a linguagem de programação Java, os nós e a lista ficariam assim:

ALGORITMOS E ESTRUTURA DE DADOS

```
class No {  
    Contato contato;  
    No proximo;  
  
    public No(Contato contato) {  
        this.contato = contato;  
    }  
}
```

Figura 2 | Classe “No” em Java.

ALGORITMOS E ESTRUTURA DE DADOS

```
class ListaLigada {  
    No primeiro;  
  
    public void adicionarContato(Contato contato) {  
        No novoNo = new No(contato);  
        novoNo.proximo = primeiro;  
        primeiro = novoNo;  
    }  
  
    public void removerContato(String nome) {  
        No atual = primeiro;  
        No anterior = null;  
  
        while (atual != null && !atual.contato.nome.equals(nome)) {  
            anterior = atual;  
            atual = atual.proximo;  
        }  
  
        if (atual != null) {  
            if (anterior != null) {  
                anterior.proximo = atual.proximo;  
            } else {  
                primeiro = atual.proximo;  
            }  
        }  
    }  
}
```

Figura 3 | Classe “ListaLigada” em Java.

ALGORITMOS E ESTRUTURA DE DADOS

```
public Contato buscarContato(String nome) {  
    No atual = primeiro;  
  
    while (atual != null) {  
        if (atual.contato.nome.equals(nome)) {  
            return atual.contato;  
        }  
        atual = atual.proximo;  
    }  
  
    return null;  
}
```

Figura 3 | Classe “ListaLigada” em Java - cont.

- **Passo 3: exemplo de teste (*main*):** dentro da classe e método principal, é necessário instanciar a classe “ListaLigada” e a classe “Contato” para executar os métodos de adicionar, buscar e remover contatos da lista. Considerando a linguagem de programação Java, a implementação ficaria assim:

ALGORITMOS E ESTRUTURA DE DADOS

```
public class Main {  
    public static void main(String[] args) {  
        Contato contato1 = new Contato("Anderson", "123-456-7890", "Rua A");  
        Contato contato2 = new Contato("Macedo", "987-654-3210", "Rua B");  
  
        ListaLigada agenda = new ListaLigada();  
        agenda.adicionarContato(contato1);  
        agenda.adicionarContato(contato2);  
  
        String nomeParaBusca = "Anderson";  
        Contato busca = agenda.buscarContato(nomeParaBusca);  
  
        if (busca != null) {  
            System.out.println("Contato encontrado: " + busca.nome);  
        } else {  
            System.out.println("Contato não encontrado.");  
        }  
    }  
}
```

Figura 4 | Implementação na Classe “Main” em Java.

O Estudo de caso envolveu a criação de uma lista ligada em Java, aplicando os conceitos de tipos abstratos de dados. Foram criados os TADs “Contato”, “No”, e “ListaLigada”. Demonstramos o uso dessas estruturas para gerenciar contatos. A implementação em Java passo a passo mostra como criar, adicionar, remover e buscar contatos na lista ligada.

Neste infográfico, apresentamos resumidamente os principais conceitos relacionados a tipos abstratos de dados (TAD) e listas ligadas, dois tópicos fundamentais em estrutura de dados e desenvolvimento de sistemas.

Convidamos você a refletir sobre os conceitos apresentados no infográfico!

ALGORITMOS E ESTRUTURA DE DADOS



CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.
CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

ALGORITMOS E ESTRUTURA DE DADOS

SZWARCFITER, J. L., MARKENZON, L. **Estruturas de dados e seus algoritmos.** 3^a ed. Rio de Janeiro: LTC, 2020.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Unidade 2

Pilhas E Filas

Aula 1

Pilhas: Definição

Pilhas: definição



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe boas-vindas a esta aula e esperamos que este conteúdo tenha valor agregado na construção do seu conhecimento.

Durante esta aula, vamos apresentar a você mais uma estrutura de dado que está presente em nossa disciplina. Estamos falando das pilhas.

ALGORITMOS E ESTRUTURA DE DADOS

Você conhecerá e compreenderá a definição de uma pilha, os seus conceitos teóricos, bem como as formas de implementação desta estrutura de dado e exemplos de aplicações práticas, o que não pode faltar para o seu aprendizado.

As pilhas são utilizadas nos mais variados cenários do cotidiano e em muitas atividades que já praticamos atualmente, sem sabermos que os conceitos e práticas da estrutura de dado pilha estão presentes neles.

Neste momento, chamamos sua atenção para um contexto. Suponha que você está trabalhando em uma aplicação de processamento de textos, um editor de textos similar ao aplicativo Word, da Microsoft, por exemplo. Você está digitando seu TCC em um documento e deseja formatar um parágrafo em negrito. Ao selecionar um texto e clicar no botão “Negrito”, o programa deve aplicar a formatação ao parágrafo atual; porém, ela deve ainda permitir que o usuário desfaça essa ação, se necessário. Neste pequeno contexto, temos implicitamente a estrutura de dados pilha em ação. Esse recurso de desfazer a última ação é um exemplo clássico da pilha.

Após a abordagem dos conceitos dessa estrutura de dado e a apresentação de exemplos de aplicação deste algoritmo, você estará apto a implementar codificações que possam resolver problemas da realidade profissional com o algoritmo de pilha.

Convidamos você a conhecer essa estrutura de dado e a entender como ela pode auxiliar sua atividade como um desenvolvedor de sistemas.

Vamos Começar!

O objetivo desta aula é abordar a definição da estrutura de dados pilha e apresentar a você algumas formas de implementação por meio de exemplos de aplicações com esta estrutura de dados. Para tanto, vamos iniciar por sua definição.

De acordo com Cormen (2022), uma pilha é uma estrutura de dados simples que utiliza ponteiros para representação de conjuntos dinâmicos. Vamos explorar este conceito: os conjuntos são dinâmicos, pois eles podem ser alterados. Entenda conjuntos como os elementos que serão adicionados em uma pilha. Por exemplo, em uma pilha de nomes de pessoas, o conjunto dinâmico são os nomes que estarão dentro da pilha. Observe a Figura 1.

ALGORITMOS E ESTRUTURA DE DADOS

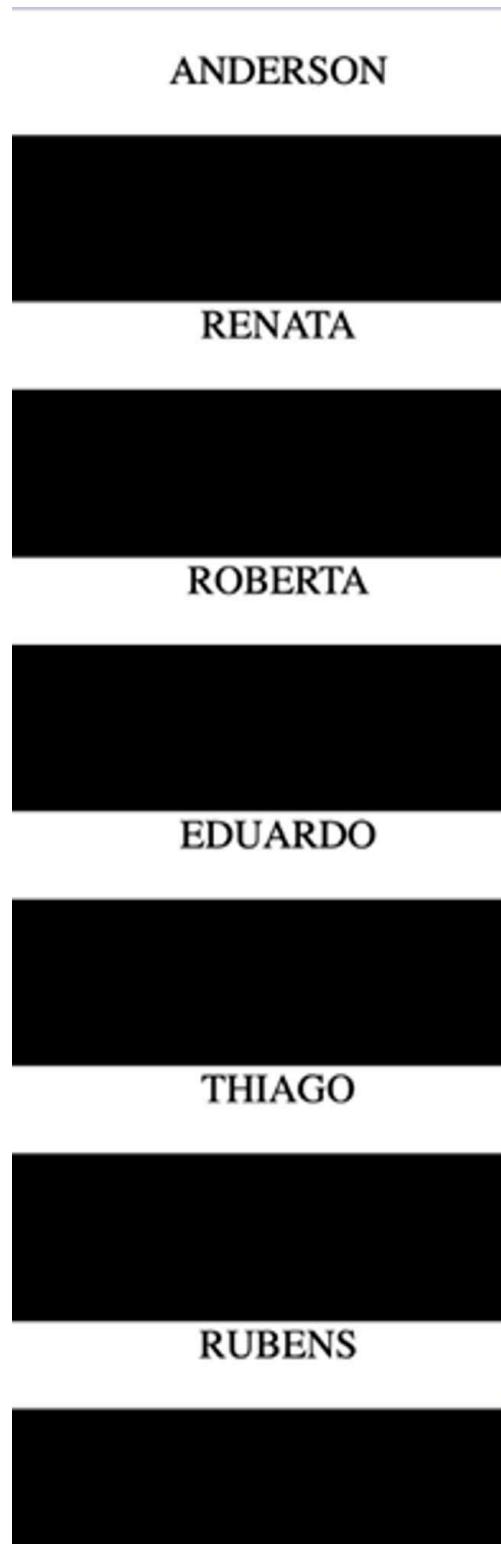


Figura 1 | Pilha de pessoas.

ALGORITMOS E ESTRUTURA DE DADOS

Os nomes que estão na pilha são do tipo texto; se fossem valores numéricos, esses elementos pertenceriam a um outro conjunto de dados.

Outra característica da pilha é que ela permite a inserção e remoção de elementos em apenas uma das extremidades da estrutura. A essa extremidade damos o nome de topo da pilha. Os algoritmos referentes a inserção e remoção de elementos serão estudados na próxima aula.

Outra definição importante da pilha que também faz parte de seus conceitos, de acordo com Cormen (2022), diz respeito ao fato de que o elemento eliminado do conjunto é o mais recente inserido. As pilhas são listas com uma disciplina de acesso denominada LIFO (*Last In, First Out*). Isso significa que, quando um elemento entra na pilha, ele pode ser removido somente enquanto for o último elemento inserido.

Analizando novamente a Figura 1, o único elemento que pode ser removido imediatamente é “Anderson”, pois ele está no topo da lista. O elemento “Renata” só poderá ser removido da pilha se, e somente se, o elemento “Anderson” for removido primeiro.

A ilustração de uma pilha também pode ser visualizada em exemplos do cotidiano, como: uma pilha de pratos, uma pilha de livros, uma pilha de roupas; ou então em exemplos do mercado de trabalho, como: o processo de produto que é composto por várias etapas em uma esteira. A Figura 2 abaixo apresenta o processo de envasamento de garrafas de vinho. Para que o vinho caia corretamente na garrafa, ele deve estar posicionada na esteira. Este exemplo também caracteriza o algoritmo de pilhas.

ALGORITMOS E ESTRUTURA DE DADOS



Figura 2 | Envasamento de vinhos. Fonte: Adobe Stock.

Também podemos explorar o conceito de pilha em atividades cotidianas, como empilhar e retirar pratos. Uma situação comum envolve o armazenamento de pratos em uma pilha em armários de cozinha. Dependendo do peso dos pratos, só é possível acessar aqueles que foram colocados no armário primeiro quando retirarmos os pratos que estão por cima. É exatamente esse tipo de controle que nossos algoritmos precisam aplicar a essa estrutura.

Na verdade, não importa qual tipo de objeto esteja sendo colocado na pilha, seja o processo de envasamento de garrafas, seja uma pilha de pratos. O aspecto crucial é a maneira como manipulamos esses elementos.

Na Figura 3, vemos uma pilha de pratos e outra de livros. É fundamental entender que, ao projetar um algoritmo, não podemos simplesmente levantar todos os livros ou pratos que estão no topo de uma só vez para acessar o último item da pilha, pois precisamos seguir as regras da estrutura de dado. Essa mesma lógica se aplica tanto à pilha de pratos quanto a qualquer outro tipo de elemento.

ALGORITMOS E ESTRUTURA DE DADOS



Figura 3 | Pilhas de objetos. Fonte: Freepik.

Siga em Frente...

De acordo as formas de implementação, temos duas opções: realizar o processo de manipulação dos elementos por meio de uma pilha ou por meio de um *array*, popularmente conhecido como vetor. A principal diferença entre essas duas estruturas está na maneira em que elas são utilizadas pela lógica do algoritmo.

Ao contrário de vetor, que possui um tamanho fixo predefinido, uma pilha é declarada com um tamanho dinâmico, o qual se ajusta continuamente à medida que elementos são inseridos ou removidos da pilha. Isso é possível graças à alocação dinâmica de memória.

Para ilustração dessas duas formas de implementação, vamos utilizar a linguagem de programação Java. A primeira será demonstrada utilizando o vetor. Algumas linguagens possuem classes nativas para trabalhar com a estrutura de pilha, porém, se entendermos o conceito, é possível criar as nossas próprias classes e, assim, sermos proprietários do nosso código, o que nos possibilita um maior controle. Observe a Figura 4.

ALGORITMOS E ESTRUTURA DE DADOS

```
3 public class Pilha {  
4  
5     static final int MAX = 3;  
6     int top;  
7     int a[] = new int[MAX];  
8  
9     // Construtor  
10    Pilha() {  
11        top = -1;  
12    }  
13  
14}
```

Figura 4 | Classe Pilha – variáveis e construtor.

A Figura 4 exibe a classe Pilha criada na linguagem de programação Java, como já descrito. Observe que as linhas 5, 6 e 7 são utilizadas para a criação de um *array* unidimensional, utilizado para controlar a pilha e as variáveis de controle de tamanho da pilha. O construtor que define a variável *top* com o valor *-1* serve para a pilha iniciar vazia. É importante lembrar que os procedimentos de inserção e remoção de elementos na pilha serão abordados na próxima aula. Em conclusão, a utilização da pilha por meio do vetor “*a[]*” é uma das maneiras de aplicação do conceito de pilha.

A outra maneira de implementação de uma pilha, demonstrada por meio de um objeto pilha, está exibida na Figura 5. Observe.

ALGORITMOS E ESTRUTURA DE DADOS

```
1. import java.util.Stack;  
2.  
3. public class ExemploPilha {  
4.     public static void main(String[] args) {  
5.         // Declaração de uma pilha de inteiros  
6.         Stack<Integer> pilha = new Stack<>();  
7.     }  
8. }
```

Figura 5 | Declaração de uma pilha.

Na linha 1, o código está fazendo a declaração do pacote “*Stack*” do Java para permitir comandos de natureza da pilha que se encontram já desenvolvidos na linguagem. Fazendo este “*import*”, você, como desenvolvedor, poderá utilizar esses comandos sem precisar criá-los.

Exploraremos o código da Figura 5: na linha 6, estamos declarando uma variável chamada “pilha” com o tipo **Stack<Integer>**. Isso significa que estamos criando uma instância da classe *Stack* que vai armazenar elementos do tipo *Integer*, ou seja, números inteiros.

A parte **Stack<Integer>** é uma declaração de tipo genérico, o que permite a você especificar o tipo de elemento contido pela pilha. Neste caso, estamos usando *Integer* como o tipo de elemento. Isso garante que só podemos adicionar números inteiros à pilha, e que, quando retirarmos elementos da pilha, eles serão automaticamente convertidos de volta para números inteiros.

A variável *pilha* é, na verdade, uma referência para a instância da pilha que acabamos de criar.

Em conclusão, mesmo utilizando a pilha por um pacote nativo da linguagem, é possível darmos um nome próprio para a variável, desde que se respeitem as regras para nomenclatura de variáveis. Algumas delas são: não começar com caractere numérico, não conter espaços em branco, não ser um nome reservado à própria linguagem; por fim, também não é recomendado utilizar acentuação, embora algumas linguagens permitam essa prática e não acusem erros.

Esperamos que tenha ficado claro para você esses conceitos iniciais da estrutura de dados da pilha. Como demonstrado, esse recurso na programação é abordado em várias temáticas e regras de negócio do mercado de trabalho.

ALGORITMOS E ESTRUTURA DE DADOS

Vamos Exercitar?

Vamos para o exemplo prático! Imagine que você tenha escrito um parágrafo importante do seu TCC e deseja aplicar a formatação de negrito a esse parágrafo para destacá-lo. Você seleciona o parágrafo e clica no botão “**Negrito**” do aplicativo. A formatação é aplicada, mas, por algum motivo, você percebe que não era o parágrafo desejado ou que a formatação não ficou como esperado.

Como o desenvolvedor do aplicativo de processamento de textos pode implementar o recurso de “Desfazer” (*undo*) para permitir que o usuário reverta a ação de formatação de negrito no parágrafo atual?

Na sequência, estão descritos os passos para implementar o recurso de “Desfazer” para a ação de formatação de negrito:

1. **Captura de ação:** quando o usuário clica no botão “Negrito”, a aplicação captura a ação de formatação, incluindo o parágrafo afetado e os detalhes da formatação (por exemplo, a ação “Negrito”).
2. **Criação da estrutura de pilha:** a aplicação cria uma estrutura de pilha (uma pilha vazia) para rastrear as ações de formatação.
3. **Empilhar ação:** a aplicação empilha a ação capturada na pilha. Isso inclui o parágrafo afetado e os detalhes da formatação.
4. **Ação desfeita (*undo*):** quando o usuário deseja desfazer a ação de formatação de negrito, a aplicação verifica a pilha para encontrar a ação mais recente.
5. **Reverter a ação:** a aplicação reverte a formatação aplicada no parágrafo afetado, desfazendo-a completamente.
6. **Remover ação da pilha:** após reverter a ação, a aplicação remove essa ação da pilha, garantindo que a pilha continue rastreando apenas as ações mais recentes.
7. **Feedback para o usuário:** a aplicação fornece feedback ao usuário, indicando que a ação de formatação de negrito foi desfeita com sucesso.

Esses passos garantem que o usuário possa desfazer a última ação de formatação de negrito, se necessário, tornando a experiência de edição do texto mais flexível e confiável. A estrutura de pilha é fundamental para rastrear as ações e permitir o desfazer (*undo*).

Saiba mais

Para saber mais sobre a estrutura de dados “Pilha”, no que diz respeito à construção de algoritmos com esta técnica algorítmica, acesse no livro em sua Biblioteca Virtual intitulado como: *Algoritmos e Estrutura de Dados em Linguagem C*, o Capítulo 9 – Pilhas, a partir da página 272.

BACKES, A. R. [Algoritmos e Estruturas de Dados em Linguagem C](#). 1ª ed. Grupo GEN, 2023.

ALGORITMOS E ESTRUTURA DE DADOS

Bons estudos!

Referências

BACKES, A. R. **Algoritmos e Estruturas de Dados em Linguagem C.** 1^a ed. Grupo GEN, 2023.

CORMEN, T. **Algoritmos – Teoria e Prática.** 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos.** São Paulo: Grupo GEN, 2013.

SZWARCFITER, J L., MARKENZON, L. **Estruturas de dados e seus algoritmos.** 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C.** São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 2

Pilhas: Implementação e Operações

Pilhas: implementação e operações

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

ALGORITMOS E ESTRUTURA DE DADOS

Ponto de Partida

Olá, estudante. Desejamos-lhe boas-vindas a esta aula e esperamos que este conteúdo tenha valor agregado na construção do seu conhecimento.

Durante esta aula, vamos apresentar a você um pouco mais sobre a estrutura de dado pilha.

Você conhecerá e compreenderá algumas operações que essa estrutura nos permite fazer. Entre elas, se encontram:

- Verificação da pilha vazia: operação que testa quando a pilha está vazia, para que não se apresente erro na remoção de elementos.
- Verificação da pilha cheia: testa quando a pilha está cheia, para que não se apresente erro na inserção de elementos.
- Inserção de elementos em uma pilha.
- Remoção de elementos de uma pilha.

Para a aplicação dessas operações em um cenário real, vamos sugerir a seguinte situação: a administração de um movimentado porto marítimo, a fim de otimizar o espaço, deseja melhorar o controle dos containers que chegam e são empilhados. Eles decidiram implementar um sistema de controle usando uma estrutura de pilha para rastrear a ordem de chegada e saída dos containers e para permitir o acesso aos containers mais recentes. Esta situação aborda a inserção de elementos em uma pilha: neste caso, os elementos não são valores numéricos ou textuais, mas containers. Em um cenário real de um porto marítimo, isso acontece todos os dias.

Após a implementação das operações de pilha descritas acima, você estará apto a implementar codificações que resolvam problemas da realidade profissional com esta estrutura de dado.

Convidamos você a aprofundar seus conhecimentos sobre pilhas e a entender como ela pode auxiliar a sua atividade técnica profissional em sistemas de informações.

Vamos Começar!

O objetivo desta aula é abordar claramente as operações de inserção e remoção de elementos em uma pilha, além de apresentar os procedimentos que garantem que essas operações não apresentarão erros em sua execução.

Abordaremos como acontece a codificação inicial no controle da estrutura de pilha a partir de agora; portanto, devemos compreender todas as regras para manipulação de elementos na pilha.

No contexto da pilha, utilizamos uma terminologia específica para representar suas operações. Quando inserimos um novo elemento à pilha, chamamos isso de **empilhamento**; quando

ALGORITMOS E ESTRUTURA DE DADOS

retiramos um elemento já existente da pilha, dizemos **desempilhamento**. Esses termos são amplamente utilizados na descrição das operações dentro desta estrutura de dados.

Inserção de elementos na pilha

Primeiramente, é necessário criar a classe Pilha, que deve conter o método *push()* para adicionar elementos e o método *pop()* para remover elementos da estrutura, bem como funções para verificar se a pilha está cheia ou vazia. Observe a codificação apresentada na Figura 1 por meio da linguagem de programação Java.

```

22  boolean push(int x) {
23
24      if (top >= (MAX - 1)) {
25          System.out.println("Pilha Cheia!");
26          return false;
27      } else {
28          a[++top] = x;
29          return true;
30      }
31  }

```

Figura 1 | Classe Pilha – método de inserção.

A Figura 1 ilustra o funcionamento do método *push()*, o qual desempenha a função de adicionar elementos à pilha. Esse método requer um valor inteiro como parâmetro e, após verificar se a pilha não está no seu limite de capacidade, insere o elemento no vetor *a[]* na linha 28. Vale observar que esse método retorna um valor booleano, indicando verdadeiro quando o valor é inserido com sucesso e falso quando a pilha já atingiu sua capacidade máxima – em outras palavras, se a pilha já está cheia. Neste último caso, o método exibe uma mensagem “Pilha Cheia!”, como pode ser observado na linha 25 da Figura 1.

Siga em Frente...

Remoção de elementos na pilha

Seguindo com nosso exemplo de estrutura de pilha, agora vamos introduzir o método *pop()* da classe Pilha, que tem a responsabilidade de remover elementos da pilha. Você pode conferir sua implementação na Figura 2.

ALGORITMOS E ESTRUTURA DE DADOS

```

33     int pop() {
34         if (top < 0) {
35             System.out.println("Pilha Vazia!");
36             return 0;
37         } else {
38             int x = a[top--];
39             return x;
40         }
41     }

```

Figura 2 | Classe Pilha – método de exclusão.

O método *pop()* retorna um valor inteiro. Esse valor é 0 (zero) quando a pilha já está vazia e o valor que estava no topo da pilha quando a remoção é realizada. A condição que determina se o valor da variável ‘top’ é menor que 0 (zero), conforme visto na linha 34, verifica se há algum elemento a ser removido. A cada remoção, o valor da variável ‘top’ é decrementado, pois ela atua como um controle para a pilha. No método de inserção, explicado anteriormente na Figura 1, a mesma variável ‘top’ é incrementada. Essa lógica de controle é essencial para identificar tanto uma “pilha cheia” quanto uma “pilha vazia”.

Dando sequência em nosso estudo, agora que você já conheceu as operações de inserção e remoção de elementos em uma pilha, vamos abordar exemplos de codificação dessas funcionalidades.

Para realizar o teste, é preciso criar uma nova classe, embora seja possível também usar a classe principal do seu projeto Java, se preferir. Nesta aula, optaremos por usar a classe principal do projeto. Neste exemplo, a classe será chamada de “PilhaTela{ }”. Não é necessário criar métodos adicionais dentro desta classe, pois usaremos o método principal “*main()*”.

Na classe “PilhaTela”, escolheremos fornecer valores fixos e aleatórios para nossos testes, eliminando, assim, a necessidade de ler elementos via teclado usando a classe “*Scanner*”.

Agora, vamos começar a construir a classe “PilhaTela”.

O passo inicial envolve criar uma instância da classe ‘Pilha’ que já foi previamente criada e que contém os métodos “*push()*” e “*pop()*”. A Figura 3 apresenta a codificação desta instância.

ALGORITMOS E ESTRUTURA DE DADOS

```
5  public class PilhaTela {  
6  
7      public static void main(String[] args) {  
8          |  
9          |  
10         Pilha novaPilha = new Pilha();  
11  
12     }
```

Figura 3 | Classe PilhaTela (instância da classe Pilha).

Na Figura 3, o código ilustra a criação de uma instância da classe “Pilha” por meio da criação do objeto **novaPilha**, linha 10. A partir desta linha de código, ganhamos a capacidade de acessar os métodos públicos da classe “Pilha”, (“*push()*” e “*pop()*”), utilizando os princípios da programação orientada a objetos.

No nosso cenário de teste para o método “*push()*”, vamos inserir quatro elementos na pilha. Observe a Figura 4.

```
13      novaPilha.push(10);  
14      novaPilha.push(20);  
15      novaPilha.push(30);  
16      novaPilha.push(40);
```

Figura 4 | Classe PilhaTela (inserindo elementos).

O que acontecerá com o nosso exemplo após a execução das linhas 13 a 16? Tente realizar uma análise passo a passo para responder a essa questão. Imagine que nossa pilha tenha somente espaço para três elementos. Considere mais esta informação para sua análise. Se você observar na Figura 5, vai perceber que o método *push()* já faz este controle, ou seja: embora ele seja usado para adicionar elementos à pilha, o teste para verificar se a pilha está “cheia” está incorporado a ele.

ALGORITMOS E ESTRUTURA DE DADOS

```

22     boolean push(int x) {
23
24         if (top >= (MAX - 1)) {
25             System.out.println("Pilha Cheia!");
26             return false;
27         } else {
28             a[++top] = x;
29             return true;
30         }
31     }

```

Figura 5 | Classe Pilha – verificação da pilha cheia.

Na linha 24, há uma condição que retorna “falso” quando a variável ‘top’ é maior ou igual a ‘MAX – 1’, indicando que o índice atual ultrapassou o último índice válido. Por exemplo, se a pilha tem tamanho 3, o maior índice é 2. Portanto, qualquer valor maior que 2 é considerado “pilha cheia”.

Voltando à Figura 4, estamos usando o método “push()” para adicionar os elementos 10, 20, 30 e 40 à nossa pilha. É importante lembrar que esses elementos serão inseridos no vetor de inteiros “[a]”, criado na classe “Pilha”. No entanto, fica evidente que o elemento 40 não fará parte da pilha, pois ela estava no seu limite de capacidade no momento da inserção.

Dado que o primeiro elemento a entrar na pilha fica na base, nossa pilha ficará com os elementos da base para o topo: 10, 20 e 30. O elemento 40 será rejeitado, como explicado, devido à condição de “pilha cheia”.

Para concluir, na Figura 5, quando a pilha está cheia, além de retornar “falso” como valor booleano, o algoritmo emite uma mensagem indicando que a pilha atingiu sua capacidade máxima, conforme a linha 25.

Vamos testar nosso exemplo para observar o que ocorre ao tentar adicionar mais elementos do que a pilha pode comportar. Uma vez que nossa pilha suporta apenas três elementos, vamos executar o programa tentando inserir os 4 elementos mencionados anteriormente: 10, 20, 30 e 40.

ALGORITMOS E ESTRUTURA DE DADOS

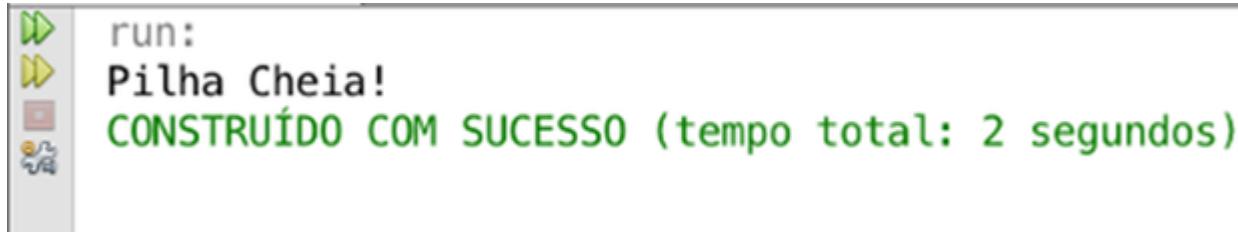


Figura 6 | Executando o programa.

Na Figura 6, podemos ver o resultado quando o algoritmo tenta adicionar o último elemento, neste caso, o valor “40”. Como já explicado, o método “*push()*” retorna o valor lógico “falso” e exibe a mensagem “Pilha Cheia!” por meio do método “*println()*”, utilizado para mostrar informações na tela do programa.

Essa é apenas uma das abordagens para lidar com o algoritmo, ou seja, para evitar que o sistema simplesmente gere um erro quando se tenta inserir mais elementos do que a pilha tem espaço para acomodar.

Considere a viabilidade da criação de métodos separados exclusivamente para verificar quando a pilha está cheia ou vazia. Contudo, para este exemplo em particular, a decisão foi tomada pela incorporação do tratamento dessas condições diretamente nas funções relevantes.

Esperamos que tenha ficado claro para você os conteúdos abordados nesta aula com o desenvolvimento dos exemplos que foram abordados. Assim, conseguimos cumprir o conteúdo, auxiliando você a construir um pouco mais do seu conhecimento sobre estrutura de pilhas.

Vamos Exercitar?

Vamos retomar o cenário proposto de aplicação real no início da aula! Em um porto marítimo, containers de carga entregues por navios e caminhões precisam ser organizados em áreas de armazenamento específicas. O objetivo é empilhar os containers de acordo com a ordem de chegada, permitindo que os mais recentes sejam facilmente acessíveis quando necessário.

Coloque-se no papel de um desenvolvedor, analise o problema e responda qual seria uma solução para realizar o empilhamento desses containers atendendo à demanda.

Na sequência estão descritos os passos do cenário de inserção ou empilhamento dos containers:

1. **Declaração da pilha:** inicialmente, uma pilha vazia é declarada para representar o espaço de armazenamento dos containers.
2. **Chegada de containers:** quando um container é entregue no porto, ele é inserido na pilha. Isso é feito empilhando o container no topo da pilha, representando a ordem de chegada.

ALGORITMOS E ESTRUTURA DE DADOS

3. **Registro da chegada:** o sistema registra as informações do container, como número de identificação, origem, destino, peso e conteúdo, juntamente com a data e hora de chegada. Esses detalhes são mantidos em um registro associado ao container.
4. **Empilhamento dos containers:** à medida que mais containers chegam, eles são adicionados à pilha, empilhados sobre os containers anteriores. Os containers mais recentes ficam no topo da pilha, enquanto os mais antigos estão na base.
5. **Acesso aos containers:** quando um container precisa ser removido ou inspecionado, a operação é realizada no container no topo da pilha, pois ele é o mais acessível. Após a remoção, a pilha é atualizada para refletir a nova ordem de containers.
6. **Operações de saída:** quando containers são enviados para seus destinos ou carregados em navios, eles são removidos da pilha de acordo com a ordem de saída. Os registros desses containers também são atualizados para registrar a data e hora da partida.
7. **Monitoramento e relatórios:** o sistema fornece um mecanismo para monitorar o estado da pilha e gerar relatórios sobre a ordem de chegada, saída e outros detalhes dos containers.

Concluímos que a utilização de uma pilha nesse cenário permite um controle eficiente da ordem de chegada e saída dos containers, simplificando o acesso aos containers mais recentes, o que é essencial para operações eficazes em um porto com bastante movimento.

Saiba mais

Para saber mais sobre a estrutura de dados pilha e suas principais operações, como a inserção e remoção de elementos, acesse a apostila: “*Pilhas e Filas*”, de Walteno Martins Parreira Júnior, professor do Instituto Federal do Triângulo Mineiro – IFTM. A apostila apresenta, de forma descomplicada, as estruturas de dados pilha e fila, com uma linguagem simples e exemplos que potencializam o aprendizado. Não deixe de conferir este material.

JUNIOR, W. P. M. [Algoritmos e Estrutura de Dados – Pilha e Fila](#). Instituto Federal do Triângulo Mineiro – IFTM. 2016.

Bons estudos!

Referências

CORMEN, T. **Algoritmos** – Teoria e Prática. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

ALGORITMOS E ESTRUTURA DE DADOS

TENENBAUM, A. M. **Estrutura de dados usando em C.** São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 3

Filas: Definição

Filas: definição

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe boas-vindas a mais uma aula desta Unidade de ensino; esperamos que este conteúdo possa contribuir muito com seu aprendizado sobre algoritmos e estrutura de dados.

Durante esta aula, vamos apresentar a você uma nova estrutura de dado: a fila. Você conhecerá e compreenderá os conceitos envolvidos com esta estrutura, suas características e também sua utilização prática em algoritmos.

Sabemos que uma fila é algo comum e que todas as pessoas têm contato com esta estrutura em algum momento do seu dia. As filas também podem incluir, além de suas regras intrínsecas, questões como prioridades e exceções, tanto na sua construção quanto na sua manutenção.

ALGORITMOS E ESTRUTURA DE DADOS

Para a compreensão dessa estrutura de dados, vamos sugerir o seguinte cenário: o gestor da empresa de software em que você trabalha atribuiu a você a tarefa de imaginar uma lava-rápido movimentado em uma área urbana; os proprietários de veículos frequentemente recorrem a ele para manter seus carros limpos e brilhantes. A operação do lava-rápido envolve uma fila de carros que aguardam atendimento.

O seu trabalho será propor uma solução sistêmica que gerencie as filas de carros e atenda aos clientes de forma eficiente e justa.

A solução deve abordar toda a logística envolvendo a chegada dos carros ao lava-rápido, até o momento da finalização do seu atendimento, ou seja, quando o veículo será removido da fila.

Depois de implementar uma solução para o problema proposto utilizando a estrutura de dado fila, você estará apto a implementar algoritmos que possam resolver problemas da realidade profissional com ela.

Convidamos você a aprofundar seus conhecimentos sobre filas e a entender como esta estrutura de dado pode auxiliar sua atividade técnica profissional em sistemas de informações.

Vamos Começar!

Os algoritmos de fila são utilizados em praticamente todas as regras de negócio no cenário organizacional. Por exemplo, podemos facilmente identificar essa estrutura em atendimentos no pronto socorro na área da saúde, ou quando estamos em um restaurante aguardando para nos servir, ou até mesmo após comprarmos algo pela internet (neste momento, uma fila, mesmo que invisível, se inicia, até que o produto seja entregue a você).

Antes de entendermos as características de construção e funcionamento da estrutura de dados fila, é útil considerá-la como uma lista. Para ilustrar esse conceito, imagine que você deseja se inscrever em uma vaga de desenvolvedor de sistemas aberta na empresa na qual você sempre quis entrar. A inscrição para a vaga requer que você siga uma lista de etapas. A primeira etapa é preencher uma ficha de inscrição para solicitar sua candidatura. A segunda etapa envolve anexar todos os documentos necessários à ficha de inscrição da vaga. Por fim, a terceira etapa é a efetivação da inscrição junto à empresa, seja por e-mail, seja por preenchimento de um formulário no site indicado. Observe que estamos lidando com uma lista de tarefas ou itens que precisam ser cumpridos. Vamos apresentar essas etapas em forma de lista:

1. Preencher a ficha de inscrição.
2. Anexar cópia dos documentos solicitados.
3. Efetivar a candidatura à vaga.

ALGORITMOS E ESTRUTURA DE DADOS

Cada item da lista contém informações implícitas sobre como cumprir o próximo item. Por exemplo, no primeiro item, que envolve o preenchimento da ficha de inscrição, já está implícito que você precisa anexar documentos necessários junto com a ficha. Assim, os procedimentos de itens subsequentes estão interligados. Esse tipo de lista é chamado de lista encadeada.

Para fornecer uma compreensão mais detalhada, podemos apresentar uma imagem que mostra a ordem de execução entre esses passos. Observe a Figura 1.

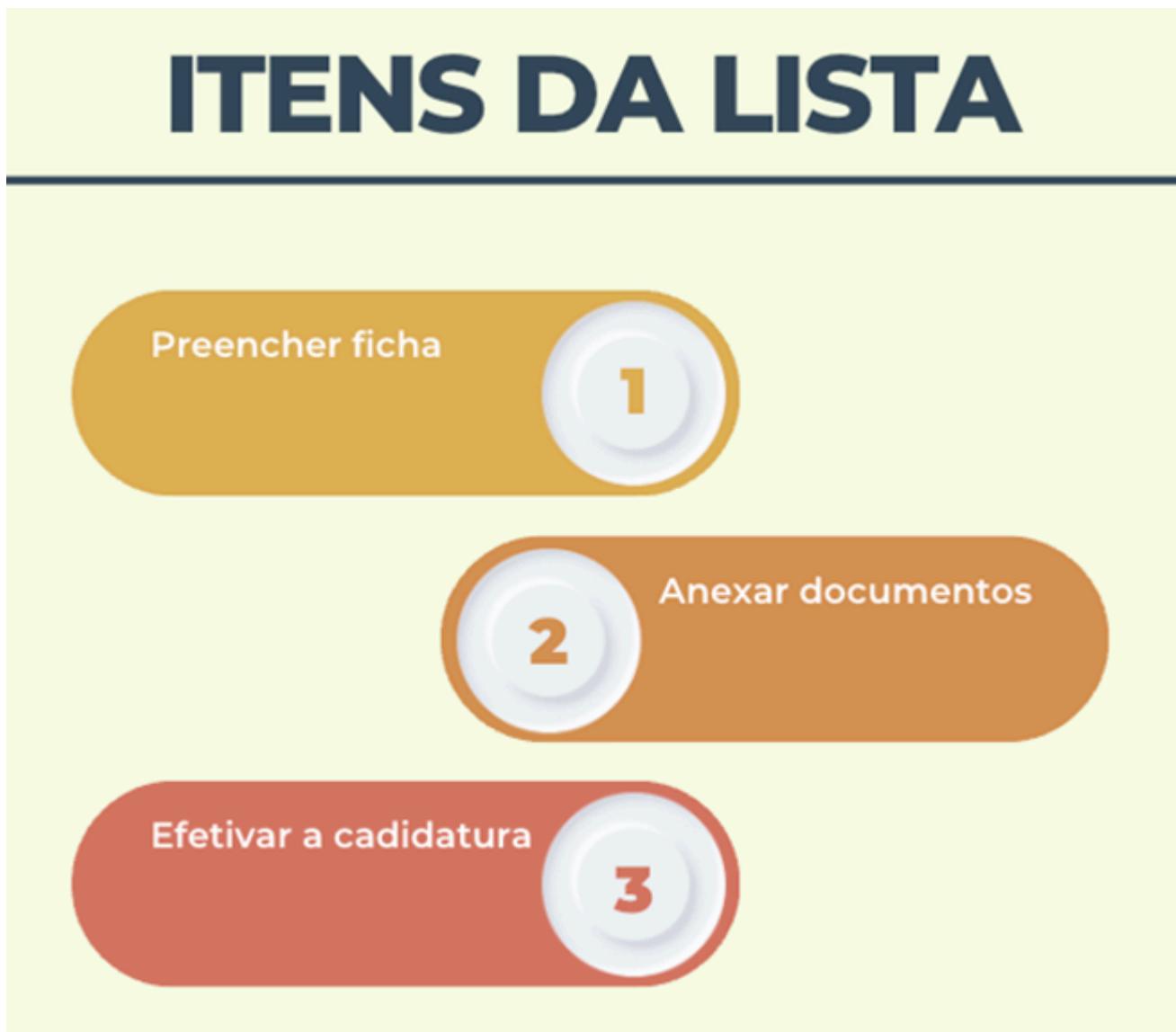


Figura 1 | Classe Pilha – método de inserção.

A Figura 1 mostra o fluxo dos itens na ordem em que devem ocorrer, indo de cima para baixo. Não é necessário a inclusão de fluxos de controles ou setas associando os itens, pois os números já deixam claro qual a ordem de execução das etapas. Será possível expressar o

ALGORITMOS E ESTRUTURA DE DADOS

mesmo encadeamento lógico sem a necessidade das setas na codificação, uma vez que elas não existem na implementação do código-fonte.

Em essência, tanto o princípio da fila quanto o da pilha são semelhantes, já que ambas são estruturas derivadas de uma lista. Embora haja diferenças em como essas estruturas funcionam e como manipulam os valores, ambas compartilham a característica comum de serem estruturas de armazenamento de valores.

- **Lista ordenada:** como o nome sugere, essas são listas organizadas com base em critérios específicos. Elas têm uma característica importante a ser ressaltada: quando um novo elemento é adicionado, ele deve ocupar uma posição específica de acordo com o critério predefinido (Cormen, 2022). Vamos analisar um exemplo:

1. Exemplo: lista de números decimais em ordem crescente: {2, 4, 6, 7, 9}. Se desejarmos adicionar o número 5, ele deve necessariamente ocupar a posição entre o número quatro e o número seis, seguindo o critério de números decimais em ordem crescente. Isso resultaria na lista da seguinte maneira: {2, 4, **5**, 6, 7, 9}.

- **Lista desordenada:** a lista desordenada é um conjunto de elementos de qualquer tipo que ainda não passou por nenhum processo de ordenação, ou, ainda, que foi mantida fora de ordem de forma proposital. Vamos analisar um exemplo:

2. Exemplo: aqui temos uma lista de números decimais: {22, 17, 32, -1, 89}. Trata-se de uma lista numérica que está fora de ordem de propósito. Nesse cenário, não buscamos realizar a ordenação, pois o objetivo é justamente manter a lista classificada como desordenada.

Estrutura da fila

A partir de agora, vamos explorar a estrutura de dados conhecida como fila, que faz parte das listas disciplinadas. A fila é considerada uma lista linear disciplinada devido às suas regras específicas para manipulação de elementos. Essas regras incluem a adição de novos elementos à lista, a remoção de elementos da lista e a pesquisa de um ou vários elementos na lista (Cormen, 2022).

A fila, em essência, segue a política FIFO (*First In, First Out*), o que significa que o primeiro elemento a entrar na fila é o primeiro a sair, como é comum em muitas situações do nosso dia a dia. Você pode encontrar filas físicas em locais como balcões de atendimento, bancos financeiros, postos de combustível e muitas outras situações cotidianas.

Quanto às regras de manipulação em uma fila, estas incluem:

- Inserção de elementos efetuadas no final da fila.
- Remoções de elementos a partir do início da fila.

ALGORITMOS E ESTRUTURA DE DADOS

- A consulta na fila é sequencial, ou seja, é realizada elemento por elemento, até que o valor desejado seja encontrado ou até que a fila seja percorrida inteiramente.

Um exemplo ilustrativo de uma fila de pessoas pode ser visto na Figura 2. Observe que as pessoas estão viradas na direção da fila, indicando o sentido em que ela se move. Pode-se concluir que uma mulher com uma bolsa é a primeira da fila, enquanto o homem com um fone de ouvido colocado no pescoço é o último da fila.



Figura 2 | Fila de pessoas. Fonte: Adobe Stock.

A Figura 2 oferece uma representação visual que ilustra o funcionamento das regras aplicadas nos algoritmos de fila. Esses algoritmos operam de maneira semelhante à dinâmica observada na fila de pessoas. Vamos abordar agora as formas de implementação dessas listas por meio de alguns exemplos.

Siga em Frente...

Implementação de filas em Java: vetores *versus* listas

ALGORITMOS E ESTRUTURA DE DADOS

A estrutura de dados de fila é amplamente utilizada na programação para gerenciar elementos de forma ordenada, de modo que o primeiro a entrar é o primeiro a sair. Em Java, existem várias maneiras de implementar filas, com duas abordagens principais sendo a utilização de vetores e listas. Ambas as implementações têm suas próprias características e vantagens, tornando-as adequadas para diferentes cenários de aplicação.

Fila com vetores em Java

A implementação de fila baseada em vetores é uma das abordagens mais simples e eficientes para criar uma fila em Java. Um vetor é uma estrutura de dados de tamanho fixo que armazena elementos de forma contígua na memória. Quando um novo elemento é adicionado à fila, ele é inserido no final do vetor, e a remoção ocorre no início do vetor. Isso segue o princípio FIFO (*First In, First Out*) de uma fila.

Uma das classes fornecidas pelo Java para implementação de filas com vetores é a *ArrayDeque*. Essa classe oferece eficiência na adição e remoção de elementos, com complexidade de tempo $O(1)$. Isso significa que as operações de inserção e remoção são muito rápidas, independentemente do tamanho da fila.

No entanto, uma desvantagem da implementação de fila baseada em vetor é que o tamanho da fila é fixo e deve ser definido durante a inicialização. Isso pode limitar a flexibilidade em cenários em que o tamanho da fila precisa ser dinâmico. Além disso, é importante gerenciar o redimensionamento do vetor, se a fila estiver se aproximando do estouro de capacidade ou se tornando muito pequena para uso eficiente da memória.

A Figura 3 apresenta um código em Java da criação de uma fila utilizando vetores. Observe a seguir.

ALGORITMOS E ESTRUTURA DE DADOS

```
1. public class FilaComVetor {  
2.     private int[] fila;  
3.     private int tamanhoMaximo;  
4.     private int tamanhoAtual;  
5.  
6.     public FilaComVetor(int tamanhoMaximo) {  
7.         this.tamanhoMaximo = tamanhoMaximo;  
8.         this.fila = new int[tamanhoMaximo];  
9.         this.tamanhoAtual = 0;  
10.    }  
11.  
12.    public boolean estaVazia() {  
13.        return tamanhoAtual == 0;  
14.    }  
15.  
16.    public boolean estaCheia() {  
17.        return tamanhoAtual == tamanhoMaximo;  
18.    }  
19.  
20.    public int tamanho() {  
21.        return tamanhoAtual;  
22.    }  
}
```

Figura 3 | Código utilizando fila com vetores.

ALGORITMOS E ESTRUTURA DE DADOS

Este código Java cria uma classe “FilaComVetor”, que possui os atributos “fila”, “tamanhoMaximo” e “tamanhoAtual”, bem como os métodos para verificar se a fila está vazia, cheia e obter seu tamanho. Não são abordadas operações específicas de manipulação da fila neste exemplo, somente a sua estrutura.

Fila com listas em Java

As listas encadeadas em Java oferecem uma alternativa flexível para implementar filas. Em vez de armazenar elementos de forma contígua em memória, uma lista encadeada consiste em “nós” que contêm um valor e uma referência ao próximo elemento na fila. Isso permite a criação de filas de tamanho dinâmico, sem a necessidade de especificar um tamanho máximo durante a inicialização.

Uma das classes de lista encadeada em Java é a *LinkedList*. Essa classe é frequentemente usada para criar filas baseadas em listas. A inserção em uma lista encadeada também tem complexidade $O(1)$, tornando-a eficiente. Além disso, a remoção no início da lista tem a mesma complexidade de tempo, $O(1)$.

A flexibilidade das listas encadeadas torna-as adequadas para cenários em que o tamanho da fila pode variar dinamicamente. Por exemplo, em sistemas de gerenciamento de tarefas, nos quais as tarefas podem ser adicionadas ou removidas a qualquer momento, as filas baseadas em listas se tornam a melhor escolha.

A Figura 4 apresenta um código em Java da criação de uma fila utilizando listas. Observe a seguir.

ALGORITMOS E ESTRUTURA DE DADOS

```
1. import java.util.LinkedList;
2.
3. public class FilaComLista {
4.     private LinkedList<Integer> fila;
5.     private int tamanhoMaximo;
6.
7.     public FilaComLista(int tamanhoMaximo) {
8.         this.tamanhoMaximo = tamanhoMaximo;
9.         this.fila = new LinkedList<>();
10.    }
11.
12.    public boolean estaVazia() {
13.        return fila.isEmpty();
14.    }
15.
16.    public boolean estaCheia() {
17.        return fila.size() == tamanhoMaximo;
18.    }
19.
20.    public int tamanho() {
21.        return fila.size();
22.    }
}
```

Figura 4 | Código utilizando fila com listas.

ALGORITMOS E ESTRUTURA DE DADOS

Neste exemplo, a classe “FilaComLista” utiliza uma lista encadeada “*LinkedList*” para representar a fila. Os métodos “estaVazia()”, “estaCheia()” e “tamanho()” foram implementados para verificar o estado da fila, mas não são abordadas operações específicas de manipulação da fila.

Escolhendo entre vetores e listas

A escolha entre uma implementação de fila baseada em vetor ou em lista na linguagem Java depende das necessidades específicas do aplicativo. Se a fila tiver um tamanho fixo e a eficiência nas operações de adição e remoção for aceitável, a implementação com vetores é adequada. Essa é uma escolha comum em aplicativos nos quais a velocidade é um fator crítico, como sistemas de gerenciamento de tarefas em tempo real.

Por outro lado, se a fila precisa crescer dinamicamente ou se for necessário um maior grau de flexibilidade, a implementação com listas encadeadas é uma opção viável. Isso é particularmente útil em cenários em que o tamanho da fila é desconhecido ou pode variar significativamente.

Em resumo, a escolha entre vetores e listas para implementar filas em Java deve ser feita com base nas necessidades específicas do aplicativo. Ambas as abordagens têm suas vantagens e desvantagens, e a seleção depende das demandas do problema em questão. A compreensão das características de cada implementação é fundamental para tomar a decisão mais adequada.

Esperamos que este conteúdo mais uma vez tenha contribuído com seu aprendizado e agregado valor a seu conhecimento.

Vamos Exercitar?

Vamos retomar o cenário proposto no início da aula! O proprietário do lava-rápido deseja implementar um sistema de gerenciamento de filas que permita atender os clientes de forma eficiente e justa. A operação do lava-rápido segue estas condições:

- Os carros chegam ao lava-rápido em momentos diferentes e são adicionados à fila na ordem de chegada.
- O lava-rápido possui um número limitado de baias de lavagem, nas quais os carros são atendidos.
- Cada tipo de serviço leva um tempo diferente para ser concluído, variando desde uma lavagem simples até uma limpeza completa.
- O objetivo é garantir que os carros sejam atendidos o mais rapidamente possível e que nenhum cliente espere por muito tempo na fila.

O seu desafio é projetar um algoritmo que ajude o lava-rápido a gerenciar sua fila de maneira eficaz, priorizando o atendimento com base na ordem de chegada e no tipo de serviço solicitado.

ALGORITMOS E ESTRUTURA DE DADOS

O sistema deve calcular o tempo de espera estimado para cada cliente e determinar qual carro deve ser atendido em seguida.

Agora você, como um profissional na área de desenvolvimento de sistemas, analise o problema e proponha pelo menos uma solução que atenda à demanda.

Na sequência estão descritos os passos para implementação do sistema de gerenciamento do lava-rápido utilizando a estrutura fila.

1. **Inicialização:** comece com uma fila vazia que representará a fila de carros a serem atendidos.
2. **Chegada de carros:** à medida que os carros chegam ao lava-rápido, adicione-os à fila, respeitando a ordem de chegada. Registre o tipo de serviço solicitado e o tempo estimado para a conclusão.
3. **Disponibilidade de baias:** monitore as baias de lavagem disponíveis. Quando uma baia se tornar livre, verifique se há carros na fila.
4. **Seleção de carro:** determine qual carro deve ser atendido a seguir com base em um critério. Pode ser uma fila de prioridade com base no tempo de espera estimado ou no tipo de serviço.
5. **Atendimento do carro:** realize o serviço no carro selecionado e registre o tempo real de conclusão.
6. **Remoção da fila:** remova o carro atendido da fila.
7. **Cálculo do tempo de espera:** recalcule o tempo de espera estimado para os carros restantes na fila.
8. **Repetição:** continue monitorando a chegada de carros, a disponibilidade de baias e o atendimento, até que todos os carros tenham sido atendidos.

Esse sistema ajudará o lava-rápido a gerenciar a fila de carro, atendendo todos os clientes de forma justa e eficiente.

Portanto, a utilização de uma estrutura de fila nesse contexto permite um controle eficiente da ordem de chegada dos carros e de todos os passos até que eles sejam removidos da fila.

Saiba mais

Para saber mais sobre a estrutura de dado fila e seus principais conceitos e características, acesse o artigo: "*Análise de eficiência na utilização dos recursos de transmissão de dados com uso de algoritmos de filas de prioridade*", de Dinailton José da Silva, publicado na Revista de Ciências Exatas e Tecnologia – vol. III, nº 3. O artigo apresenta, por meio da teoria e da prática, algumas funcionalidades e as características do modelo qualidade de serviço (QoS – *Quality of Service*) baseado no algoritmo de filas de prioridade aplicáveis às redes corporativas. Não deixe de conferir esse artigo e aprender com ele.

ALGORITMOS E ESTRUTURA DE DADOS

SILVA, D. J. da. [Análise de eficiência na utilização dos recursos de transmissão de dados com uso de algoritmos de filas de prioridade](#). Revista de Ciências Exatas e Tecnologia – vol. III, nº 3. Faculdade Anhanguera, Anápolis, GO, 2008.

Bons estudos!

Referências

CORMEN, T. **Algoritmos** – Teoria e Prática. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SILVA, D. J. Análise de Eficiência na Utilização dos Recursos de Transmissão de Dados com Uso de Algoritmos de Filas de Prioridade. **Revista de Ciências Exatas e Tecnologia**, Anápolis, vol. III, nº 3, p. 27- 49, 2008. Disponível em:

<https://exatastecnologias.pgscogna.com.br/rcext/article/view/2367>. Acesso em: 20 out. 2023.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 4

Filas: Implementação e Operações

Filas: implementação e operações

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

ALGORITMOS E ESTRUTURA DE DADOS

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Vamos iniciar nossos estudos nesta aula continuando a trabalhar com estrutura de dados. Desejamos que este material contribua de forma significativa para o enriquecimento do seu conhecimento.

Durante esta aula, vamos apresentar a você um pouco mais sobre a estrutura de dado fila.

Você conhecerá e compreenderá algumas operações que essa estrutura nos permite fazer. Entre essas operações se encontram:

- Inserção de elementos em uma fila.
- Remoção de elementos em uma fila.
- Verificação da fila vazia: operação que testa quando a fila está vazia, para que não se apresente erro na remoção de elementos.
- Verificação da fila cheia: operação que testa quando a fila está cheia, para que não se apresente erro na inserção de elementos.

Para a aplicação dessas operações, vamos considerar o cenário a seguir: em um pronto-socorro de um hospital movimentado, a organização da fila de atendimento é fundamental para garantir que os pacientes recebam a assistência médica de que precisam. A fila segue uma ordem de chegada, mas também considera atendimento preferencial para mulheres grávidas e idosos.

Esta situação aborda todo o controle gerencial da fila. Os elementos dessa estrutura são os pacientes que chegam e precisam aguardar atendimento.

Após a implementação das operações de fila que serão abordadas ao longo desta aula, você estará apto a implementar codificações que possam resolver esses e outros problemas.

Encorajamos você a explorar com mais profundidade o conceito de filas e a compreender como essa estrutura de dados pode ser uma ferramenta valiosa em sua carreira técnica, especialmente em sistemas de informação. Ótimos estudos!

ALGORITMOS E ESTRUTURA DE DADOS

Vamos Começar!

O objetivo desta aula é abordar claramente as operações de inserção e remoção de elementos em uma fila, além de apresentar funções que verificam se a fila está vazia ou cheia para evitar erros na execução do algoritmo.

Dentro do contexto da fila, empregamos expressões específicas para descrever suas operações. Quando adicionamos um novo elemento à fila, nos referimos a esse processo como “enfileiramento”, correspondente ao verbo “enfileirar”. Por outro lado, quando retiramos um elemento já presente na fila, denominamos isso como “desenfileiramento”, correspondente ao verbo “desenfileirar”. Esses termos desempenham um papel fundamental na descrição das operações realizadas nesta estrutura de dados.

Vamos detalhar um pouco mais sobre as regras de manipulação, iniciando pela inserção de elementos na fila.

Inserção de elementos na fila

Dado que a fila segue a política FIFO (*First In First Out*), compreendemos que a inserção de um novo elemento na fila ocorre sempre no seu final. Isso é normalmente alcançado através de uma variável de controle que mantém o registro da posição do último elemento na fila (Cornem, 2022).

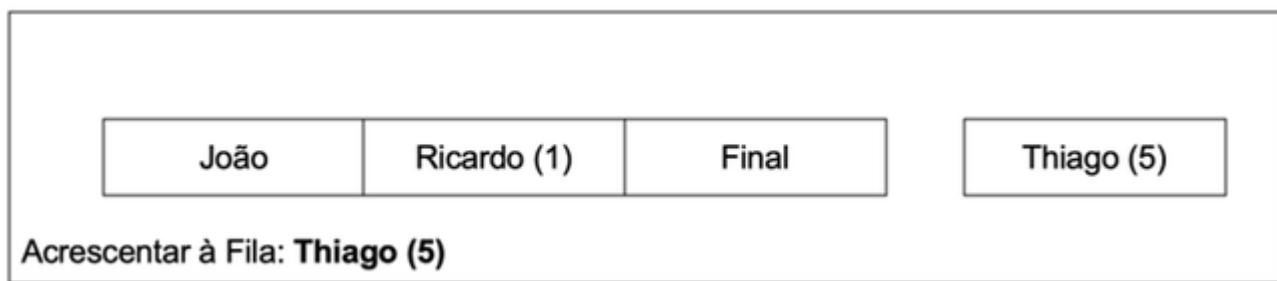


Figura 1 | Inserção (parte I).

Na Figura 1, temos os elementos João e Ricardo na fila: {"João", "Ricardo"}, enquanto o elemento Thiago aguarda sua vez para entrar nela. Repare que, neste momento, ele ainda não se encontra na estrutura.

Nesse cenário, a extremidade final da fila está apontando para Ricardo. Para inserir o Thiago na fila, são necessários dois passos. O primeiro é fazer com que Ricardo aponte para Thiago e, em seguida, o próximo passo é fazer com que a extremidade final, que antes apontava para Ricardo, aponte agora para Thiago. Observe a Figura 2.

ALGORITMOS E ESTRUTURA DE DADOS

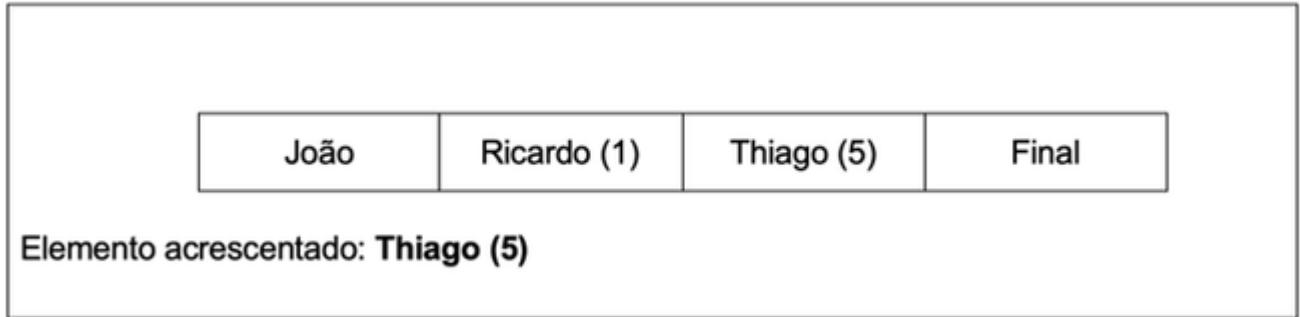


Figura 2 | Inserção (parte II).

Para efetuar a adição de um elemento à fila, é fundamental determinar qual elemento será incluído. É importante destacar que sempre é necessário executar ambos os processos ao inserir elementos na fila. Em uma perspectiva algorítmica, ao analisar as instruções em linha de comando, podemos observar como esses dois processos são realizados (Cormen, 2022):

- 1º Passo: *fila[1].prox = 5.*
- 2º Passo: *final = 5.*

Agora que você tem uma compreensão sobre o processo de criação e adição de elementos em uma fila, avançaremos em nosso estudo, explorando como remover elementos da estrutura de fila.

Remoção de elementos na fila

O procedimento de remoção de um elemento na lista segue uma lógica inversa à da inserção. Enquanto para adicionar um elemento à lista é necessário identificar o final da lista, ao realizar uma remoção, é essencial localizar o elemento no início da fila (Cormen, 2022).

O comportamento FIFO, como já mencionado, valida essa regra. Portanto, a remoção ocorre no início da fila, da mesma forma que em uma fila convencional de pessoas. Se alguém chegou primeiro, será atendido e retirado da fila antes de todos os outros (Cormen, 2022).

A seguir, apresentaremos um exemplo de remoção de um elemento da lista. Através desse exemplo, será possível desenvolver um algoritmo que coloque em prática essa operação.

ALGORITMOS E ESTRUTURA DE DADOS

Início: João (1)

João (1)	Ricardo (5)	Thiago (8)	Final
----------	-------------	------------	-------

Elemento que será excluído: **João (1)**

Figura 3 | Exclusão (parte I).

Na Figura 3, encontramos os elementos João, Ricardo e Thiago na fila, representada como {"João", "Ricardo", "Thiago"}. Suponhamos que o primeiro da fila, João, tenha sido atendido e agora será removido. Após a sua retirada, o início da fila passará a ser o elemento "Ricardo". Para realizar esse procedimento, é necessário atualizar o apontador do início da fila, fazendo-o apontar para o segundo elemento da fila, que é Ricardo. Dessa forma, João é retirado da fila, como ilustrado na Figura 4.

Início: Ricardo (5)

João (1)	Ricardo (5)	Thiago (8)	Final
----------	-------------	------------	-------

Elemento excluído: **João (1)**

Figura 4 | Exclusão (parte II).

No algoritmo de fila, não realizamos a exclusão direta do elemento, ou seja, não apagamos o "João" da lista. Em vez disso, o que fazemos é alterar o apontador para o próximo elemento na fila. Mais uma vez, em uma perspectiva algorítmica em um ambiente de linha de comando, podemos observar as etapas executadas durante a remoção de um elemento da fila. Observe a seguir:

- 1º Passo: *inicio = fila[5].prox.*

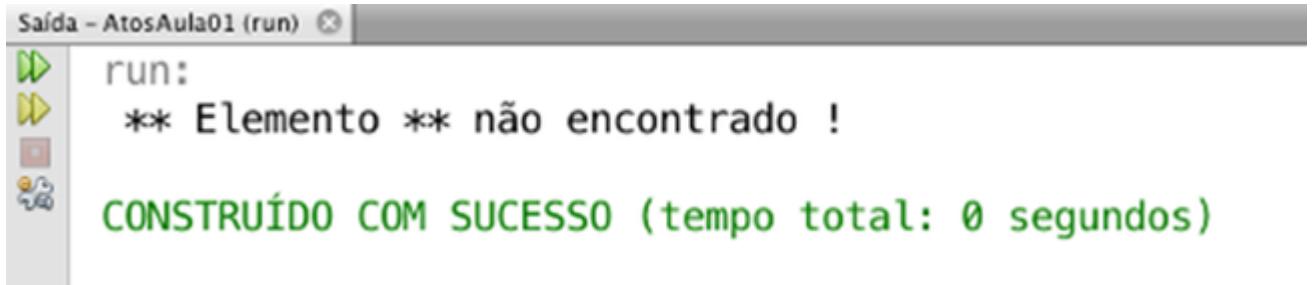
Já apresentamos como adicionar e remover elementos de uma fila. Agora, vamos prosseguir com nossa exploração, desta vez abordando a operação de consulta em relação a um elemento na estrutura.

Pesquisa de um elemento na fila

ALGORITMOS E ESTRUTURA DE DADOS

Para encontrar elementos na fila, você pode utilizar algoritmos de busca; por exemplo, a busca sequencial. Esta é uma opção viável, especialmente porque as filas geralmente não são ordenadas. A ordem em uma fila é tipicamente determinada pela ordem de entrada dos elementos ou, então, por prioridades definidas conforme o cenário.

Para procurar um elemento na fila, você pode percorrê-la, comparando o valor de busca com os elementos da fila. Se o valor for encontrado, ele é exibido no programa. No entanto, se o valor não for encontrado em nenhum elemento da fila, isso indica que ele não existe. Nesse caso, você pode mostrar uma mensagem ao usuário informando que o elemento não foi encontrado, como exemplificado na Figura 5.



```
run:  
** Elemento ** não encontrado !  
  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Figura 5 | Mensagem na pesquisa de elementos.

Siga em Frente...

Fila vazia

Antes de excluir elementos da fila, é essencial verificarmos se a fila contém pelo menos um elemento. Caso contrário, seria como tentar excluir um cliente que não está cadastrado, ou alterar o valor de um produto inexistente.

Tentar remover elementos de uma fila vazia não é permitido; é importante realizar essa verificação para evitar erros. Um código que faz essa verificação de forma correta é ilustrado na Figura 6.

ALGORITMOS E ESTRUTURA DE DADOS

```
public class Fila {  
  
    private List<Fila> alunos = new LinkedList<Fila>();  
  
    public void insere(Aluno aluno) {  
        // implementação  
    }  
  
    public Aluno remove() {  
        // implementação  
        return null;  
    }  
  
    public boolean vazia() {  
  
        return this.alunos.size() == 0;  
    }  
}
```

Figura 6 | Codificação da fila.

Na Figura 6, temos a definição da classe Fila, que inclui três métodos principais: “insere()”, “remove()” e “vazia()”. O método “vazia()” tem a funcionalidade de retornar verdadeiro quando o número de elementos na fila for igual a zero. Isso significa que, enquanto o método “vazia()” retornar falso, é permitido prosseguir com a remoção de elementos da fila, indicando que ainda não está vazia.

Podemos visualizar um exemplo da execução deste método na Figura 7.

ALGORITMOS E ESTRUTURA DE DADOS

```
12  class Aluno {  
13  
14      public static void main(String[] args) {  
15  
16          Fila f = new Fila();  
17          boolean ret = f.vazia();  
18  
19  
20          if(f.vazia()){  
21              System.out.println("Exclusão não permitida! Fila vazia ");  
22          }  
23  
24      }  
25  
26  }  
27 }  
28 }  
29 }
```

Figura 7 | Fila vazia.

Na Figura 7, a classe Aluno cria uma instância da classe Fila com o propósito de utilizar o método “vazia()” pertencente a esta última. Quando esse método retorna verdadeiro, uma mensagem específica é exibida como resultado, conforme visto no comando de saída da linha 22, na Figura 8.



The screenshot shows the 'Saída - AtosAula01 (run)' window of a Java IDE. It displays the command 'run:' followed by the output text 'Exclusão não permitida! Fila vazia' in red, indicating an error. Below this, the text 'CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)' is shown in green, indicating a successful build.

Figura 8 | Execução do algoritmo.

Para finalizar o conteúdo, vamos apresentar um código em Java para inserção de uma fila com três elementos. Observe a Figura 9.

ALGORITMOS E ESTRUTURA DE DADOS

```
1. import java.util.LinkedList;
2. import java.util.Queue;
3.
4. public class Main {
5.     public static void main(String[] args) {
6.         // Cria uma fila (Queue) para armazenar os elementos
7.         Queue<String> fila = new LinkedList<>();
8.
9.         // Inserindo elementos na fila
10.        fila.offer("Anderson"); // Insere "Anderson"
11.        fila.offer("Rômulo"); // Insere "Rômulo"
12.        fila.offer("Fernanda"); // Insere "Fernanda"
13.
14.        // Imprimindo a fila para verificar os elementos inseridos
15.        int i = 1;
16.        for (String elemento : fila) {
17.            System.out.println("Elemento " + i + ": " + elemento);
18.            i++;
19.        }
20.    }
21. }
```

Figura 9 | Inserção de elementos na fila.

Logo abaixo se encontra o passo a passo da codificação exibida na Figura 9.

1. Importamos as classes necessárias do pacote `java.util`, que incluem `LinkedList` e `Queue`. `LinkedList` é uma estrutura de dados usada para implementar uma fila, e `Queue` é uma interface que define métodos comuns para operações de fila.
2. Começamos a definição da classe principal, chamada `Main`.
3. No método `main`, que é o ponto de entrada do programa, iniciamos a execução.
4. Criamos uma fila (Queue) chamada `fila` usando a implementação `LinkedList`. A fila será usada para armazenar elementos do tipo `String`.
5. Iniciamos a inserção de elementos na fila usando o método `offer()` da fila.
6. `fila.offer("Anderson")`: Insere o elemento "Anderson" na fila.
7. `fila.offer("Rômulo")`: Insere o elemento "Rômulo" na fila.

ALGORITMOS E ESTRUTURA DE DADOS

8. `fila.offer("Fernanda")`: Insere o elemento “Fernanda” na fila.
9. Iniciamos um *loop for-each* para percorrer os elementos na fila.
10. Para cada elemento na fila, imprimimos uma mensagem que indica o número do elemento e o valor do elemento. O número do elemento é incrementado à medida que percorremos a fila.
11. O *loop* continua até que todos os elementos na fila tenham sido impressos.

No final, a saída do programa exibe os elementos inseridos na fila com seus respectivos números. O resultado final é apresentado na Figura 10.

```
Elemento 1: Anderson
Elemento 2: Rômulo
Elemento 3: Fernanda
```

Figura 10 | Tela de saída do programa.

Desejamos que os tópicos explicados nesta aula tenham se tornado compreensíveis por meio dos exemplos apresentados. Nosso objetivo foi fornecer informações que contribuíssem para a ampliação de seu entendimento sobre estruturas de fila.

Vamos Exercitar?

Vamos retomar o cenário proposto no início da aula! A equipe de atendimento do pronto-socorro recebe pacientes em ordem de chegada e mantém uma fila para garantir que todos sejam atendidos de maneira justa. No entanto, o hospital tem uma política de atendimento preferencial para mulheres grávidas e idosos (pessoas com mais de 60 anos).

Suponha que, em determinado momento, a fila de espera do pronto-socorro seja a seguinte:

1. João: 35 anos.
2. Maria: 28 anos (grávida).
3. José: 67 anos (idoso).
4. Ana: 42 anos.
5. Luis: 25 anos.
6. Sofia: 74 anos (idoso).
7. Carlos: 50 anos.

Considere que, de acordo com a política de atendimento preferencial, sempre que uma mulher grávida ou um idoso chega ao pronto-socorro, ele ou ela é imediatamente encaminhado para o

ALGORITMOS E ESTRUTURA DE DADOS

atendimento, independentemente da ordem de chegada. Além disso, após o atendimento de um paciente preferencial, a equipe médica retorna à fila para atender o próximo paciente.

Com base na descrição da situação-problema, utilizando a estrutura de dados fila, apresente uma solução para atender esta demanda.

Na sequência estão descritos os passos do cenário para atendimento do pronto-socorro, considerando para isso valores reais presentes na solução. Os valores são os mesmos informados na descrição da situação-problema.

1. Inicialmente, a equipe de atendimento verifica a fila de pacientes e observa que Maria, uma mulher grávida, e José, um idoso, estão presentes na fila. De acordo com a política de atendimento preferencial, Maria será atendida imediatamente, seguida por José.
2. Maria é atendida e recebe o tratamento médico necessário.
3. Após o atendimento de Maria, a equipe médica retorna à fila e inicia o atendimento de acordo com a ordem de chegada. Portanto, João é o próximo a ser atendido.
4. O atendimento de João é concluído.
5. Novamente, a equipe verifica a fila e observa que Sofia, outra idosa, está presente. De acordo com a política de atendimento preferencial, Sofia é imediatamente encaminhada para o atendimento.
6. Sofia é atendida e recebe o tratamento médico necessário.
7. Após o atendimento de Sofia, a equipe retorna à fila e continua atendendo os pacientes com base na ordem de chegada, começando por Ana, Luis e, por fim, Carlos.

Chegamos à conclusão de que o emprego de uma fila nesta situação possibilita um gerenciamento eficaz da sequência de atendimento dos pacientes, facilitando o trabalho dos médicos ao decidir qual paciente deverão atender primeiro. Isso é essencial para um atendimento seguro, eficaz e justo em qualquer pronto-socorro.

Saiba mais

Se você quiser se aprofundar um pouco sobre a estrutura de dado fila e suas principais operações para a criação de algoritmos, acesse o artigo: "Gerência ativa de filas usando redes neurais LSTM para fluxos DASH ao vivo", de Carlos Eduardo Maffini Santos e Carlos Marcelo Pedroso, da Universidade Federal do Paraná (UFPR). O artigo apresenta um novo método AQM para melhorar a qualidade percebida do usuário em transmissões DASH de vídeo ao vivo. Não deixe de conferir esse artigo para contribuir com seu aprendizado.

SANTOS, C. E. M; PEDROSO, C. M. [Gerência Ativa de Filas usando Redes Neurais LSTM para Fluxos DASH ao Vivo](#). XXXIX Simpósio Brasileiro de Telecomunicações e Processamento de Sinais, Fortaleza, 09, 2021.

Bons estudos!

ALGORITMOS E ESTRUTURA DE DADOS

Referências

CORMEN, T. **Algoritmos – Teoria e Prática.** 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos.** São Paulo: Grupo GEN, 2013. E-book.

SANTOS, C. E. M; PEDROSO, C. M.. Gerência Ativa de Filas usando Redes Neurais LSTM para Fluxos DASH ao Vivo. **XXXIX Simpósio Brasileiro de Telecomunicações e Processamento de Sinais**, Fortaleza, 09, 2021. Disponível em:

<https://www.eletrica.ufpr.br/pedroso/Artigos/SBRT2021.pdf>. Acesso em: 20 out. 2023.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos.** 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C.** São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 5

Encerramento da Unidade

Videoaula de Encerramento

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

ALGORITMOS E ESTRUTURA DE DADOS

Ponto de Chegada

Olá, estudante! Para desenvolver a competência desta Unidade, que é “compreender os conceitos fundamentais de pilha e fila, operações básicas e como elas diferem de outras estruturas de dados”, você deverá primeiramente conhecer os conceitos fundamentais abordados ao longo da Unidade de ensino.

Uma das competências essenciais é a capacidade de entender e identificar as situações em que as pilhas e filas são empregadas.

O algoritmo de pilha permite gerenciar dados de forma eficiente, possibilitando empilhar e desempilhar informações, o que é útil para operações de rastreamento, chamadas de função e controle de histórico de tarefas. Já o algoritmo de fila permite uma gestão eficaz de dados, possibilitando a organização e recuperação de informações na ordem de chegada, ideal para tarefas como escalonamento de processos e agendamento de tarefas.

À medida que avançamos no conteúdo, confrontamos problemas do mundo real nos quais pilhas e filas demonstram eficácia. Você adquiriu a habilidade de avaliar o uso dessas estruturas em várias situações, desde a concepção até a manipulação de diferentes tipos de elementos.

Essas aptidões não apenas o ajudarão, como estudante, a criar algoritmos mais otimizados, mas também estabelecerão uma base sólida para lidar com desafios complexos no campo do desenvolvimento de software. Ao compreender quando e como aplicar as estruturas de pilhas e filas, você estará preparado para desenvolver soluções sólidas e escaláveis em sua carreira profissional na área de tecnologia.

É Hora de Praticar!



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Convidamos você a colocar em prática o conhecimento adquirido nesta Unidade de ensino. Para isso, vamos abordar um estudo de caso com o cenário descrito a seguir: **implementação da Torre de Hanói utilizando pilha**.

ALGORITMOS E ESTRUTURA DE DADOS



Figura 1 |Torre de Hanói. Fonte: Shutterstock.

Cenário: você foi desafiado a implementar o famoso quebra-cabeça da Torre de Hanói, que envolve a movimentação de três discos de diferentes tamanhos entre três pinos. O desafio é realizar essa tarefa utilizando uma estrutura de dados de pilha.

Objetivo: o objetivo do estudo de caso é demonstrar como o algoritmo de pilha pode ser aplicado para resolver o quebra-cabeça da Torre de Hanói, movendo os discos de um pino para outro de acordo com regras específicas.

Regras da Torre de Hanói:

1. Os discos são empilhados em ordem decrescente de tamanho, com o disco maior na parte inferior.
2. Apenas um disco pode ser movido por vez.
3. Um disco nunca pode ser colocado sobre outro menor.

Crie sua solução algorítmica de preferência na linguagem de programação Java. Bom trabalho!

- Você já refletiu sobre como as estruturas de pilha e fila são aplicadas em cenários do mundo real e como elas contribuem para a eficiência de operações específicas? Quais situações em que o uso de pilhas e filas é crucial?
- Considerando a natureza das pilhas e filas, reflita quais as principais diferenças entre essas estruturas e como suas características únicas influenciam a escolha de uma ou outra em diferentes contextos de programação. Quais situações em que uma estrutura é mais apropriada do que a outra?

Sugestão de solução

Passo 1: preparação - primeiro, criamos três pilhas, uma para cada pino da Torre de Hanói. Os discos são representados por números inteiros, com os números menores representando os discos maiores.

```
Pilha<Integer> pinoA = new Pilha<>();  
Pilha<Integer> pinoB = new Pilha<>();  
Pilha<Integer> pinoC = new Pilha<>();
```

Passo 2: Preenchimento Inicial - preenchemos o pino A com os três discos no início.

ALGORITMOS E ESTRUTURA DE DADOS

```
pinoA.push(3);
pinoA.push(2);
pinoA.push(1);
```

Passo 3: Resolvendo a Torre de Hanói com Pilha - agora, implementamos o algoritmo para mover os discos da Pilha A para a Pilha C, utilizando a Pilha B como auxiliar.

```
public static void resolverTorreHanoi(int n, Pilha<Integer> origem, Pilha<Integer> destino,
Pilha<Integer> auxiliar) {
    if (n > 0) {
        // Move n-1 discos da origem para a auxiliar usando o destino como auxiliar
        resolverTorreHanoi(n - 1, origem, auxiliar, destino);

        // Move o disco superior da origem para o destino
        int disco = origem.pop();
        destino.push(disco);
        System.out.println("Mover disco " + disco + " de " + origem + " para " + destino);

        // Move os n-1 discos da auxiliar para o destino usando a origem como auxiliar
        resolverTorreHanoi(n - 1, auxiliar, destino, origem);
    }
}

// Chama a função para resolver a Torre de Hanói
resolverTorreHanoi(pinoA.size(), pinoA, pinoC, pinoB);
```

Este algoritmo move os discos da Pilha A para a Pilha C, respeitando todas as regras da Torre de Hanói e usando a Pilha B como auxiliar.

A implementação da Torre de Hanói com o uso de uma estrutura de pilha demonstra a versatilidade dessa estrutura de dados na resolução de problemas complexos. Nesse cenário, a pilha é usada para manter o controle dos discos, garantindo que as regras do quebra-cabeça sejam seguidas corretamente.

No infográfico, apresentamos resumidamente os principais conceitos relacionados à estrutura de pilha, bem como suas operações: empilhar e desempilhar elementos, além dos procedimentos de segurança que testam quando a pilha está cheia ou vazia.

Convidamos você a refletir sobre os conceitos apresentados neste infográfico!

ALGORITMOS E ESTRUTURA DE DADOS



ESTRUTURA DE PILHA

CONCEITOS

EMPILHAR (PUSH)

Adicionar um elemento no topo da pilha.

DESEMPILHAR (POP)

Remover o elemento no topo da pilha.

TOPO

O elemento no topo da pilha.

VAZIA

Uma pilha vazia não contém nenhum elemento.

TAMANHO

O número de elementos na pilha.

CAPACIDADE

O número máximo de elementos que a pilha pode conter.



Uma pilha é uma estrutura de dados linear que segue o princípio "último a entrar, primeiro a sair" (LIFO - Last In, First Out).

CORMEN, T. **Algoritmos** – Teoria e Prática. 3^a ed. Rio de Janeiro: LTC, 2022.
CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

ALGORITMOS E ESTRUTURA DE DADOS

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos.** 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C.** São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Unidade 3

Definição E Usos De Tabela De Espalhamento

Aula 1

Definição de Tabela de Espalhamento

Definição de tabela de espalhamento

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe um ótimo estudo nesta aula; cremos que os assuntos abordados serão de suma importância para o seu aprendizado em algoritmos e estrutura de dados.

ALGORITMOS E ESTRUTURA DE DADOS

Vamos apresentar a você outra estrutura de dado que está presente em nossa disciplina. Assim como as filas e pilhas, as tabelas de espalhamento, também conhecidas como tabelas *hash*, trazem grandes benefícios para construção de algoritmos.

Nesta aula, você vai conhecer os conceitos teóricos e as características das tabelas de espalhamento; e aprenderá a evitar alguns problemas que acontecem em algoritmos com elas. Essa estrutura de dado pode solucionar vários problemas de inserção e pesquisa de elementos em diversos tipos de conjunto de dados.

Vamos a um exemplo prático? Imagine que você foi contratado para desenvolver um aplicativo de organização de filmes. Os usuários desejam armazenar informações detalhadas sobre os filmes, como título, diretor, gênero e ano de lançamento; também desejam pesquisar e acessar esses dados de forma eficiente. A solução envolve o uso de tabelas de espalhamento para criar uma estrutura de organização que permite recuperação rápida de informações sobre os filmes, simplificando a experiência dos usuários na gestão de sua coleção cinematográfica.

Após a abordagem dos conceitos dessa estrutura de dado e a apresentação de exemplos desses algoritmos, você estará apto a implementar soluções que possam resolver problemas da realidade profissional com o algoritmo de tabelas de espalhamento.

Convidamos você a conhecer essa estrutura de dado e a entender como ela pode auxiliar sua atividade como um desenvolvedor de sistemas.

Vamos Começar!

Nesta aula, você vai estudar as tabelas de espalhamento, usadas para facilitar a busca de elementos em um determinado repositório ou conjunto de dados. Assim, vamos iniciar com algumas definições.

De acordo com Cormen (2022, p. 184), “embora a busca por um elemento em uma tabela de espalhamento possa demorar tanto quanto procurar um elemento em uma lista ligada, na prática o *hashing* funciona extremamente bem”. Quando o autor faz a comparação dessa estrutura com a estrutura de lista ligada, podemos ter em mente que o *hashing* é uma técnica de trabalho para organização de elementos dentro de um sistema computacional.

Definição de tabela de espalhamento

As tabelas de espalhamento recebem vários nomes, como tabelas de dispersão, tabelas de indexação, tabelas de escrutínio, método de cálculo de endereço, *hash table* ou tabelas *hash*. Essas tabelas utilizam a técnica de endereçamento para tornar mais eficiente o processo de busca por informações de forma mais ágil, pois não requerem que os dados estejam organizados em uma ordem específica para a consulta.

ALGORITMOS E ESTRUTURA DE DADOS

Essa técnica de organização e pesquisa das informações em conjunto de dados está dividida em duas partes:

- **A tabela de espalhamento:** responsável por armazenar, de forma distribuída, os elementos de um conjunto de dados, porém seguindo uma função lógica para esta distribuição, e não simplesmente de forma sequencial, a exemplo de uma lista linear, como as estruturas de pilhas e filas.
- **Função de espalhamento:** função que mapeia uma entrada (geralmente uma chave) para um valor em um espaço de índice em uma tabela de espalhamento. Sua principal finalidade é criar uma distribuição uniforme dos dados na tabela, permitindo o acesso eficiente aos elementos.

A tabela de espalhamento tem a tarefa de guardar diferentes subconjuntos de dados, e cada subconjunto possui uma identificação especial chamada de índice. Esses índices são criados através de uma função de espalhamento, que determina a qual subconjunto um elemento específico pertence. Para isso, a função avalia as características distintivas do próprio elemento. Além disso, a função de espalhamento desempenha um papel crucial na recuperação rápida do subconjunto de dados a que pertencem os elementos, conforme ilustrado na Figura 1.

ALGORITMOS E ESTRUTURA DE DADOS

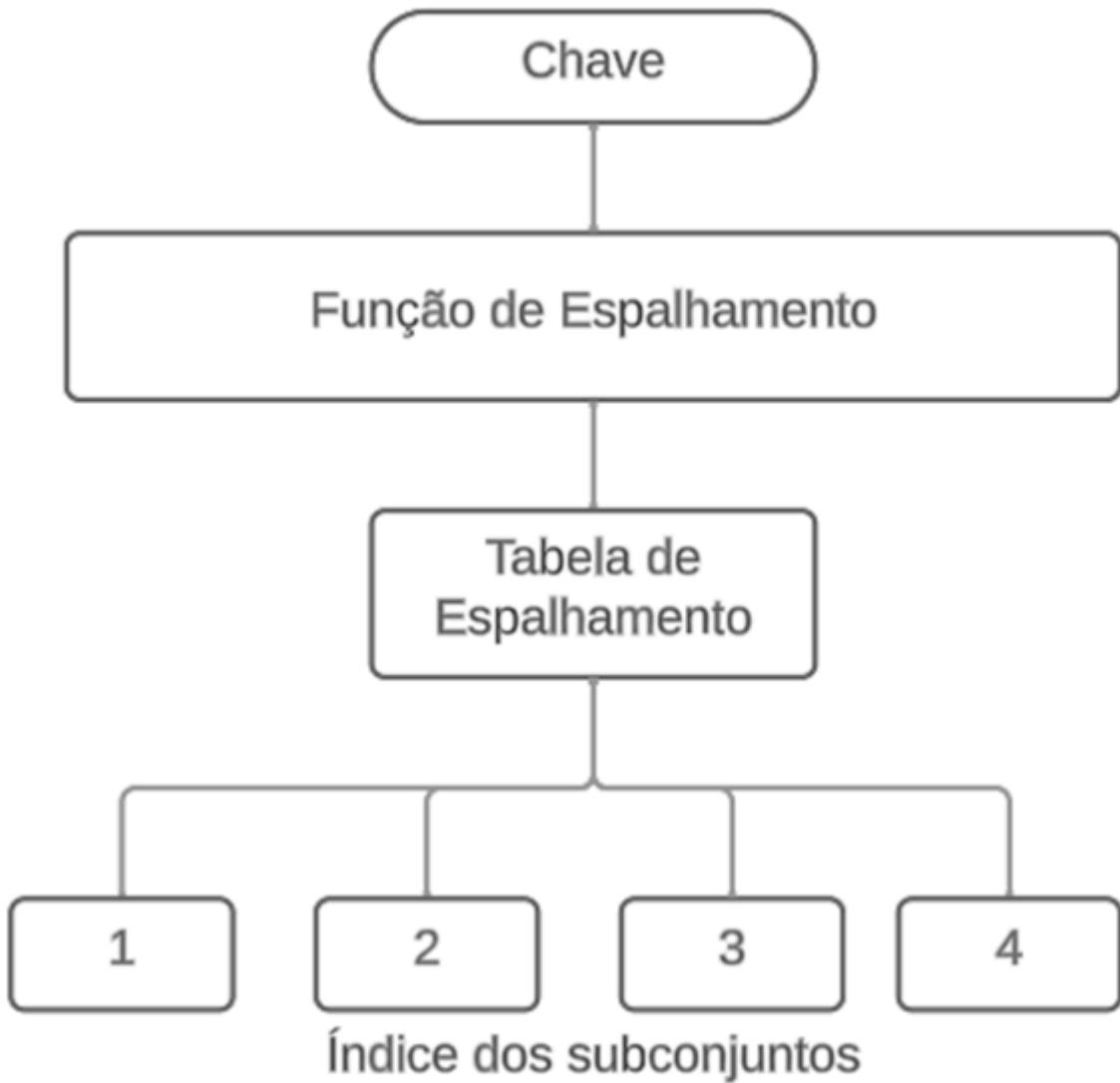


Figura 1 | Esquema da tabela de espalhamento.

Para exemplificação, podemos considerar nosso aplicativo de filmes. A função de espalhamento deve verificar qual é o gênero do filme e apresentar o número da categoria referente ao gênero pesquisado.

Dessa maneira, a chave seria o gênero do filme, que será utilizado pela função de espalhamento para responder em qual índice da tabela de espalhamento o filme será definitivamente armazenado.

Assim, considerando a perspectiva da ciência da computação, uma tabela *hash* é uma estrutura de dados que estabelece uma relação entre chaves e os valores associados a seus elementos. Essa estrutura de dados é projetada com o propósito principal de permitir a recuperação eficiente

ALGORITMOS E ESTRUTURA DE DADOS

de valores a partir de uma chave específica. A chave, nesse contexto, funciona como um tipo de identificador que direciona a busca pelos valores (Cormen, 2022).

Além disso, a lógica subjacente a essa técnica também encontra aplicação na indexação de grandes volumes de informações, por exemplo, em bancos de dados extensos (Cormen, 2012).

Diante do contexto, a função de espalhamento tem como finalidade principal converter o valor da chave de um elemento de dados em uma posição dentro de um subconjunto criado na estrutura. Portanto, a tabela de espalhamento é concebida para dividir os elementos em diferentes subconjuntos com base em suas características distintivas. Isso implica que a busca por um elemento específico se baseia na identificação do subconjunto ao qual o elemento pertence, permitindo eliminar a necessidade de considerar outros subconjuntos. Isso funciona de maneira semelhante a um filtro de pesquisa. Um exemplo ilustrativo pode ser observado na Figura 2.

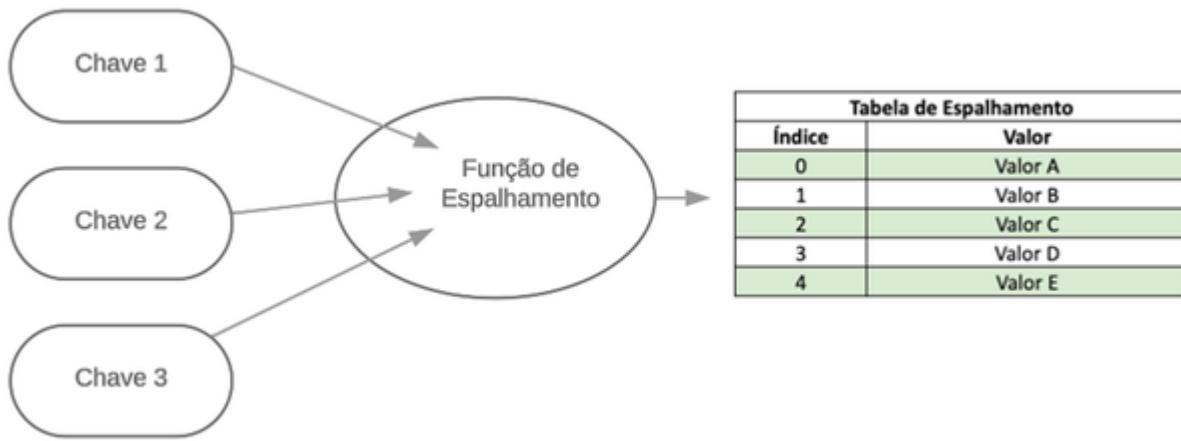


Figura 2 | Esquema da função e tabela de espalhamento.

Aplicações de tabelas de espalhamento

Dentro de vários cenários organizacionais, podem-se aplicar os algoritmos ou técnicas da tabela de espalhamento. Vamos exemplificar um caso de emergência hospitalar apresentado no Quadro 1, que exibe a classificação de risco pelo 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.

ALGORITMOS E ESTRUTURA DE DADOS

URGÊNCIA	Urgente: Caso de gravidade moderada, necessidade de atendimento médico, sem risco imediato.
POUCO URGENTE	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.

Quadro 1 | Protocolo de Manchester. Fonte: adaptado de Hospital Sta. Cruz.

A classificação de prioridade no atendimento é determinada com base nas categorias representadas no Quadro 1. No entanto, é essencial organizar o atendimento de acordo com a especialidade necessária, pois os pacientes que buscam atendimento de emergência no pronto-socorro têm diversas necessidades.

Nesse cenário, os algoritmos de espalhamento com tabelas *hash* desempenham um papel eficaz. Eles auxiliam a distribuição dos pacientes, tratados como dados na tabela de espalhamento, com base em suas características específicas. Quando um paciente chega ao hospital, é realizado o processo de triagem para garantir uma organização eficaz. Sem essa organização, o processo se tornaria lento e ineficiente, resultando em insatisfação e desconforto para os pacientes.

Para melhorar a organização dos atendimentos, consideramos que eles são conduzidos por enfermeiros ou médicos plantonistas. Em nosso exemplo, vamos supor vários pacientes, cada um com necessidades médicas distintas, representadas como índices.

Com base nesse contexto, exploraremos algumas especialidades médicas apresentadas no Quadro 2.

ALGORITMOS E ESTRUTURA DE DADOS

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

Quadro 2 | Divisão de pacientes por especialidade médica.

Conforme observado, os pacientes são identificados por códigos como P1, P2, representando Paciente 1, Paciente 2, etc. Além disso, o Quadro 2 ilustra a organização dos pacientes em filas separadas com base em suas necessidades médicas específicas.

A fim de evitar que os médicos precisem procurar individualmente por cada paciente, eles são organizados de acordo com índices, utilizando tabelas *hash*. Essa abordagem agiliza significativamente o processo de atendimento médico.

Após a triagem, os pacientes são direcionados para especialistas conforme a natureza de seus problemas de saúde, podendo ser separados em salas diferentes dentro do hospital, caso haja essa possibilidade. Nesse contexto, as tabelas *hash* desempenham um papel fundamental ao trazer maior organização e eficiência para o ambiente hospitalar, aprimorando o atendimento de forma notável.

ALGORITMOS E ESTRUTURA DE DADOS

Siga em Frente...

Evitando problemas no uso de tabelas de espalhamento

Como em qualquer aspecto da programação e da computação, o uso inadequado de tabelas de espalhamento pode levar a problemas de desempenho e comportamento inesperado. Agora, vamos explorar como evitar problemas no uso de tabelas de espalhamento por meio de dicas e melhores práticas. A seguir são citados alguns problemas comuns que podem acontecer nesse processo. Observe a seguir:

- **Colisões:** colisões ocorrem quando duas chaves diferentes são mapeadas para o mesmo índice na tabela de espalhamento. Isso pode resultar na sobreposição de valores e na perda de informações.
- **Desempenho inadequado:** se a função de espalhamento não for eficiente, a tabela pode se tornar ineficaz, com pesquisas lentas e alta carga de trabalho no sistema.
- **Alocação de memória insuficiente:** se a tabela não for dimensionada adequadamente, pode ocorrer falta de espaço para armazenar novos dados.
- **Conflitos de chave:** se duas chaves diferentes têm a mesma representação *hash* (colisão), pode ser difícil gerenciar conflitos e recuperar informações corretas.
- **Incompatibilidade de chaves:** a escolha inadequada de chaves ou funções de espalhamento pode levar a incompatibilidades, tornando as tabelas de espalhamento inúteis.

Agora que compreendemos os problemas comuns, vejamos como evitá-los e garantir o uso eficaz de tabelas de espalhamento. Acompanhe algumas sugestões a seguir:

- **Escolha de uma função de espalhamento eficiente:** a função de espalhamento desempenha um papel crucial. Certifique-se de escolher uma função eficiente, que distribua as chaves uniformemente pela tabela de espalhamento. Evite funções de espalhamento que geram muitas colisões.
- **Gerenciamento de colisões:** implemente estratégias para lidar com colisões, como encadeamento separado ou sondagem linear. Isso garantirá que as colisões sejam tratadas de maneira adequada e não causem perda de dados.
- **Dimensionamento adequado:** monitore a carga da tabela de espalhamento e dimensione-a conforme necessário. Um fator de carga (*load factor*) muito alto pode prejudicar o desempenho. Um fator de carga ideal é geralmente próximo a 1.
- **Escolha de chaves únicas:** use chaves que sejam exclusivas e representativas dos valores que você deseja armazenar. Isso reduzirá a probabilidade de colisões.
- **Teste e avaliação:** realize testes extensivos em sua implementação de tabela de espalhamento para garantir que ela funcione conforme o esperado. Avalie o desempenho e faça ajustes conforme necessário.

ALGORITMOS E ESTRUTURA DE DADOS

Além das dicas específicas, existem algumas melhores práticas gerais que podem ajudar a evitar problemas no uso de tabelas de espalhamento:

- **Documentação adequada:** mantenha documentação detalhada de como sua tabela de espalhamento é implementada, incluindo a função de espalhamento escolhida e a estratégia de tratamento de colisões. Isso facilitará a manutenção e o entendimento futuro do código.
- **Compreensão do domínio:** entenda completamente o domínio do problema que sua tabela de espalhamento deve resolver. Isso ajudará a escolha de chaves apropriadas e a definição de uma função de espalhamento eficaz.
- **Monitoramento de desempenho:** esteja atento ao desempenho da sua tabela de espalhamento, especialmente em situações de uso real. Isso permitirá identificar gargalos de desempenho e tomar medidas corretivas.
- **Considere estruturas de dados alternativas:** em alguns casos, outras estruturas de dados, como árvores, listas ou conjuntos, podem ser mais adequadas do que tabelas de espalhamento. Avalie cuidadosamente o cenário antes de escolher uma estrutura.

Vamos explorar algumas situações em que tabelas de espalhamento são comumente aplicadas na área técnica de desenvolvimento de sistemas e tecnologia:

- **Dicionários e tradução:** tabelas de espalhamento são ideais para implementar dicionários, nos quais palavras ou termos são associados a suas definições ou traduções. A chave pode ser a palavra e o valor associado, a definição ou a tradução.
- **Cache de dados:** em sistemas que precisam armazenar temporariamente dados frequentemente acessados, tabelas de espalhamento podem ser usadas para criar um mecanismo de cache eficiente.
- **Sistemas de banco de dados:** bancos de dados relacionais usam tabelas de espalhamento internamente para indexar registros, permitindo acesso rápido aos dados.
- **Implementação de conjuntos ou listas:** tabelas de espalhamento podem ser usadas para implementar estruturas de conjuntos ou listas, permitindo a verificação eficiente de pertencimento de elementos.

Tabelas de espalhamento são ferramentas poderosas na ciência da computação quando usadas corretamente. Evitar problemas no uso dessas estruturas requer uma compreensão sólida de suas características e a aplicação de dicas e melhores práticas. Ao escolher funções de espalhamento eficientes, gerenciar colisões adequadamente e dimensionar tabelas de forma apropriada, é possível obter desempenho excepcional e evitar problemas indesejados. Além disso, manter uma documentação clara e seguir as melhores práticas gerais contribuirá para o sucesso na implementação de tabelas de espalhamento em suas aplicações e sistemas.

Esperamos que tenham ficado claros para você, neste primeiro momento, os conceitos iniciais da estrutura de dados de tabela de espalhamento.

ALGORITMOS E ESTRUTURA DE DADOS

Vamos Exercitar?

Vamos retomar o exemplo prático apresentado no início da aula! Imagine que você foi contratado para desenvolver um aplicativo de organização de filmes. O aplicativo deve permitir que os usuários adicionem informações sobre filmes, como título, diretor, gênero e ano de lançamento. Para otimizar a busca e recuperação de filmes, você decide implementar tabelas de espalhamento (*hash tables*). O desafio é criar uma estrutura de dados eficaz que associe cada filme a uma chave única, garantindo que os usuários possam encontrar rapidamente informações sobre os filmes que desejam assistir.

Descreva como você aplicaria os algoritmos de tabela de espalhamento para o desenvolvimento desta aplicação.

Esta resolução trata apenas de forma descritiva os passos para solução da situação-problema, não levando em conta que o estudante, neste momento, deva desenvolver os algoritmos. Diante disso, vamos à resolução.

O aplicativo deve usar uma tabela de espalhamento; a chave única pode ser gerada a partir de uma combinação de informações do filme, como o título, diretor e ano de lançamento. Ao adicionar um filme, calcule a chave apropriada e insira o filme na tabela de espalhamento. Isso permitirá que os usuários pesquisem filmes de forma rápida e eficiente com base em diferentes critérios, garantindo uma organização eficaz dos filmes no aplicativo.

Na sequência, estão descritos os passos para implementar a solução da situação-problema:

- 1. Definir a estrutura de dados:** crie uma estrutura de dados para representar as informações de um filme, como título, diretor, gênero e ano de lançamento. Por exemplo, você pode criar uma classe "Movie" com esses atributos.
- 2. Criar a tabela de espalhamento:** implemente uma tabela de espalhamento (*hash table*) que será responsável por associar cada filme a uma chave única. A chave pode ser gerada a partir de informações do filme, como título, diretor e ano de lançamento.
- 3. Adicionar filmes à tabela de espalhamento:** ao adicionar um filme ao aplicativo, calcule a chave única para esse filme com base em suas informações. Use a função de espalhamento para determinar a posição em que o filme será armazenado na tabela.
- 4. Pesquisar filmes com eficiência:** implemente um mecanismo de pesquisa que permita aos usuários encontrar filmes de forma rápida e eficiente. Isso pode ser feito usando a chave gerada a partir das informações do filme.
- 5. Atualização e remoção de filmes:** ofereça funcionalidades para atualização e remoção de filmes. Ao atualizar informações de um filme, recalcule a chave e faça a atualização na tabela. Na remoção, localize o filme com base em sua chave e remova-o da tabela.
- 6. Garantir mínimos conflitos:** implemente técnicas para lidar com colisões, que ocorrem quando dois filmes geram a mesma chave. Pode-se usar encadeamento separado ou sondagem linear para resolver colisões.

ALGORITMOS E ESTRUTURA DE DADOS

Seguindo esses passos, você criará um aplicativo de organização de filmes que usa tabelas de espalhamento para facilitar a busca e a recuperação eficiente de informações sobre os filmes. Isso proporcionará aos usuários uma maneira conveniente de explorar e gerenciar sua coleção de filmes.

Saiba mais

Para saber mais sobre pesquisas recentes de algoritmos baseados em tabela de espalhamento e conhecer um pouco mais sobre esta técnica, acesse a pesquisa: "*Identificação de faces baseada em tabelas de espalhamento associadas com partial least squares*", de Gérson de Paulo Carlos, tendo como orientadores Helio Pedrini e Willian Robson Schwartz. Encontra-se no repositório da Universidade Federal de Minas Gerais (UFMG). O artigo propõe um método para identificação de faces que utiliza a tabela de espalhamento para obtenção do conjunto de dados do experimento. Não deixe de conferir para compreender um pouco mais sobre este assunto.

CARLOS, G. P. [Identificação de Faces baseada em Tabelas de Espalhamento Associadas com Partial Least Squares](#). Departamento de Ciência da Computação – UFMG: Minas Gerais, 2011.

Bons estudos!

Referências

CARLOS, G. P. **Identificação de Faces baseada em Tabelas de Espalhamento Associadas com Partial Least Squares**. Departamento de Ciência da Computação – UFMG: Minas Gerais, 2011. Disponível em: https://homepages.dcc.ufmg.br/~william/papers/paper_2011_SPSb.pdf. Acesso em: 23 nov. 2023.

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos: com implementações em Pascal e C**. 3^a ed. São Paulo: Cengage Learning, 2011.

ALGORITMOS E ESTRUTURA DE DADOS

Aula 2

Operações em Tabelas de Espalhamento

Operações em tabelas de espalhamento



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe um ótimo estudo nesta aula. Aproveite os conteúdos para entender um pouco mais sobre a tabela de espalhamento e as técnicas que essa estrutura de dados disponibiliza aos profissionais da área de desenvolvimento de sistemas.

Nesta aula, você vai conhecer um pouco das operações em tabelas de espalhamento. Algumas delas são: inserção, recuperação, atualização e exclusão de dados. Todas essas operações dentro das tabelas de espalhamento desempenham um papel central na eficiência e na performance dos sistemas de armazenamento e recuperação de dados.

Vamos a um exemplo prático? De início, suponha que você trabalhe como desenvolvedor de software em uma empresa que cria sistemas de gerenciamento de reservas de hotel. O sistema utiliza uma tabela de espalhamento para armazenar as informações das reservas de clientes. Cada reserva é identificada por um número de confirmação exclusivo, que é usado como chave na tabela de espalhamento. Recentemente, você foi designado para desenvolver a funcionalidade de exclusão de reservas do sistema. No entanto, você enfrenta um problema específico relacionado à exclusão de elementos na tabela de espalhamento.

ALGORITMOS E ESTRUTURA DE DADOS

Após o estudo dos conteúdos da aula e a apresentação de exemplos desses algoritmos, você estará apto a implementar a solução de exclusão que possa resolver problemas da realidade profissional com as tabelas de espalhamento.

Compreender como inserir, recuperar, atualizar, excluir dados, e como lidar com colisões, é fundamental para aproveitar ao máximo essa estrutura de dados. Além disso, a capacidade de dimensionar a tabela adequadamente é essencial para garantir que as operações continuem rápidas e eficazes à medida que mais dados são inseridos.

Convidamos você a conhecer essas operações da tabela de espalhamento e a estudar um pouco mais a esse respeito.

Vamos Começar!

Nesta aula, você vai estudar as operações básicas em tabelas de espalhamento. Assim como estudamos as ações de inserir, excluir e pesquisar elementos dentro de um vetor ou de uma lista, agora vamos abordar as tabelas de espalhamento para aplicar essas operações. Iniciemos o conteúdo desta aula.

Operações em tabelas de espalhamento

O método de “transformação de chave”, como alguns autores nomeiam as funções de *hashing*, é completamente diferente, pois os registros armazenados em uma tabela são diretamente endereçados a partir de uma transformação aritmética sobre a chave de pesquisa. De acordo com a tradução, a palavra *hash* em português quer dizer “fazer picadinho de carne e vegetais para cozinhar”. Segundo Ziviani (2011), o significado do termo realmente é apropriado para o método.

As duas etapas principais de um método de pesquisa com o uso de transformação de chave são:

- **Cálculo da função de transformação (função *hashing*):** a primeira etapa envolve o cálculo do valor da função de transformação, também conhecida como função de espalhamento (*hash function*). Essa função é responsável por converter a chave original, que pode ser qualquer dado, em um valor numérico chamado de “*hash*” ou “código *hash*”. A função de transformação é projetada de maneira que, idealmente, cada chave seja mapeada para um valor de *hash* exclusivo na tabela de espalhamento. Isso é fundamental para distribuir as chaves de forma uniforme na tabela, garantindo que a recuperação de dados seja eficiente.
- **Resolução de colisões:** a segunda etapa lida com o problema das colisões, que ocorre quando duas ou mais chaves são transformadas em um mesmo endereço da tabela; ou seja, quando dois ou mais valores de *hash* são idênticos. Para resolver colisões, é necessário existir um método eficaz que permita armazenar e recuperar múltiplas chaves associadas ao mesmo endereço.

ALGORITMOS E ESTRUTURA DE DADOS

Considerando chaves de valores inteiros de 1 a n, pode-se armazenar o registro com chave de i na posição i da tabela. Qualquer registro poderia ser acessado a partir do valor da chave. Porém, suponha que uma tabela fosse capaz de armazenar M = 97 chaves, considerando ainda que cada chave pode ser um número decimal de quatro dígitos. Então, agora existem N = 10.000 chaves possíveis e a função de espalhamento não pode ser um para um, mesmo se os elementos que serão armazenados forem bem menores que o total, ou seja, 97. As colisões vão acontecer e elas tem que ser resolvidas de alguma maneira (Ziviani, 2011).

O “paradoxo do aniversário” nos diz que, em um grupo de 23 pessoas, existe a probabilidade de mais que 50% de duas ou mais pessoas fazerem aniversário no mesmo dia. portanto se for utilizada uma função de espalhamento que enderece 23 chaves aleatórias em uma tabela de tamanho 365, a probabilidade de que haja colisões é maior que 50%. A probabilidade p de se inserirem N itens consecutivos sem colisão em uma tabela de tamanho M seria a fórmula como pode ser observada a seguir:

$$p = \frac{M-1}{M} \times \frac{M-2}{M} \times \dots \times \frac{M-N+1}{M} = \prod_{i=1}^N \frac{M-i+1}{M} = \frac{M!}{(M-N)!M^N}$$

Equação 1 | Fórmula de probabilidades sem colisão. Fonte: adaptada de Ziviani (2011).

O Quadro 1 a seguir, apresenta alguns valores de p para diferentes valores de N, em que M = 365. Observe.

N	p
10	0,883
22	0,524
23	0,493
30	0,303

Quadro 1 | Fórmula de probabilidades sem colisão. Fonte: adaptado de Ziviani (2011).

Para N pequeno, a probabilidade de p pode ser aproximada em N = 10; então: p 87,7%.

Função de espalhamento

Como já explicado, a função de espalhamento, também conhecida como função *hash*, é um componente crítico no uso de tabelas de espalhamento. Ela desempenha um papel fundamental na distribuição eficiente de dados ao mapear chaves (ou dados) para posições específicas na tabela. Vamos explorar essa função em detalhes com exemplos.

ALGORITMOS E ESTRUTURA DE DADOS

Uma função de espalhamento é um algoritmo que aceita uma chave como entrada e produz um valor numérico chamado de “código *hash*”. O objetivo principal dessa função é distribuir as chaves de forma uniforme na tabela de espalhamento. Isso é importante para garantir que as operações de busca, inserção e exclusão sejam eficientes, uma vez que as chaves estejam espalhadas por toda a tabela, em vez de ficarem agrupadas em um local específico.

Exemplo: Vamos considerar um exemplo de função de espalhamento. Suponha que temos uma tabela de espalhamento com 10 posições e desejamos armazenar palavras. A função de espalhamento poderia ser a seguinte:

1. Converta as letras da palavra em valores numéricos (por exemplo, “a” seria 1, “b” seria 2, “c” seria 3 e assim por diante).
2. Some os valores numéricos das letras da palavra.

Agora, vamos aplicar essa função a algumas palavras:

- “apple” => $a(1) + p(16) + p(16) + l(12) + e(5) = 50$
- “banana” => $b(2) + a(1) + n(14) + a(1) + n(14) + a(1) = 33$
- “cherry” => $c(3) + h(8) + e(5) + r(18) + r(18) + y(25) = 77$

Esses valores (50, 33 e 77) são os códigos *hash* resultantes da aplicação da função de espalhamento às palavras. Agora, podemos usar esses códigos *hash* para mapear as palavras nas posições da tabela de espalhamento.

Suponha que queremos armazenar as palavras em nossa tabela de espalhamento usando os códigos *hash* como índices. O processo de armazenamento seria o seguinte:

- “apple” (código *hash* 50) é armazenada na posição 50 da tabela.
- “banana” (código *hash* 33) é armazenado na posição 33 da tabela.
- “cherry” (código *hash* 77) é armazenado na posição 77 da tabela.

Quando desejarmos recuperar uma palavra da tabela, aplicamos a mesma função de espalhamento à chave (a palavra que estamos procurando) para calcular o código *hash*. Em seguida, usamos esse código *hash* para encontrar a posição correspondente na tabela e recuperar a palavra.

Importância da função de espalhamento

Uma função de espalhamento eficaz é crucial para evitar colisões, que ocorrem quando duas chaves produzem o mesmo código *hash*. Colisões podem prejudicar o desempenho das tabelas de espalhamento. Portanto, a escolha de uma função de espalhamento adequada é fundamental para garantir que os dados sejam distribuídos de maneira uniforme na tabela.

ALGORITMOS E ESTRUTURA DE DADOS

Em resumo, a função de espalhamento desempenha um papel vital no funcionamento eficiente das tabelas de espalhamento, permitindo o mapeamento adequado das chaves para as posições na tabela. Ela é projetada de maneira a minimizar colisões e distribuir as chaves de maneira uniforme, facilitando as operações de busca e recuperação de dados.

Siga em Frente...

Redimensionamento dinâmico

O redimensionamento dinâmico, também conhecido como *rehashing*, é um conceito essencial na gestão de tabelas de espalhamento. Ele se refere à capacidade de ajustar o tamanho da tabela de espalhamento à medida que mais elementos são inseridos. O objetivo principal do redimensionamento dinâmico é manter um fator de carga aceitável, garantir um desempenho eficiente e evitar colisões excessivas. Vamos explorar esse conceito em detalhes.

À medida que os elementos são inseridos em uma tabela de espalhamento, o número de colisões pode aumentar, tornando a busca e a recuperação de dados menos eficientes. Para evitar isso, é necessário ajustar o tamanho da tabela para acomodar o crescimento dos dados. O redimensionamento dinâmico permite que a tabela aumente de tamanho à medida que mais elementos são adicionados, mantendo um fator de carga (relação entre o número de elementos e o tamanho da tabela) razoável.

O processo de redimensionamento dinâmico envolve as seguintes etapas:

- Monitoramento do fator de carga:** a tabela de espalhamento mantém o controle do fator de carga. Este é calculado como o número de elementos na tabela dividido pelo tamanho da tabela. Se o fator de carga exceder um limite predeterminado (por exemplo, 0,7), é hora de redimensionar a tabela.
- Criação de uma nova tabela:** quando é detectado que o fator de carga está alto, uma nova tabela de espalhamento é criada com um tamanho maior. O tamanho da nova tabela é geralmente escolhido como um número primo, para evitar colisões mais frequentes.
- Rehashing dos elementos:** todos os elementos da tabela original são reorganizados na nova tabela. Isso envolve a aplicação da função de espalhamento atual a cada elemento para calcular seu novo código *hash* na tabela maior. Em seguida, os elementos são inseridos nas novas posições com base em seus novos códigos *hash*.
- Descarte da tabela antiga:** após a conclusão do *rehashing*, a tabela de espalhamento original é descartada, uma vez que todos os elementos foram transferidos para a nova tabela.

Agora vamos a um exemplo para explicar um pouco melhor na prática este conceito. Suponha que você tenha uma tabela de espalhamento inicial com 10 posições e insira 7 elementos. O

ALGORITMOS E ESTRUTURA DE DADOS

fator de carga atual é $7/10 = 0,7$, o que excede o limite predeterminado de 0,7. É hora de redimensionar a tabela.

Você decide criar uma nova tabela com 20 posições (um tamanho maior) e reorganiza os elementos. O redimensionamento envolve a aplicação da função de espalhamento atual a cada elemento para calcular seu novo código *hash* na tabela de 20 posições. Os elementos são então distribuídos nas novas posições.

Após o redimensionamento, a nova tabela tem um fator de carga de $7/20 = 0,35$, o que está dentro do limite aceitável. A tabela redimensionada é apresentada no Quadro 2, a seguir:

1	
2	
3	45
4	
5	32
6	
7	89
8	
9	
10	
11	
12	
13	21
14	65
15	
16	
17	67
18	

ALGORITMOS E ESTRUTURA DE DADOS

19	99
20	

Quadro 2 | Tabela redimensionada.

Vantagens do redimensionamento dinâmico

Algumas vantagens da utilização do redimensionamento dinâmico em tabelas de espalhamento são apresentadas a seguir:

- **Melhor desempenho:** mantém um fator de carga baixo, o que leva a pesquisas mais eficientes.
- **Evita colisões excessivas:** o aumento do tamanho da tabela reduz as colisões de chaves, melhorando a distribuição dos elementos.
- **Adapta-se ao crescimento:** à medida que mais elementos são inseridos, a tabela pode se ajustar automaticamente.

Em resumo, o redimensionamento dinâmico é uma estratégia fundamental na gestão de tabelas de espalhamento, permitindo que elas se adaptem ao crescimento de dados e evitem problemas de desempenho associados a colisões e fatores de carga elevados.

Esperamos que você tenha assimilado os conteúdos abordados nesta aula. Com certeza eles serão muito importantes para que você continue se aprofundando e utilizando as técnicas de operações em tabelas de espalhamento.

Vamos Exercitar?

Vamos retomar nosso exemplo prático apresentado no início da aula! Os clientes do hotel podem fazer reservas para várias datas e quartos, e cada reserva é registrada no sistema com um número de confirmação exclusivo. O sistema utiliza uma tabela de espalhamento para armazenar as informações das reservas, com os números de confirmação sendo as chaves. Agora, você precisa implementar a funcionalidade de exclusão de reservas. No entanto, ao tentar excluir uma reserva, você percebe que existem colisões de chaves devido a diferentes reservas compartilharem o mesmo número de confirmação.

Como você deve lidar com essa situação e implementar com sucesso a exclusão de reservas na tabela de espalhamento?

Lidar com colisões de chaves é uma parte crítica da operação de exclusão em tabelas de espalhamento. Aqui está um plano para resolver esse problema:

ALGORITMOS E ESTRUTURA DE DADOS

1. **Identificar a reserva a ser excluída:** antes de excluir uma reserva, é importante identificar qual reserva deve ser removida. O cliente fornecerá detalhes da reserva, como o nome do titular, as datas da reserva ou outras informações identificadoras.
2. **Pesquisar na tabela de espalhamento:** usando as informações fornecidas pelo cliente, pesquise na tabela de espalhamento para localizar a reserva desejada. Como pode haver colisões de chaves, você pode encontrar mais de uma reserva com o mesmo número de confirmação.
3. **Lidar com colisões:** se você encontrar múltiplas reservas com o mesmo número de confirmação, precisará considerar estratégias de resolução de conflitos. Uma opção é usar uma lista ligada (encadeamento separado) para manter várias entradas com a mesma chave.
4. **Exclusão da reserva:** agora que você identificou a reserva a ser excluída, remova a entrada correspondente ou, se não houver colisões, simplesmente exclua a entrada na tabela de espalhamento.

Lidar com a exclusão de elementos em tabelas de espalhamento pode ser desafiador, especialmente quando ocorrem colisões de chaves. No cenário profissional de gerenciamento de reservas de hotel, é essencial que o desenvolvedor seja capaz de identificar corretamente a reserva a ser excluída, pesquisar na tabela de espalhamento e aplicar estratégias apropriadas para resolver conflitos de colisões. O uso de listas ligadas para armazenar várias entradas com a mesma chave é uma técnica eficaz para lidar com essa situação. Garantir que a funcionalidade de exclusão seja empregada sem problemas é fundamental para manter a integridade e a confiabilidade do sistema de reservas de hotel.

Saiba mais

Para saber mais sobre as operações em tabelas de espalhamento, acesse o artigo: "*Hashing na solução de problemas atípicos*", dos autores Lucila Maria de Souza Bento, Vinícius Gusmão Pereira de Sá e Jayme Luiz Szwarcfiter, do Departamento de Ciência da Computação – Instituto de Matemática da Universidade Federal do Rio de Janeiro (UFRJ). A pesquisa ilustra o poder dessa técnica, também chamada de *hashing*, em situações que não lhe são comumente associadas. O artigo também explica em detalhes a técnica algorítmica utilizada no estudo. Não deixe de conferir na íntegra o artigo e enriquecer seu aprendizado com mais essa oportunidade de conhecimento.

BENTO, L. M. S.; SÁ, V.s G. P.; SZWARCFITER, Jayme L. [Hashing na Solução de Problemas Atípicos](#). Congresso Latino-Iberoamericano de Investigación Operativa, Simpósio Brasileiro de Pesquisa Operacional, Rio de Janeiro, 24-28 set. 2012.

Bons estudos!

ALGORITMOS E ESTRUTURA DE DADOS

Referências

BENTO, L. M. S.; SÁ, V.s G. P.; SZWARCFITER, Jayme L. Hashing na Solução de Problemas Atípicos. **Congresso Latino-Iberoamericano de Investigación Operativa, Simpósio Brasileiro de Pesquisa Operacional**, Rio de Janeiro, 24-28 set. 2012. Disponível em: <http://www.din.uem.br/sbpo/sbpo2012/pdf/arg0449.pdf>. Acesso em: 23 nov. 2023.

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 3

Otimização de Tabelas de Espalhamento

Otimização de tabelas de espalhamento

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

ALGORITMOS E ESTRUTURA DE DADOS

Ponto de Partida

Olá, estudante. Desejamos-lhe um ótimo estudo nesta aula. Aproveite os conteúdos que serão abordados sobre assuntos referentes a tabelas de espalhamento, tema do nosso estudo.

Nesta aula, você vai estudar e compreender o tratamento de colisões em tabelas de espalhamento, a complexidade de uso em tabelas de espalhamento e a atualização de valores e tratamento dos elementos.

Por hora, vamos abordar um cenário profissional no qual você possa aplicar as técnicas dos algoritmos em tabelas de espalhamento. Suponha o seguinte: em uma biblioteca universitária, a organização e o acesso eficiente aos livros são fundamentais para atender às necessidades dos alunos, professores e pesquisadores. Para gerenciar o acervo, o responsável pelo sistema de catalogação da biblioteca implementou uma tabela de espalhamento. Nesse sistema, o título dos livros é usado como chave para mapear cada livro para uma prateleira específica.

Essa abordagem inicialmente funcionou bem, mas recentemente surgiu um problema preocupante. Alguns títulos de livros, muito populares e frequentemente usados, estão causando colisões na tabela de espalhamento. Em outras palavras, vários livros têm o mesmo título, o que dificulta a recuperação de livros específicos. Imagine a frustração de um aluno que deseja encontrar um livro importante e descobre que há vários exemplares com o mesmo título, mas de autores diferentes.

Como o responsável pelo sistema de catalogação, você enfrenta o desafio de lidar com essas colisões e garantir que os livros sejam facilmente acessíveis. Sua tarefa é implementar uma estratégia eficaz para resolver esse problema, mantendo a eficiência do sistema e melhorando a experiência dos usuários da biblioteca.

Diante do cenário apresentado, compreender como lidar com as colisões em tabelas de espalhamento vai fazer com que você utilize as técnicas abordadas ao longo da aula e, assim, tenha a oportunidade de entender a prática, fixando melhor o conteúdo e dando significado ao seu aprendizado. O tratamento de colisões é essencial para a inserção de elementos em tabelas de espalhamento.

Convidamos você a estudar este conteúdo e a conhecer um pouco mais sobre a disciplina de Algoritmos e Estrutura de Dados por meio desta aula.

Vamos Começar!

Nesta aula, você vai continuar estudando as técnicas de tabelas de espalhamento, porém com um foco na otimização de alguns processos já estudados como: inserção, exclusão, etc. Iniciemos o conteúdo.

ALGORITMOS E ESTRUTURA DE DADOS

Otimização de tabelas de espalhamento

Até agora, você explorou os conceitos relacionados a tabelas de espalhamento e aprendeu como a função de espalhamento desempenha um papel crucial ao determinar:

1. O índice usado para recuperar informações ou elementos na estrutura.
2. O local onde um elemento ou informação é armazenado na tabela de espalhamento.

Sabe-se que uma função de espalhamento perfeita tem a capacidade de alocar todos os elementos de um conjunto de chaves em apenas uma entrada da tabela de espalhamento. Geralmente, atribuímos o termo “perfeita” a uma função de espalhamento quando ela é uma correspondência um a um, como ilustrado na Figura 1.

ALGORITMOS E ESTRUTURA DE DADOS

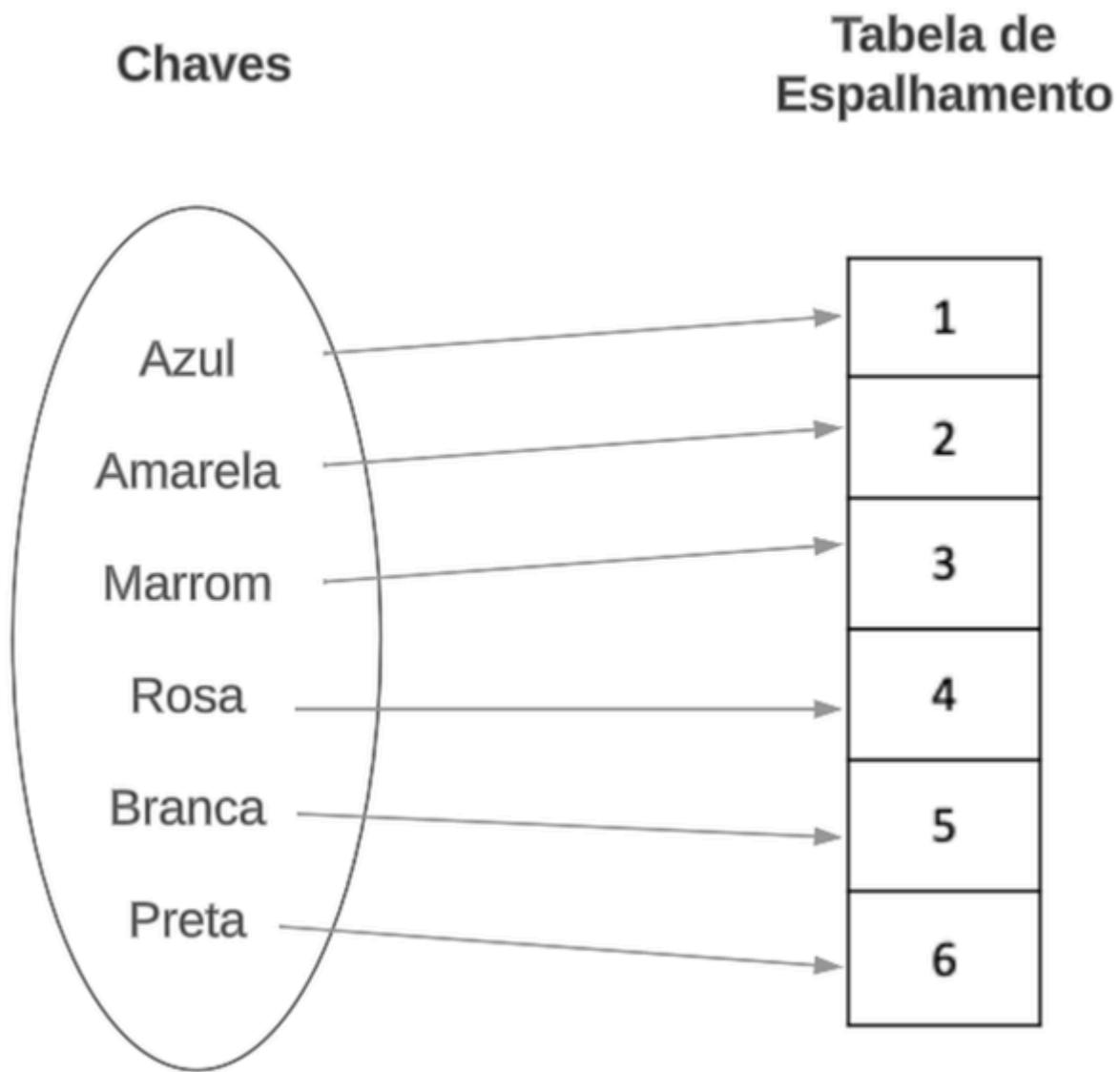


Figura 1 | Exemplo de função de espalhamento perfeita (encadeamento perfeito).

Siga em Frente...

Tratamento de colisões em tabelas de espalhamento

As tabelas de espalhamento podem experimentar uma redução de desempenho devido principalmente ao aumento gradual de elementos com a mesma chave calculada pela função de espalhamento, o que chamamos de colisão.

ALGORITMOS E ESTRUTURA DE DADOS

As colisões podem ser tratadas de várias formas; algumas das principais estratégias estão elencadas a seguir:

1. **Encadeamento separado:** cada entrada da tabela de espalhamento contém uma lista ligada de elementos com a mesma chave de espalhamento. As colisões são resolvidas adicionando elementos a essa lista.
2. **Endereçamento aberto:** nessa abordagem, quando ocorre uma colisão, você procura a próxima posição vazia na tabela para inserir o elemento. Isso pode ser feito de várias maneiras, como sondagem linear, sondagem quadrática ou sondagem dupla.
3. **Re-hash:** quando ocorre uma colisão, a função de espalhamento é recalculada com base na chave ou em uma fórmula modificada para encontrar uma posição alternativa na tabela.
4. **Espalhamento perfeito:** embora seja difícil de atingir, o espalhamento perfeito envolve o uso de uma função de espalhamento que garante que cada chave tenha uma posição única na tabela, eliminando colisões.

Vamos ver um pouco mais detalhadamente cada uma dessas estratégias para lidar com colisões em conjunto de dados dentro de tabelas de espalhamento.

Encadeamento separado

O encadeamento separado é uma técnica usada para lidar com colisões em tabelas de espalhamento. Quando duas ou mais chaves são mapeadas para o mesmo índice na tabela de espalhamento (ou seja, elas têm a mesma função de espalhamento), ocorre uma colisão. Em vez de substituir o valor existente, o encadeamento separado permite que várias chaves com a mesma função de espalhamento sejam armazenadas juntas em uma estrutura de dados, geralmente em uma lista ligada.

Cada entrada na tabela de espalhamento mantém uma estrutura de dados que pode conter múltiplos elementos com a mesma chave de espalhamento. Uma lista ligada é comumente usada para esse fim.

Quando um novo elemento é inserido na tabela de espalhamento, a função de espalhamento é usada para calcular o índice no qual o elemento deve ser colocado. Se a posição está vazia, o novo elemento é inserido diretamente. No entanto, se já houver elementos naquela posição (colisão), o novo elemento é adicionado à lista ligada existente nessa posição.

Para buscar um elemento na tabela de espalhamento, a função de espalhamento é usada novamente para calcular o índice. Em seguida, a lista ligada nessa posição é percorrida para encontrar o elemento desejado. Isso garante que todas as chaves com o mesmo índice sejam verificadas.

Para remover um elemento, a busca é realizada da mesma maneira, e quando o elemento é encontrado na lista ligada, ele é removido. Se a lista ligada ficar vazia após a remoção, a posição na tabela de espalhamento é definida como vazia.

ALGORITMOS E ESTRUTURA DE DADOS

Algumas vantagens da utilização do encadeamento separado incluem: flexibilidade, eficiência e simplicidade na implementação, tornando-o uma escolha popular para lidar com colisões.

Encadeamento aberto

Ao contrário do encadeamento separado, que usa uma estrutura de dados adicional (geralmente uma lista ligada) para armazenar elementos com a mesma chave de espalhamento, o encadeamento aberto lida com colisões diretamente na própria tabela de espalhamento. Para utilização do encadeamento aberto, alguns pontos são necessários:

- **Tabela de espalhamento:** a tabela de espalhamento é uma matriz na qual os elementos são armazenados. Cada posição na matriz pode conter um elemento ou estar vazia.
- **Função de espalhamento:** a função de espalhamento é usada para calcular a posição (ou índice) da tabela de espalhamento na qual um elemento deve ser inserido. Ela também é usada para buscar elementos.
- **Inserção:** quando um novo elemento deve ser inserido na tabela de espalhamento, a função de espalhamento é usada para calcular a posição. Se a posição estiver vazia, o elemento é inserido nessa posição. Se a posição estiver ocupada (colisão), o encadeamento aberto busca a próxima posição disponível na tabela (geralmente de forma sequencial) e insere o elemento lá.
- **Busca:** para buscar um elemento na tabela de espalhamento, a função de espalhamento é usada para calcular a posição. Se o elemento estiver na posição calculada, ele é encontrado. Se não estiver, a busca continua na próxima posição (se houver) até encontrar o elemento desejado ou determinar que ele não está na tabela.
- **Remoção:** a remoção de um elemento segue um processo semelhante à busca. A função de espalhamento é usada para calcular a posição, e a tabela de espalhamento é verificada nessa posição. Se o elemento estiver lá, ele é removido. Se não estiver, a busca continua nas posições subsequentes, se houver.

O encadeamento aberto oferece vantagens, como economia de espaço (não há estruturas de dados adicionais) e eficiência de memória. No entanto, ele pode ficar menos eficiente à medida que a tabela de espalhamento fica mais cheia, e o processo de busca pode se tornar mais demorado, já que é necessário verificar posições adicionais. Além disso, é importante ter uma estratégia sólida para lidar com colisões para evitar problemas de desempenho.

Em resumo, o encadeamento aberto é uma técnica de resolução de colisões que lida diretamente com colisões na própria tabela de espalhamento, armazenando elementos com a mesma chave na próxima posição disponível.

Re-hash

O encadeamento com *re-hash* é uma técnica de tratamento de colisões em tabelas de espalhamento que combina elementos do encadeamento aberto e do *re-hash* tradicional. Essa

ALGORITMOS E ESTRUTURA DE DADOS

abordagem é projetada para lidar com colisões, mantendo um controle mais eficiente do fator de carga da tabela e evitando o desperdício de espaço.

Este tipo de encadeamento funciona inicialmente pela tabela de espalhamento principal. Essa tabela é responsável por armazenar os elementos; é nela que a função de espalhamento é aplicada para calcular a posição de inserção.

A função de espalhamento principal é usada para calcular a posição na qual um elemento deve ser inserido na tabela de espalhamento principal. Se ocorrer uma colisão (ou seja, a posição estiver ocupada), o encadeamento com *re-hash* entra em ação.

Quando ocorre uma colisão na tabela de espalhamento principal, em vez de usar uma lista ligada ou sondagem linear para encontrar a próxima posição disponível, o encadeamento com *re-hash* cria uma segunda tabela de espalhamento, chamada tabela de espalhamento auxiliar. Essa tabela também tem uma função de espalhamento auxiliar que é usada para calcular a posição na qual o elemento colidido deve ser inserido na tabela de espalhamento auxiliar. Se, por um acaso, ocorrer uma nova colisão na tabela de espalhamento auxiliar, um novo *re-hash* pode ser executado para criar uma terceira tabela de espalhamento, e assim por diante, até que uma posição vazia seja encontrada.

A busca por um elemento na tabela de espalhamento segue um processo semelhante. A função de espalhamento principal é usada para calcular a posição na tabela de espalhamento principal, e se ocorrer uma colisão, a função de espalhamento auxiliar é usada na tabela de espalhamento auxiliar.

O encadeamento com *re-hash* ajuda a manter um fator de carga controlado na tabela de espalhamento principal, garantindo que ela não fique muito cheia. Quando o fator de carga excede um limite predefinido, um novo *re-hash* é acionado para criar uma nova tabela de espalhamento principal maior, redistribuindo os elementos de acordo com ele.

Essa técnica é eficaz para evitar colisões e controlar o fator de carga, mantendo o uso eficiente de espaço. No entanto, requer a implementação de lógica adicional para gerenciar as tabelas de espalhamento auxiliares e o processo de *re-hash*, tornando-a mais complexa do que algumas outras abordagens de tratamento de colisões.

Encadeamento perfeito

O encadeamento perfeito, ou “*perfect hashing*”, é uma técnica avançada de resolução de colisões em tabelas de espalhamento que busca eliminar completamente as colisões, garantindo que cada elemento tenha uma posição única na tabela. Essa técnica é chamada “perfeita” porque cria uma função de espalhamento tão eficiente que não permite colisões entre os elementos. Observe novamente a Figura 1 no início dessa aula para visualizar o esquema do encadeamento perfeito.

ALGORITMOS E ESTRUTURA DE DADOS

Na sequência, são apresentados os principais aspectos do encadeamento perfeito:

- **Função de espalhamento principal:** a chave para o encadeamento perfeito está na escolha de uma função de espalhamento principal altamente eficiente, que é capaz de mapear cada chave exclusivamente para uma posição na tabela de espalhamento. A função de espalhamento principal deve ser cuidadosamente projetada para evitar colisões.
- **Tabelas de espalhamento auxiliares:** para atingir a perfeição, o encadeamento perfeito usa duas ou mais tabelas de espalhamento auxiliares. Isso significa que, em vez de uma única tabela de espalhamento, você terá várias tabelas auxiliares, cada uma com uma função de espalhamento auxiliar.
- **Primeira função de espalhamento auxiliar:** é usada para distribuir os elementos nas tabelas auxiliares. Cada tabela auxiliar é responsável por um subconjunto dos elementos.
- **Segunda função de espalhamento auxiliar:** dentro de cada tabela auxiliar, uma segunda função de espalhamento auxiliar é usada para calcular a posição final de inserção para cada elemento. Essa função deve ser projetada para eliminar colisões dentro de cada tabela auxiliar, garantindo que cada elemento tenha uma posição única.
- **Busca e inserção:** quando você deseja buscar ou inserir um elemento, a primeira função de espalhamento auxiliar direciona você para a tabela auxiliar correta. Em seguida, a segunda função de espalhamento auxiliar determina a posição exata em que o elemento deve ser colocado ou encontrado.
- **Complexidade:** o encadeamento perfeito é uma técnica avançada e geralmente requer um pré-processamento intensivo para calcular as funções de espalhamento principais e auxiliares. No entanto, uma vez que as funções são determinadas, as operações de busca e inserção são extremamente eficientes, com tempo de execução $O(1)$ em média.

O encadeamento perfeito é ideal quando você precisa de garantias de que não haverá colisões na sua tabela de espalhamento, e está disposto a investir esforços na fase de pré-processamento para projetar as funções de espalhamento. Ele é especialmente útil em cenários nos quais o espaço é limitado e a eficiência é crucial, como sistemas embarcados e algoritmos de tempo real. No entanto, a complexidade de implementação pode torná-lo inviável em certas situações.

Em resumo, o redimensionamento dinâmico é uma estratégia fundamental na gestão de tabelas de espalhamento, permitindo que elas se adaptem ao crescimento de dados e evitem problemas de desempenho associados a colisões e fatores de carga elevados.

Esperamos que você tenha assimilado os conteúdos abordados nesta aula. Com certeza eles serão muito importantes para que você continue se aprofundando e utilizando as técnicas de operações em tabelas de espalhamento.

Vamos Exercitar?

ALGORITMOS E ESTRUTURA DE DADOS

Voltando ao cenário proposto inicialmente! Você é o responsável pelo sistema de catalogação de uma biblioteca universitária. Para organizar os livros, você implementou uma tabela de espalhamento que utiliza o título dos livros como chave para armazená-los nas prateleiras correspondentes. No entanto, recentemente, você notou um aumento nas colisões, pois dois ou mais livros têm o mesmo título. Isso está dificultando a recuperação dos dados. Agora é com você!

Como você pode lidar com as colisões de forma eficaz neste exemplo em tabela de espalhamento?

Para lidar com as colisões na tabela de espalhamento neste exemplo da biblioteca, você pode aplicar a estratégia de “encadeamento separado”. Aqui estão os passos:

1. **Crie uma lista ligada em cada posição da tabela:** em vez de armazenar um único livro em cada posição da tabela, crie uma lista ligada em cada posição. Isso permite que você armazene vários livros na mesma posição sem substituir os existentes.
2. **Inserção de livros:** quando você insere um livro na tabela, calcule o código *hash* com base em seu título. Em seguida, vá para a posição correspondente na tabela.
3. **Trate de colisões:** se houver mais de um livro na mesma posição devido a uma colisão, basta adicionar o novo livro à lista ligada. Dessa forma, você pode manter todos os livros com o mesmo título juntos.
4. **Recuperação de livros:** para recuperar um livro, calcule o código *hash* com base em seu título, vá para a posição correspondente na tabela e percorra a lista ligada para encontrar o livro desejado.

Lidando com colisões de maneira eficaz, você pode manter a integridade do sistema de catalogação da biblioteca e garantir que os livros com títulos idênticos sejam armazenados e recuperados corretamente. A estratégia de encadeamento separado é uma abordagem eficaz para gerenciar colisões, garantindo que todos os livros estejam acessíveis aos usuários da biblioteca.

Saiba mais

Para saber mais sobre tabelas de espalhamento, acesse a pesquisa: “*Tabelas hash distribuídas com consultas de um salto com baixa carga de manutenção*”, de Luiz Rodolpho Rocha Monnerat, apresentada no Programa de Pós-graduação em Engenharia de Sistemas da Universidade Federal do Rio de Janeiro (UFRJ), disponível em: A pesquisa apresenta uma inovadora tabela *hash* que é capaz de resolver consultas com um salto e tem baixas demandas de rede, mesmo em ambientes com dinâmicas representativas de aplicações distribuídas populares. O trabalho explica as técnicas e procedimentos utilizados no estudo. Não deixe de conferir na íntegra o texto e enriquecer seu aprendizado com mais essa oportunidade de conhecimento.

ALGORITMOS E ESTRUTURA DE DADOS

MONNERAT, L. R. R. [Tabelas Hash Distribuídas com Consultas de um Salto com Baixa Carga de Manutenção](#). Rio de Janeiro: UFRJ, 2010.

Bons estudos!

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

MONNERAT, L. R. R. **Tabelas Hash Distribuídas com Consultas de um Salto com Baixa Carga de Manutenção**. Rio de Janeiro: UFRJ, 2010. Disponível em:
http://objdig.ufrj.br/60/teses/coppe_d/LuizRodolphoRochaMonnerat.pdf. Acesso em: 27 nov. 2023.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 4

Uso Avançado e Aplicações

Uso avançado e aplicações

ALGORITMOS E ESTRUTURA DE DADOS



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante. Desejamos-lhe um ótimo estudo nesta aula. Leia e estude este material de apoio para que tenha o suporte necessário no que diz respeito aos conteúdos apresentados sobre as tabelas de espalhamento.

Nesta aula, você vai estudar e compreender como funciona o armazenamento, a recuperação, o agrupamento de dados e a implementação de dicionários em tabelas de espalhamento. Para começar a dar significado a este conteúdo, vamos abordar um cenário profissional no qual você possa compreender que é possível aplicar as técnicas mencionadas.

Suponha o seguinte cenário: você trabalha em uma empresa de logística responsável por fornecer produtos a uma variedade de clientes. Eles podem ser lojas de varejo, restaurantes ou empresas de diversos setores. Cada cliente faz pedidos regularmente, que são identificados por números de referência. Um cliente pode fazer um pedido com o número de referência “1234”, e outro cliente pode ter um pedido com o mesmo número, já que eles usam sistemas de gerenciamento de pedidos independentes.

Diante do cenário apresentado, qual é a solução para esse desafio e como podemos assegurar que os pedidos de cada cliente sejam devidamente organizados, evitando quaisquer problemas durante o processo de distribuição?

Convidamos você a estudar este conteúdo e conhecer um pouco mais sobre o agrupamento de dados em tabelas de espalhamento, presente na disciplina de Algoritmos e Estrutura de Dados.

Vamos Começar!

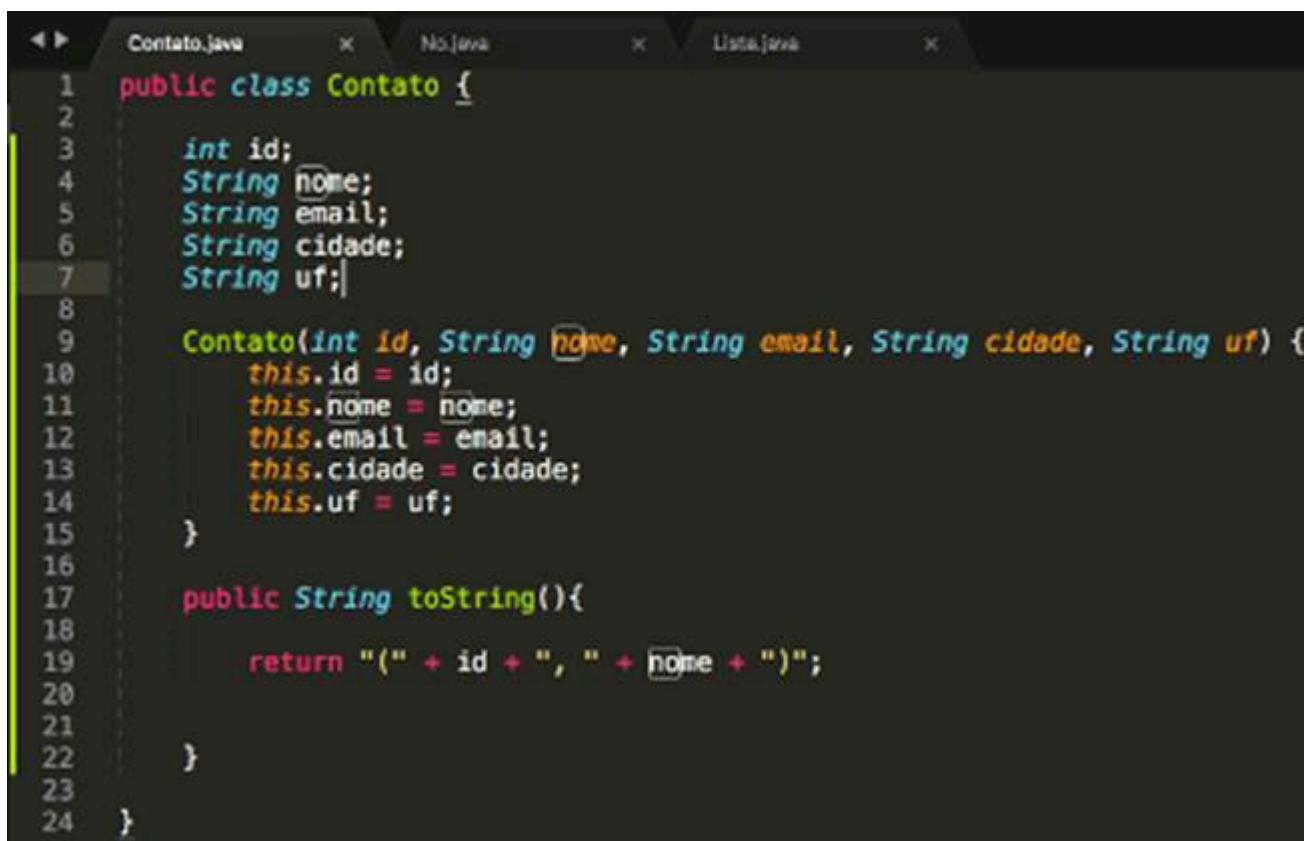
ALGORITMOS E ESTRUTURA DE DADOS

Nesta aula, você vai conhecer as técnicas de armazenamento e recuperação de dados em tabelas de espalhamento. Saberá também como agrupar dados e como implementar dicionários de dados em tais tabelas. Vamos ainda revisar alguns conteúdos que envolvem os algoritmos e, principalmente, a função de espalhamento. Iniciemos o conteúdo.

Armazenamento e recuperação de dados em tabelas de espalhamento

Partiremos para a nossa demonstração prática. Nesse cenário, lidaremos com um tipo específico de informação, a qual será customizada pelo usuário. A nossa tarefa consiste em gerenciar a inserção de contatos que serão armazenados em uma tabela *hash*, um exemplo clássico associado a essas tabelas. Este exemplo será conduzido por meio da linguagem de programação Java. A nossa aplicação será composta por quatro classes distintas: a Classe Contato, a Classe Nó, a Classe Lista e a Classe Hash.

No contexto da Classe Contato, teremos a definição de alguns atributos essenciais, como identificação (ID), nome, endereço de e-mail, cidade e unidade federativa (UF). É importante mencionar que, para a criação desse exemplo, utilizaremos a versão 3.2.2 do editor de texto Sublime, conforme ilustrado na Figura 1.

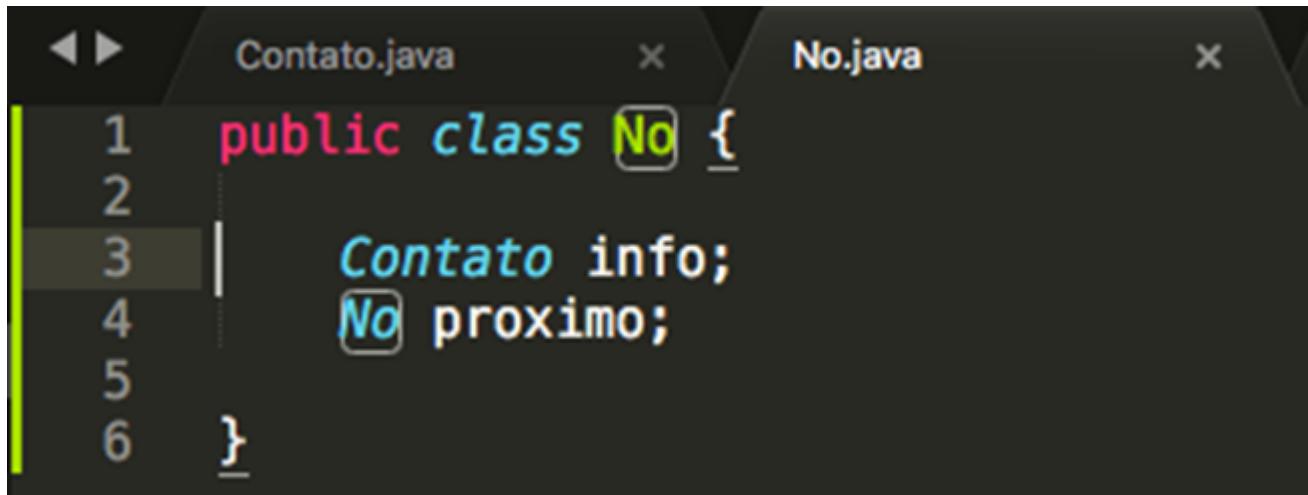


```
Contato.java      No.java      Lista.java
1  public class Contato {
2
3      int id;
4      String nome;
5      String email;
6      String cidade;
7      String uf;
8
9      Contato(int id, String nome, String email, String cidade, String uf) {
10         this.id = id;
11         this.nome = nome;
12         this.email = email;
13         this.cidade = cidade;
14         this.uf = uf;
15     }
16
17     public String toString(){
18
19         return "(" + id + ", " + nome + ")";
20
21     }
22
23 }
24 }
```

Figura 1 | Classe Contato.java.

ALGORITMOS E ESTRUTURA DE DADOS

A Figura 1 ilustra a elaboração da classe Contato, a qual é equipada com os atributos mencionados. Além disso, o método construtor é responsável por inicializar os valores no momento em que a instância da classe é criada.



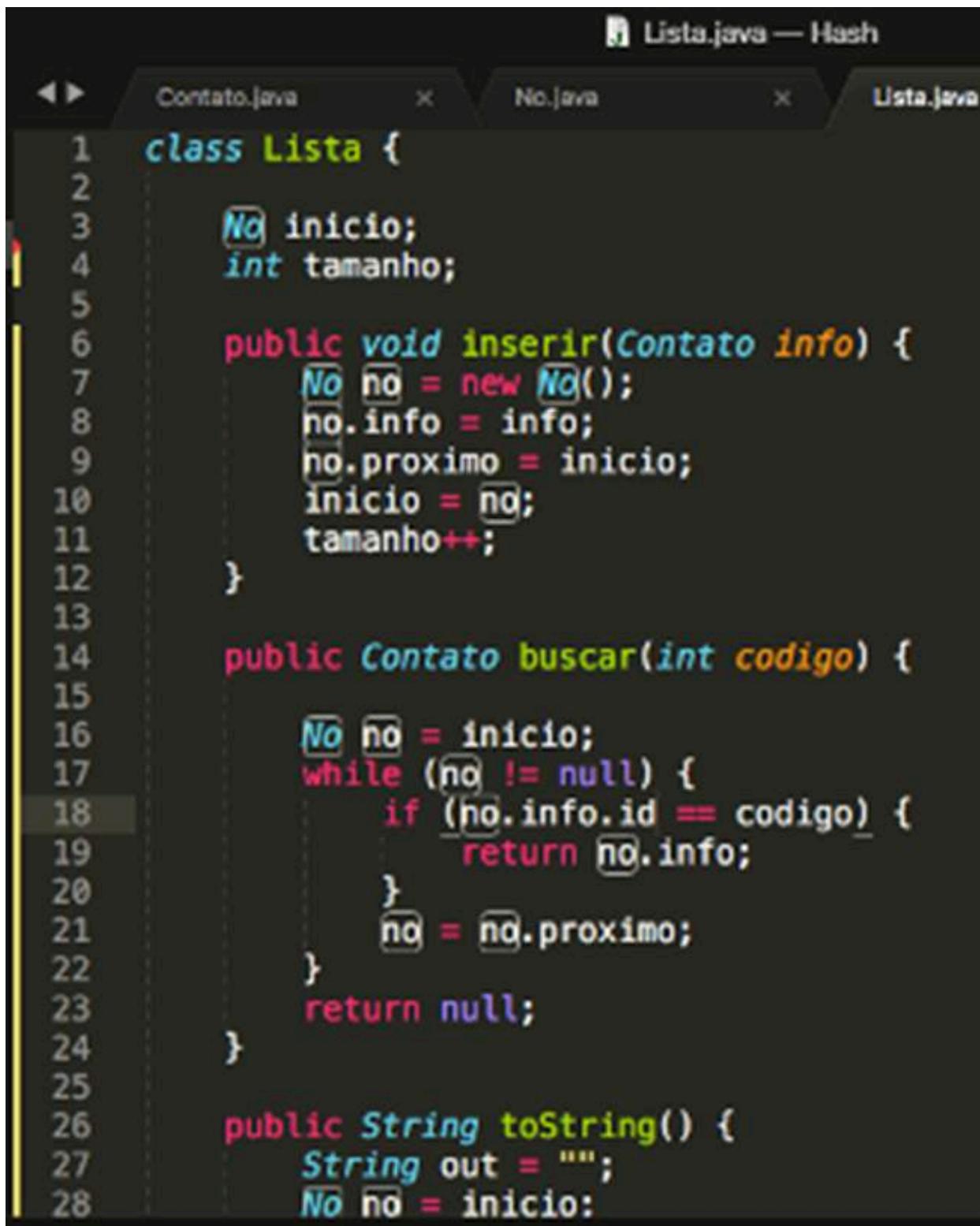
```
1 public class No {
2
3     Contato info;
4     No proximo;
5
6 }
```

Figura 2 | Classe No.java.

A Figura 2 nos mostra a Classe No, que desempenha a função de criar um atributo da classe Contato, juntamente com outro atributo do tipo No.

Agora, avançaremos para a introdução das duas partes restantes da estrutura do nosso exemplo. Estas partes incluem a Classe Lista e a Classe Hash. A Classe Lista possui uma lista encadeada e incorpora funcionalidades como o método “inserir()”, que é encarregado de adicionar elementos. A Lista também inclui um método para a busca de elementos com base no parâmetro “código”, e um terceiro método que exibe os elementos contidos na lista. Para uma visualização mais detalhada, consulte a Figura 3.

ALGORITMOS E ESTRUTURA DE DADOS

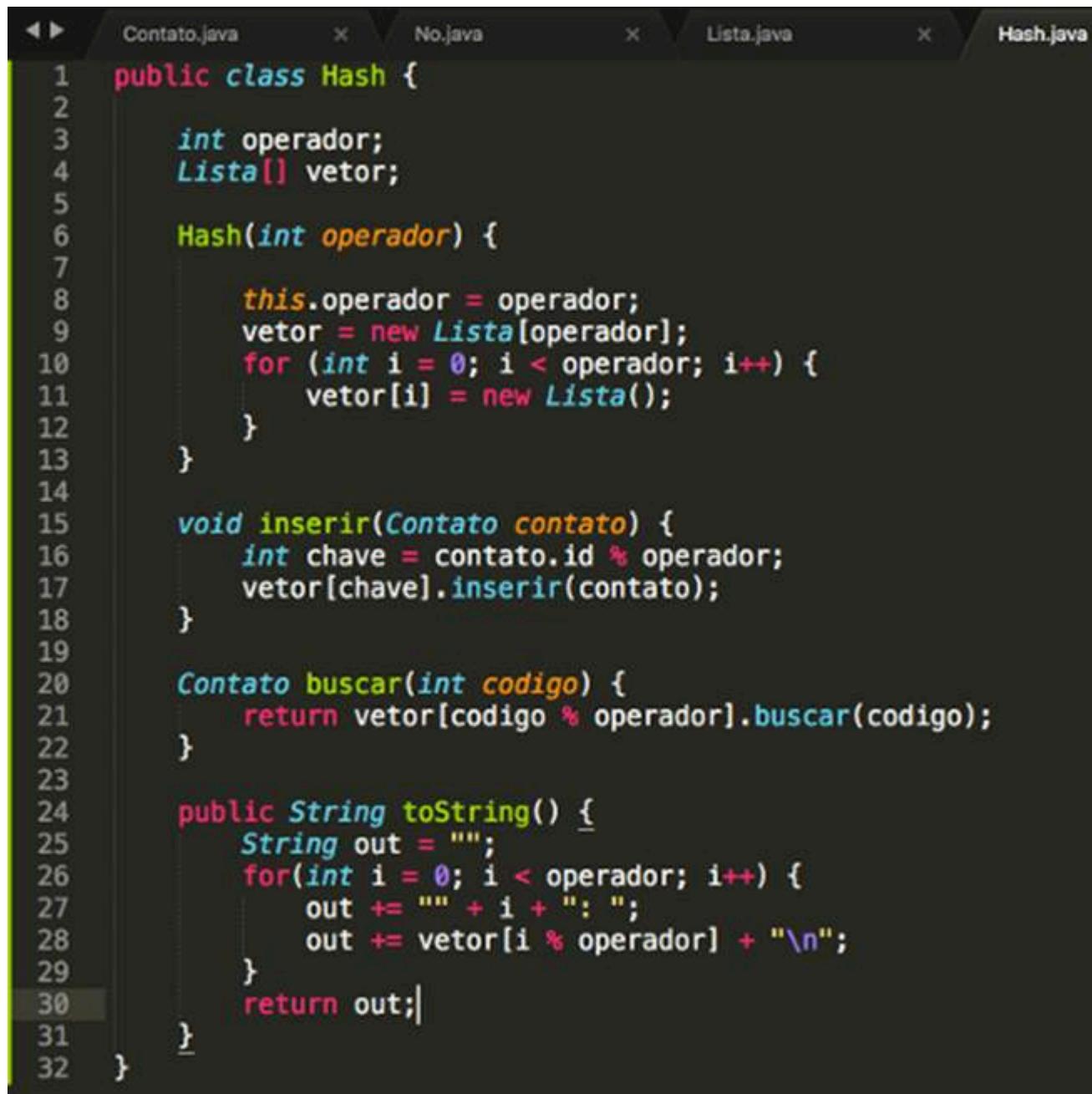


```
1 class Lista {
2
3     No inicio;
4     int tamanho;
5
6     public void inserir(Contato info) {
7         No no = new No();
8         no.info = info;
9         no.proximo = inicio;
10        inicio = no;
11        tamanho++;
12    }
13
14    public Contato buscar(int codigo) {
15
16        No no = 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;
```

Figura 3 | Classe Lista.java.

ALGORITMOS E ESTRUTURA DE DADOS

Na Figura 4, encontramos o código da Classe Hash, a qual desempenha a função de implementar a estrutura da tabela *hash*. No interior desta classe, é criado um *array* que faz referência à Classe Lista. No contexto da classe de testes, um vetor é criado de acordo com o valor do operador. A posição na qual o contato será inserido no vetor é determinada com base no resultado do cálculo do resto da divisão.



```
Contato.java      No.java      Lista.java      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     }
31 }
32 }
```

Figura 4 | Classe Hash.java.

ALGORITMOS E ESTRUTURA DE DADOS

Após a implementação das Classes Lista e Hash, basta criar a classe principal para executar os métodos “inserir()” e “buscar()”, que estão na Classe Hash.

Siga em Frente...

Agrupamento de dados em tabelas de espalhamento

O agrupamento de dados em tabelas de espalhamento refere-se a uma estratégia utilizada para organizar elementos ou registros de dados em compartimentos específicos dentro de uma tabela de espalhamento ou *hash tables*. Essa organização é realizada com base em critérios específicos, como características comuns ou relacionamentos entre os dados. O principal objetivo do agrupamento é facilitar a busca e recuperação eficiente de informações semelhantes ou relacionadas, uma vez que os dados são distribuídos em “*buckets*” (compartimentos) com base em alguma função de espalhamento.

Esse processo permite otimizar operações de busca e manipulação de dados, uma vez que os elementos com características compartilhadas estão agrupados em um mesmo compartimento. Isso reduz o tempo de acesso aos dados e pode melhorar o desempenho de algoritmos que dependem de tabelas de espalhamento, como a resolução de colisões, a otimização de buscas e a organização de informações de forma eficiente.

Para exemplificar melhor o agrupamento de dados em tabelas de espalhamento, vamos trabalhar com um cenário da realidade profissional em que este agrupamento é fundamental para a construção de um algoritmo eficiente.

- **Cenário:** Suponha que uma distribuidora de produtos, que atenda uma vasta gama de clientes em diferentes setores, está passando pela seguinte situação. A empresa lida com uma enorme quantidade de pedidos todos os dias, variando desde pequenas empresas locais até grandes varejistas regionais. A fim de atender a essa demanda, a distribuidora precisa otimizar suas operações de entrega para garantir que os produtos cheguem aos clientes no prazo e de forma eficiente. A empresa enfrenta um desafio complexo de agrupar pedidos de clientes que compartilham um mesmo destino de entrega. Isso é fundamental para reduzir os custos de transporte e melhorar a eficiência das entregas. O objetivo é agrupar pedidos de maneira a minimizar o número de veículos necessários para a entrega, reduzir o tempo de trânsito e, ao mesmo tempo, atender às restrições de horário de entrega dos clientes.
- **Solução proposta:** Para resolver esse problema, a distribuidora decide implementar uma tabela de espalhamento que permita o agrupamento eficiente de pedidos de entrega com base na localização dos clientes. Cada pedido é representado como um elemento na tabela de espalhamento, e a função de espalhamento é projetada de forma a distribuir pedidos em grupos com base em critérios de proximidade geográfica. A tabela de espalhamento é composta por uma série de “*buckets*”, de modo que cada um representa uma área

ALGORITMOS E ESTRUTURA DE DADOS

geográfica específica. Os pedidos são atribuídos a um “bucket” com base na localização do cliente de destino. Dessa forma, pedidos com destinos próximos são agrupados no mesmo “bucket”. A função de espalhamento é projetada para minimizar colisões, ou seja, garantir que não haja conflitos entre pedidos em compartimentos diferentes. Um exemplo para implementar a solução para a distribuidora de produtos está descrito a seguir:

1. **Implementação da tabela de espalhamento:** a distribuidora implementa uma tabela de espalhamento para representar os pedidos de entrega. Cada pedido é mapeado para um “bucket” na tabela com base na localização do cliente de destino.
2. **Função de espalhamento:** a função de espalhamento é projetada para distribuir os pedidos de forma eficiente nos compartimentos. Isso é feito levando em consideração critérios de proximidade geográfica, como a distância entre os endereços de entrega.
3. **Agrupamento de pedidos:** os pedidos são agrupados nos “buckets” com base na função de espalhamento. Isso resulta em pedidos com destinos próximos sendo armazenados no mesmo compartimento.
4. **Otimização das entregas:** quando chega a hora de fazer as entregas, a distribuidora otimiza as rotas dos veículos de transporte. Os pedidos armazenados em um mesmo “bucket” são entregues conjuntamente, o que reduz o número de veículos necessários e diminui o tempo de trânsito.
5. **Redução de custos:** a implementação desse sistema de agrupamento de pedidos resulta em uma redução significativa nos custos de transporte, uma vez que menos veículos são necessários para realizar as entregas. Isso economiza combustível, tempo e recursos da empresa.

O uso de tabelas de espalhamento para agrupar pedidos de entrega com base na localização dos clientes se mostra uma solução eficaz para o desafio enfrentado pela distribuidora. A otimização das entregas não somente reduz custos operacionais, como também melhora a eficiência da prestação de serviços, resultando em clientes mais satisfeitos. A maior satisfação dos clientes significa que a confiabilidade na empresa também aumenta. Essa abordagem demonstra como a aplicação prática de tabelas de espalhamento pode resolver problemas reais e impactar positivamente as operações de negócios de uma empresa.

O agrupamento de dados em tabelas de espalhamento é uma técnica valiosa em ciência da computação e é amplamente utilizada em várias aplicações, incluindo bancos de dados, sistemas de gerenciamento de informações, processamento de dados e muito mais.

Implementação de dicionário em tabelas de espalhamento

Os dicionários de dados em tabelas de espalhamento são estruturas de dados que permitem o armazenamento e a recuperação eficiente de informações, com base em uma chave associada a cada elemento de dados. Eles são usados para mapear chaves e valores, de modo que a recuperação de informações seja rápida e eficiente.

ALGORITMOS E ESTRUTURA DE DADOS

Nesse contexto, um dicionário de dados é uma coleção de pares, ou seja, “chave e valor”, em que cada chave é exclusiva e está associada a um valor específico. A tabela de espalhamento é um mecanismo usado para implementar esse tipo de estrutura de dados.

Quando os dados são armazenados em um dicionário de dados, em uma tabela de espalhamento, uma função de espalhamento é usada para calcular o índice (ou endereço) em que um determinado par “chave e valor” será armazenado. Isso permite uma recuperação rápida, pois, em vez de percorrer todos os elementos em busca de uma chave específica, a função de espalhamento direciona a busca para a posição na qual a informação deve estar armazenada. Essa abordagem é muito eficiente para a busca e recuperação de informações em grande escala.

Dicionários de dados em tabelas de espalhamento são amplamente utilizados em algoritmos, sistemas de gerenciamento de bancos de dados, implementação de sistemas e em muitas outras aplicações que envolvam a necessidade de associar chaves a valores para facilitar a recuperação de informações.

Um exemplo prático de dicionário de dados em uma tabela de espalhamento pode ser o armazenamento de informações de contato, como números de telefone, em uma espécie de agenda de contatos. Neste caso, cada contato pode ser representado por um par “chave e valor”, em que a chave é o nome da pessoa e o valor é o número de telefone associado a ela.

Um esquema para implementar esta solução em pseudolinguagem é exibido na Figura 5 apresentada a seguir.

```
1. # Criar uma tabela de espalhamento para armazenar contatos
2. tabela_contatos = TabelaEspalhamento()

3. # Inserir contatos na tabela
4. tabela_contatos.inserir("Professor", "45-99999-9999")
5. tabela_contatos.inserir("Advogado", "45-99999-9999")
6. tabela_contatos.inserir("Médico", "45-99999-9999")

7. # Recuperar o número de telefone de um contato
8. numero_telefone = tabela_contatos.recuperar("Professor")
9. imprimir("Número de telefone de Professor:", numero_telefone)
```

Figura 5 | Pseudocódigo para lista de contatos.

ALGORITMOS E ESTRUTURA DE DADOS

Em conclusão, os dicionários de dados desempenham um papel vital no mundo da programação e da ciência da computação, simplificando a organização e auxiliando a recuperação de informações de modo inteligente. Isso torna os processos mais eficientes e proporciona uma base sólida para o desenvolvimento de aplicativos e sistemas de alta qualidade.

Esperamos que você estude o que foi abordado nesta aula e acesse as indicações de estudo. Desta maneira, você contribuirá com seu aprendizado, obtendo um entendimento sólido sobre os conceitos estudados.

Lembre-se de que o conhecimento adquirido aqui pode ser aplicado em diversas áreas de conhecimento. Continue explorando e aprimorando suas habilidades, pois a tecnologia e a ciência da informação oferecem um vasto campo de possibilidades.

Vamos Exercitar?

Voltando ao nosso cenário proposto inicialmente, imagine que você trabalha em uma empresa de logística responsável por fornecer produtos a uma variedade de clientes, que podem ser lojas de varejo, restaurantes ou empresas de diversos setores. Cada cliente faz pedidos regularmente, e esses pedidos são identificados por números de referência. Por exemplo, um cliente pode fazer um pedido com o número de referência “1234”, e outro cliente pode ter um pedido com o mesmo número, já que eles usam sistemas de gerenciamento de pedidos independentes.

Aqui reside o desafio: ao receber esses pedidos, você precisa garantir que eles sejam distribuídos corretamente. Afinal, a loja “1234” não deve receber pedidos destinados a um restaurante com o mesmo número de referência. Portanto, é vital agrupar os pedidos de cada cliente de maneira organizada e precisa.

Nesse caso, a tabela de espalhamento entra em jogo. Essa estrutura de dados pode ajudar a resolver o problema, mas é preciso projetar cuidadosamente o processo para garantir que os pedidos sejam agrupados corretamente, evitando confusões e a distribuição incorreta dos produtos. Como fazer isso de maneira eficaz é o foco desta situação-problema.

Agora é com você! Para resolver o problema de agrupamento de pedidos de clientes em uma tabela de espalhamento, siga os seguintes passos:

- 1. Identificação única:** garanta que cada pedido tenha uma identificação única, especialmente se pertencer a diferentes clientes. Isso pode ser alcançado adicionando um prefixo com o código do cliente ao número de identificação. Por exemplo, se um cliente com código “A” fizer um pedido com o número “1234”, esse número pode ser transformado em “A1234”.
- 2. Função de espalhamento:** projete uma função de espalhamento eficiente que leve em consideração a identificação única de cada pedido. A função deve distribuir os pedidos nas tabelas de espalhamento de forma a minimizar colisões entre pedidos de clientes

ALGORITMOS E ESTRUTURA DE DADOS

diferentes. Uma estratégia comum é usar o código do cliente como parte da função de espalhamento.

3. **Tabelas de espalhamento separadas:** crie tabelas de espalhamento separadas para cada cliente. Isso garante que os pedidos sejam agrupados por cliente e evita conflitos entre pedidos de clientes diferentes. Por exemplo, o cliente "A" terá sua própria tabela, o cliente "B" terá outra tabela e assim por diante.
4. **Colisões locais:** dentro de cada tabela de espalhamento, se ocorrerem colisões entre pedidos do mesmo cliente, você pode lidar com elas usando métodos tradicionais de tratamento de colisões, como encadeamento separado ou sondagem linear. No entanto, como os pedidos são agrupados por cliente, as colisões são menos prováveis.

Ao seguir essas etapas, você pode garantir que os pedidos de cada cliente sejam agrupados adequadamente, evitando confusões na distribuição. Cada cliente terá sua própria tabela de espalhamento, e a função de espalhamento cuidará para que os pedidos sejam alocados corretamente. Essa abordagem é fundamental para manter a eficiência e a organização no processo de distribuição de produtos em empresas de logística.

Saiba mais

Para compreender mais sobre o conteúdo desta aula, acesse a pesquisa: "*Visual TaHs: software para auxiliar o ensino de tabelas Hash na disciplina de Estrutura de Dados*", de Fábio Carlos Moreno, Cinthyan R. Sachs C de Barbosa e Edio Roberto Manfio, publicado em 2019 nos anais do Seminário Integrado de Software e Hardware (SEMISH). A pesquisa apresenta o software "Visual TaHs", que poderá contribuir para o ensino de estrutura de dados, com base na abordagem da contextualização da tabela *hash*. Foi feito um experimento com quatorze funções *hash* implementadas no software, sendo realizada uma sobrecarga de elementos no léxico. Este experimento possibilitou, através dos relatórios e gráficos, analisar e identificar as melhores funções de espalhamento. Todas as análises realizadas poderão ser avaliadas pelos discentes, contextualizando o conceito trabalhado em sala de aula.

MORENO, F. C.; DE BARBOSA, C. R. S. C.; MANFIO, E. R. [Visual TaHs: software para auxiliar o ensino de tabelas Hash na disciplina de Estrutura de Dados](#). In: **Seminário Integrado de Software e Hardware (SEMISH)**, 46, 2019, Belém. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 33-44.

Bons estudos!

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

ALGORITMOS E ESTRUTURA DE DADOS

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

MORENO, F. C.; DE BARBOSA, C. R. S. C.; MANFIO, E. R. Visual TaHs: software para auxiliar o ensino de tabelas Hash na disciplina de Estrutura de Dados. In: **Seminário Integrado de Software e Hardware (SEMISH)**, 46, 2019, Belém. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 33-44. Disponível em: <https://doi.org/10.5753/semish.2019.6565>. Acesso em: 23 nov. 2023.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, Aarom M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, Nivio. **Projeto de algoritmos: com implementações em Pascal e C**. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 5

Encerramento da Unidade

Videoaula de Encerramento

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Chegada

ALGORITMOS E ESTRUTURA DE DADOS

Olá, estudante! Para desenvolver a competência desta Unidade (“entender e aplicar os conceitos fundamentais relacionados a tabelas de espalhamento, bem como ao processo de espalhamento (*hashing*)”), você deverá primeiramente conhecer os conceitos fundamentais abordados ao longo desta Unidade de ensino.

Uma das competências essenciais é a capacidade de entender e identificar as situações em que aplicamos a função de espalhamento em um determinado conjunto de dados. Ao longo da Unidade, exploramos a sua definição, seus componentes, a forma como opera, as estratégias para aprimorar sua eficiência, além de como lidar com colisões durante o processo de *hashing*, isto é, a aplicação da função de espalhamento.

O algoritmo que envolve o processo de espalhamento é fundamental para o sucesso da organização dos elementos, pois determina como as chaves são associadas a índices na tabela, afetando diretamente a eficiência, a segurança e a confiabilidade das operações de busca e manipulação de dados.

Conforme progredimos no estudo, nos deparamos com desafios do mundo real, nos quais a tabela de espalhamento se revela crucial. Você desenvolveu a capacidade de analisar a aplicação dessas estruturas em uma variedade de cenários, desde o planejamento inicial até a gestão de diversos tipos de elementos.

Desenvolver essas habilidades é o primeiro passo para que você consiga criar soluções mais eficientes, contribuindo com seu aprendizado. Assim, você estará pronto para novos desafios em desenvolvimento de sistemas. Ao dominar a arte de decidir quando e como implementar as estruturas de tabela de espalhamento, você estará bem-preparado para prosseguir na sua carreira profissional no campo da tecnologia.

É Hora de Praticar!



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Convidamos você estudante a colocar em prática o conhecimento adquirido nesta Unidade de ensino. Para isso, vamos abordar um estudo de caso com o cenário descrito a seguir: **Gestão de estoque em uma loja de varejo - Centro de distribuição de varejo**.

ALGORITMOS E ESTRUTURA DE DADOS



Figura 1 | Exemplo de um centro de distribuição.

- **Cenário:** uma loja de varejo deseja aprimorar a gestão de seu estoque para otimizar o atendimento ao cliente. Eles têm um grande número de produtos e desejam criar um sistema eficiente de gestão de estoque usando tabelas de espalhamento (*hash tables*) para acelerar a busca e a atualização dos produtos.
- **Objetivo:** o objetivo do estudo de caso é demonstrar como o processo de atendimento ao cliente pode ser melhorado otimizado com o conceito de tabelas de espalhamento.
- **Passos para resolução:**
 1. Para implementar um sistema de gestão de estoque, primeiro, é necessário criar a estrutura da tabela de espalhamento. A tabela deve ser projetada para armazenar informações sobre os produtos, como nome, código de barras, quantidade em estoque e preço. Cada produto será associado a uma chave única, que é gerada a partir do código de barras.
 2. À medida que novos produtos chegam à loja, eles são adicionados à tabela de espalhamento. O código de barras do produto é usado como chave para inserção na tabela, e as informações relevantes são associadas a essa chave.

ALGORITMOS E ESTRUTURA DE DADOS

3. Quando um cliente deseja comprar um produto, a busca na tabela de espalhamento é acionada usando o código de barras do produto como chave. Isso permite uma busca instantânea, fornecendo informações sobre o produto, como preço e disponibilidade em estoque.
4. Após uma venda, o estoque do produto é atualizado na tabela de espalhamento. A quantidade em estoque é diminuída, garantindo que o sistema reflita com precisão a disponibilidade de produtos.
5. Em cenários nos quais produtos diferentes compartilham o mesmo código de barras, é necessário lidar com colisões. Uma abordagem eficaz é usar técnicas de encadeamento para tratar de várias entradas na mesma posição da tabela.

Crie sua solução algorítmica, preferencialmente na linguagem de programação Java. Bom trabalho!

- O que é necessário para garantir a segurança das informações pessoais em serviços online que utilizam tabelas de espalhamento? Quais medidas você considera importantes para proteger suas próprias informações?
- Você já considerou como as tabelas de espalhamento podem ser aplicadas em áreas fora da tecnologia da informação? Em que outros campos ou situações você acredita que a estrutura de tabela de espalhamento poderia ser benéfica e por quê?

Tente exercitar empenhando-se em refletir e em responder a estas questões.

Sugestão de solução

- **Passo 1:** crie uma tabela com uma função de espalhamento adequada, que utilize o código de barras como chave. Defina a estrutura de dados para armazenar informações sobre os produtos.
- **Passo 2:** ao receber novos produtos, insira-os na tabela de espalhamento, associando as informações do produto à chave gerada a partir do código de barras.
- **Passo 3:** para buscar um produto, utilize o código de barras como chave na tabela de espalhamento. Acesse as informações do produto, incluindo preço e quantidade em estoque.
- **Passo 4:** após cada venda, atualize, na tabela de espalhamento, a quantidade do produto em estoque.
- **Passo 5:** caso ocorram colisões (ou seja, produtos diferentes com o mesmo código de barras), utilize técnicas de encadeamento separado para gerenciar várias entradas na mesma posição da tabela.

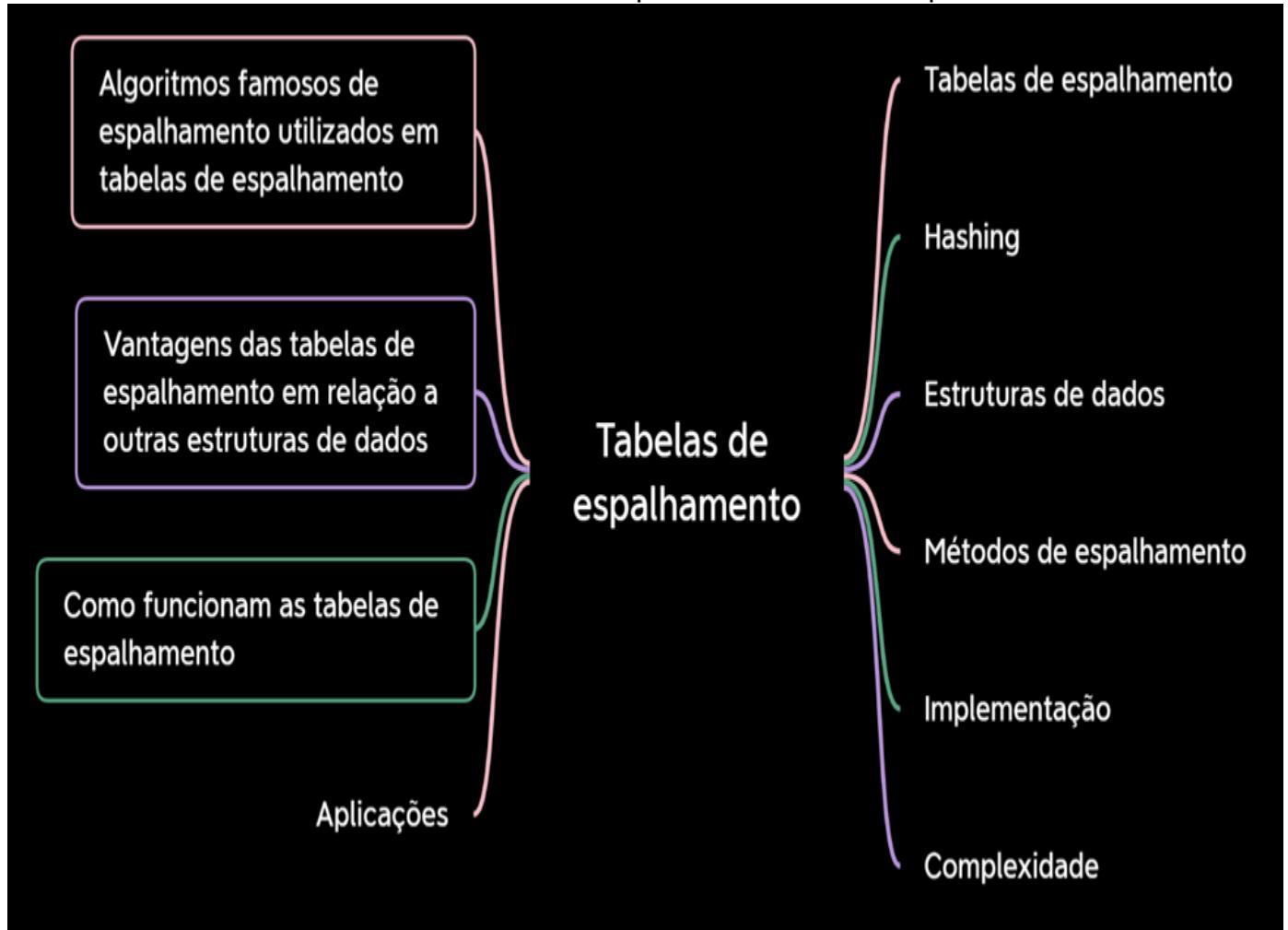
Esse estudo de caso demonstra como as tabelas de espalhamento podem ser aplicadas para aprimorar a gestão de estoque em uma loja de varejo, oferecendo busca rápida e atualização eficiente dos produtos.

No mapa mental a seguir, são apresentadas resumidamente algumas características em relação ao aprendizado das tabelas de espalhamento, bem como conceitos e termos geralmente

ALGORITMOS E ESTRUTURA DE DADOS

utilizados.

Convidamos você a refletir sobre os conceitos apresentados neste mapa mental!



CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos: com implementações em Pascal e C**. 3^a ed. São Paulo: Cengage Learning, 2011.

Unidade 4

Armazenamento Associativo

ALGORITMOS E ESTRUTURA DE DADOS

Aula 1

Definição e usos de Mapas de Armazenamento

Definição e usos de mapas de armazenamento

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante! Esperamos que aproveite bastante esta aula, pois os temas discutidos serão muito relevantes para seu entendimento de algoritmos e estruturas de dados.

Estamos prestes a introduzir uma nova forma de organização de dados nesta disciplina. Da mesma forma que outras estruturas de dados, o armazenamento associativo também pode oferecer inúmeras vantagens que, se empregadas corretamente, serão significativas na criação de algoritmos.

Durante esta aula, faremos uma introdução ao armazenamento associativo, seus principais conceitos teóricos e as vantagens do seu uso em algoritmos. Você também vai compreender como os mapas associativos funcionam e de que maneira esta estrutura de dado pode auxiliar o profissional de desenvolvimento de sistemas a prover soluções algorítmicas de mercado.

Vamos para um exemplo prático? Imagine o seguinte problema: em uma empresa de e-commerce, a equipe de desenvolvimento está buscando otimizar a busca por produtos em seu sistema. Eles desejam implementar um método de busca eficiente, utilizando o armazenamento associativo para organizar e recuperar rapidamente informações sobre os produtos disponíveis.

ALGORITMOS E ESTRUTURA DE DADOS

Você foi escalado para gerenciar esse processo. Reflita sobre esta situação, pois, no final da aula, você vai propor uma solução dentro do conceito de armazenamento associativo.

Depois de explorar os princípios desta estrutura de dados e observar exemplos de como esses algoritmos funcionam, você estará pronto para criar soluções que resolvam desafios do mundo profissional utilizando armazenamento associativo e mapas associativos.

Convidamos você a descobrir mais essa estrutura de dados e compreender como ela pode ser valiosa no desenvolvimento de algoritmos computacionais.

Vamos Começar!

Nesta aula, você vai estudar o armazenamento associativo, suas principais características e funcionamento, além das vantagens do seu uso em estruturas de dados. Assim, vamos iniciar com algumas definições.

Um mapa associativo é uma estrutura de dados que armazena uma coleção de pares chave-valor; cada chave é única, portanto, está associada a um único valor. O mapa associativo permite a rápida recuperação de um valor específico por meio da sua chave correspondente, proporcionando agilidade e eficiência na busca e manipulação de dados (Cormen, 2022).

Introdução ao armazenamento associativo

O armazenamento associativo é uma técnica dentro do campo da ciência da computação, que oferece uma maneira eficaz de associar chaves a valores. Imagine um catálogo de uma biblioteca, no qual cada livro é identificado por um número único. Esse sistema é semelhante ao armazenamento associativo, no qual as chaves são os identificadores únicos que direcionam a seus respectivos valores.

O armazenamento associativo é uma estrutura de dados robusta capaz de dispor várias formas de implementação com o intuito de gerenciar e controlar como o armazenamento será atribuído posteriormente a uma linguagem de programação.

Essa estrutura tem como base tabelas associativas para a sua construção dentro do desenvolvimento de sistemas. As tabelas associativas têm um papel fundamental em diversas áreas, sendo amplamente empregadas na construção de tabelas de símbolos. Elas desempenham um papel importante em compiladores e montadores, além de serem essenciais em sistemas operacionais, para gerenciar informações vitais, como tabelas de arquivos e processos necessários para a execução de programas. Na representação da Figura 1, vemos um exemplo ilustrativo de uma tabela que associa caracteres a sequências de sinais curtos ou longos, como é o caso da tabela de código Morse.

ALGORITMOS E ESTRUTURA DE DADOS

A	--	J	-----	S	---	2	-----
B	-...	K	-..	T	-	3	-----
C	---	L	...-	U	..-	4	-----
D	-..	M	--	V	...-	5
E	.	N	--	W	---	6
F	---	O	---	X	...-	7
G	---	P-	Y	...-	8	----
H	Q	----	Z	---	9	-----
I	..	R	---	1	-----	0	-----

Figura 1 | Tabela de código Morse. Fonte: Souza e Nery Filho (2017).

Um exemplo prático de armazenamento associativo remete à lista de presença de uma turma da pós-graduação. Essa lista é organizada alfabeticamente, atribuindo a cada aluno um número de matrícula que o identifica unicamente. Essa associação entre um número e o nome do aluno representa uma tabela associativa e faz parte do cenário do armazenamento associativo, de modo que é possível relacionar diretamente cada aluno a um número específico.

Em um cenário da compra de um produto (por exemplo, um notebook) por meio de uma loja online, o armazenamento associativo desempenha um papel fundamental na organização e recuperação de informações. Imagine um site de e-commerce oferecendo uma variedade de notebooks. Cada modelo de notebook possui um identificador único, sendo associado a um conjunto de detalhes, como marca, especificações técnicas e o seu preço.

Ao realizar, no site, uma busca por um determinado modelo, o sistema utiliza essa estrutura associativa. O identificador único atua como chave, permitindo ao sistema acessar rapidamente todas as informações relevantes, como marca, configurações específicas e preço atualizado. Esse método ágil e eficiente de armazenamento e recuperação de dados permite uma experiência de compra online mais dinâmica e simplificada para o usuário, oferecendo informações precisas sobre os produtos disponíveis.

ALGORITMOS E ESTRUTURA DE DADOS

O número de possibilidades para a utilização de associação nos dias de hoje é imenso. Outro exemplo é o controle de veículos que é feito pelo Detran de cada estado. Para cada veículo, existe um conjunto de informações diretamente vinculadas à placa, como: nome do proprietário, endereço, número da carteira nacional de habilitação (CNH), entre outras informações. Observe a Figura 2, que mostra uma CNH, a qual também reúne várias informações do motorista, como: nome, CPF, data da 1^a habilitação, vencimento etc.



Figura 2 | Carteira nacional de habilitação. Fonte: Shutterstock.

Esses e outros cenários são possíveis de implementar utilizando o conceito de armazenamento associativo.

Vantagens do armazenamento associativo

As vantagens do armazenamento associativo são numerosas. A principal é a rápida recuperação de dados. Isso se deve à capacidade de a estrutura associativa mapear chaves e valores, permitindo acesso direto aos dados desejados sem percorrer toda a estrutura. É especialmente útil em aplicações que lidam com grandes volumes de informações. Em um contexto profissional, sistemas de recomendação em plataformas de streaming são um excelente exemplo. Eles utilizam mapas associativos para associar preferências de conteúdo (chaves) aos usuários (valores), proporcionando sugestões personalizadas de vídeos ou músicas.

Outra vantagem é a flexibilidade na organização dos dados. Ao contrário de algumas estruturas de dados lineares, os mapas associativos permitem que os dados sejam organizados de forma

ALGORITMOS E ESTRUTURA DE DADOS

mais complexa e hierárquica. Por exemplo, em sistemas de gerenciamento de estoque, pode-se associar um código de produto (chave) a uma estrutura de dados complexa contendo informações detalhadas sobre o produto (valor). A flexibilidade também proporciona aos mapas associativos a associação de diferentes tipos de dados, como *strings*, números ou estruturas mais complexas. Isso facilita a organização de informações heterogêneas em um único local.

Outra vantagem está na sua adaptabilidade a diferentes contextos. Seja na organização de grandes bancos de dados ou no gerenciamento de informações em tempo real, a estrutura associativa pode ser ajustada para atender às necessidades específicas de cada situação.

Essas vantagens tornam o armazenamento associativo uma escolha valiosa em uma variedade de aplicações, desde sistemas de gerenciamento de banco de dados até a manipulação ágil de informações em tempo de execução em aplicativos e sistemas distribuídos. Observe ,na Figura 3, um exemplo de streaming utilizando armazenamento associativo.



Figura 3 | Aplicativo de streaming. Fonte: Shutterstock.

A Figura 4 apresenta um exemplo simples em Java. Ele demonstra uma vantagem do armazenamento associativo, utilizando-o para associar nomes a idades.

ALGORITMOS E ESTRUTURA DE DADOS

```
import java.util.HashMap;

public class ExemploArmazenamentoAssociativo {
    public static void main(String[] args) {
        // Criando um HashMap para associar nomes a idades
        HashMap<String, Integer> mapaIdades = new HashMap<>();

        // Adicionando valores ao mapa
        mapaIdades.put("João", 25);
        mapaIdades.put("Maria", 30);
        mapaIdades.put("Pedro", 22);

        // Acessando a idade de um nome específico
        int idadeMaria = mapaIdades.get("Maria");
        System.out.println("A idade de Maria é: " + idadeMaria);

        // Verificando se um nome está no mapa
        boolean contemPedro = mapaIdades.containsKey("Pedro");
        System.out.println("O mapa contém Pedro? " + contemPedro);
    }
}
```

Figura 4 | Código associando nomes a idades.

O código cria um *HashMap* chamado “mapaldades”, associando nomes (chaves do tipo *String*) a idades (valores do tipo *Integer*). Ele insere valores no mapa usando o método *put()*, atribuindo um nome a uma idade correspondente. Depois, utiliza o método *get()* para recuperar a idade de um nome específico e o método *containsKey()* para verificar se um nome está presente no mapa.

Essa implementação mostra como o armazenamento associativo, neste caso o *HashMap*, permite associar informações diretamente, facilitando a recuperação e verificação de dados baseados em chaves específicas, como nomes, neste exemplo.

ALGORITMOS E ESTRUTURA DE DADOS

Siga em Frente...

Funcionamento dos mapas associativos

Para compreender seu funcionamento, imagine uma biblioteca. Nela, cada livro é categorizado por um código único que permite aos funcionários localizar rapidamente qualquer um na estante.

No contexto dos mapas associativos, essa biblioteca seria parecida à estrutura de dados. Em vez de livros e códigos, temos pares de chave-valor. A chave é como o código do livro, única e identificadora; o valor é o conteúdo do livro, representando os dados associados àquela chave específica.

Por exemplo, considere um mapa associativo para um sistema de gerenciamento de clientes de uma loja online. Cada cliente possui um identificador exclusivo (a chave) e um conjunto de informações associadas (o valor), como nome, endereço, histórico de compras, etc. Ao adicionar um novo cliente ao sistema, uma chave única é atribuída a esse cliente, e todas as informações relevantes são associadas a essa chave.

Então, na nossa estrutura, precisamos de uma função que estabeleça conexões entre os elementos. Essa operação deve vincular uma chave a um valor para criar essa conexão. É crucial recuperar rapidamente o valor vinculado a uma chave específica. Uma operação precisa ser implementada para isso: ao receber uma chave, ela retorna o valor associado, caso exista. Às vezes, pode ser necessário remover uma conexão da estrutura. Para isso, devemos ter uma funcionalidade que, dada uma chave, elimine a associação correspondente. Outra operação importante seria verificar se uma chave está ou não associada a um valor. Além disso, é essencial saber a quantidade de associações presentes na estrutura. Um requisito essencial para o mapa é a unicidade das chaves. Não é permitido haver duas chaves idênticas, seja para carros com a mesma placa, canais de televisão com números iguais ou pedidos com números idênticos.

Os mapas são as estruturas de dados que implementam ou permitem implementar as seguintes funcionalidades:

1. Adicionar uma associação.
2. Recuperar um valor, dada uma chave.
3. Remover uma associação, dado um par valor-chave.
4. Verificar se existe uma associação para determinada chave.
5. Informar a quantidade de associações na estrutura.

Pretendemos criar uma estrutura de dados voltada para um sistema de registro de veículos, com possível aplicação no Detran. A ideia é que as informações específicas de um carro sejam acessadas por meio da sua placa. Dessa forma, o mapa que desejamos criar vai vincular cada

ALGORITMOS E ESTRUTURA DE DADOS

placa a um veículo correspondente. As placas serão tratadas como *Strings*, enquanto os veículos podem ser representados por uma classe Carro. Observe a Figura 5.



The screenshot shows a code editor with the file 'Carro.java' open. The code defines a class 'Carro' with private attributes for name, brand, color, and year, along with a constructor that initializes these attributes to specific values. The code is color-coded for syntax highlighting.

```
Carro.java > ...
1  public class Carro {
2
3      private String nome;
4      private String marca;
5      private String cor;
6      private int ano;
7
8      // getters e setters (encapsulamento)
9
10     public Carro(){
11         nome = "Polo";
12         marca = "Carro2";
13         cor = "branca";
14         ano = 2012;
15     }
16
17 }
18
```

Figura 5 | Classe Carro.

O ponto fundamental do mapa é a associação das chaves com os valores. Para modelar uma associação, vamos definir uma classe, Figura 6.

ALGORITMOS E ESTRUTURA DE DADOS

```
1  public class Associacao {  
2  
3      private String placa;  
4      private Carro carro;  
5  
6      public Associacao(String placa, Carro carro){  
7          this.placa = placa;  
8          this.carro = carro;  
9      }  
10  
11 }  
12
```

Figura 6 | Classe Associacao.

Esperamos que tenha ficado claro para você, neste primeiro, momento esses conceitos iniciais do armazenamento associativo.

Vamos Exercitar?

Vamos retomar nosso exemplo prático apresentado no início da aula! Em uma empresa de e-commerce, a equipe de desenvolvimento pretende otimizar a busca por produtos em seu sistema. Eles desejam implementar um método de busca eficiente, utilizando o armazenamento associativo para organizar e recuperar rapidamente informações sobre os produtos disponíveis.

Objetivo: implementar um sistema de busca por produtos que utilize armazenamento associativo para agilizar a recuperação de informações.

Descreva como você aplicaria os algoritmos de armazenamento associativo para o desenvolvimento desta aplicação.

Esta resolução trata apenas de forma descritiva os passos para solução da situação-problema, não levando em conta que o estudante, neste momento, tenha que desenvolver os algoritmos. Diante disso, vamos à resolução.

Na sequência, estão descritos os passos para implementar a solução da situação-problema:

ALGORITMOS E ESTRUTURA DE DADOS

1. **Preparação dos dados:** o primeiro passo é coletar informações sobre os produtos disponíveis, como nomes, códigos ou categorias. Outro passo é associar cada produto a uma chave única que facilite a busca.
2. **Implementação do armazenamento associativo:** escolher uma estrutura de armazenamento associativo adequada, como tabelas *hash* ou dicionários em linguagens de programação. Armazenar os produtos associados às chaves correspondentes.
3. **Busca eficiente:** desenvolver um algoritmo de busca que utilize a estrutura de armazenamento associativo para recuperar informações sobre os produtos de maneira rápida e precisa.
4. **Teste do sistema de busca:** realizar testes para verificar a eficácia do sistema de busca. Certificar-se de que a busca retorna os produtos corretos de maneira ágil.

A implementação bem-sucedida do armazenamento associativo nesse cenário de busca por produtos destaca a importância dessa estrutura de dados na otimização de operações de pesquisa. A habilidade de utilizar esse método não só agiliza as operações, mas também revela a relevância do conhecimento em estruturas de dados para a eficiência e melhoria de sistemas reais, como é o caso no desenvolvimento de soluções em empresas de comércio eletrônico.

Saiba mais

Para saber mais sobre pesquisas recentes de algoritmos para a manipulação de mapas na TerraLib, acesse o artigo com o título: "*Algoritmos exatos para manipulação de mapas na TerraLib*", de Vinícius Lopes Rodrigues, Ângelo Sousa Cavalier, Marcus Vinícius Alvim Andrade e Gilberto Ribeiro de Queiroz, do Departamento de Informática da Universidade Federal de Viçosa (UFV). O trabalho aborda algoritmos exatos (livre de erros de arredondamento) para a manipulação geométrica de dados geográficos. O principal foco deste artigo é o algoritmo de sobreposição de mapas, tendo como ambiente de desenvolvimento a TerraLib, uma biblioteca de componentes espaciais com suporte a algoritmos geométricos. Não deixe de conferir para compreender um pouco mais sobre este assunto.

RODRIGUES, V. L.; et al. [Algoritmos exatos para manipulação de mapas na TerraLib](#). In: **Simpósio Brasileiro de Geoinformática**, Campos do Jordão, 2005. p. 245-252.

Bons estudos!

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

ALGORITMOS E ESTRUTURA DE DADOS

RODRIGUES, V. L.; et al. Algoritmos exatos para manipulação de mapas na TerraLib. In: **Simpósio Brasileiro de Geoinformática**, Campos do Jordão, 2005. p. 245-252. Disponível em: <http://mtc-m16c.sid.inpe.br/col/dpi.inpe.br/geoinfo@80/2006/07.11.14.09/doc/P30.pdf>. Acesso em: 26 nov. 2023.

SOUZA, C. J. de.; NERY FILHO, J. Entre a escuridão e o silêncio: a relação entre as TICs e a surdocegueira utilizando a ferramenta do código Morse. Revista on line de Política e Gestão Educacional, Araraquara, v. 21, n. esp. 1, p. 881-895, out./2017. Disponível em: <https://periodicos.fclar.unesp.br/rpge/article/view/10458/6816>. Acesso em: 01 mar. 2024.

SZWARCFITER, J L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 2

Mapas com Lista

Mapas com lista



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

ALGORITMOS E ESTRUTURA DE DADOS

Olá, estudante! Esperamos que os temas discutidos nesta aula possam ser relevantes para seu aprendizado em algoritmos e estruturas de dados.

Vamos abordar uma outra maneira de implementar associações das informações. Agora vamos utilizar o conceito de mapas com lista.

Durante esta aula, serão abordados assuntos como: fundamentos de mapas com listas, manipulação de associações de chaves e validações pertinentes a esta estrutura. Você também vai conhecer como os mapas com listas funcionam e de que maneira esta estrutura de dado pode auxiliar o profissional de desenvolvimento de sistemas a prover soluções algorítmicas de mercado.

Vamos para um exemplo prático? Imagine o seguinte: um sistema de gerenciamento de estoque em uma loja de eletrônicos. Esse sistema precisa armazenar informações sobre os produtos disponíveis, como nome, preço e quantidade em estoque. Você foi escolhido para liderar este projeto. No final da aula, você vai propor uma solução dentro do conceito de armazenamento associativo usando mapas com listas.

Depois de explorar os princípios desta estrutura de dados e observar exemplos de como esses algoritmos funcionam, você estará pronto para criar soluções que resolvam desafios do mundo profissional, aplicando técnicas específicas dentro de mapas com listas.

Convidamos você a descobrir mais essa estrutura de dados e compreender como ela pode ser valiosa no desenvolvimento de algoritmos computacionais, criando soluções para os problemas sistêmicos do cotidiano.

Vamos Começar!

Nesta aula, você vai conhecer os mapas com lista, suas principais características e funcionamento, além do seu conceito e utilização em estruturas de dados. Assim, vamos iniciar com algumas definições.

Ao se aprofundar sobre o assunto, você vai compreender as estruturas dinâmicas fundamentais, bem como suas aplicações na solução de problemas em mapas com lista.

Nessa aula, você vai estudar:

- Definições e conceitos sobre mapas com listas.
- Operações em mapas.
- Manipulação de associações de chaves em mapas com listas.
- Validação de chaves em mapas com listas.

ALGORITMOS E ESTRUTURA DE DADOS

Introdução a mapas com listas

Mapas com listas são estruturas de dados que oferecem uma forma eficiente de armazenar e acessar informações por meio de associações entre chaves e valores. Essas estruturas são valiosas para mapear chaves com determinados elementos, proporcionando um acesso rápido e eficaz aos dados. Ao contrário de *arrays* ou listas convencionais, os mapas com listas permitem o acesso aos valores por meio de chaves únicas, facilitando a recuperação e manipulação de informações, a exemplo de uma tabela que tem o seu índice e seu elemento.

Índice	Elemento	
	Chave	Informação
1	1	Azul
2	2	Amarelo
3	3	Vermelho
4	4	Verde
5	5	Preto

Tabela 1 | Tabela de elementos.

Na Tabela 1, pode-se observar que a coluna de índice se refere aos índices da lista. Os elementos da lista são formados por uma estrutura de mapa, em que é associada uma chave a uma informação. Para tanto, temos um índice associado a uma chave contendo uma informação.

Os mapas com listas são estruturas baseadas em chaves únicas associadas a valores correspondentes. Cada chave em um mapa está vinculada a um único valor. Para implementar essa estrutura, utiliza-se a ideia de uma lista encadeada, na qual cada nó contém a chave e o valor associado a ela. Isso permite uma busca rápida e eficiente por meio das chaves, mantendo a estrutura flexível para adicionar, remover ou atualizar associações.

Um “esqueleto” de código para a implementação de um mapa se constrói armazenando as associações que pertencem a ele em uma lista; observe um exemplo desse procedimento na Figura 1.

ALGORITMOS E ESTRUTURA DE DADOS

```
MapaLista.java > ...
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class MapaLista {
5      private List<Associacao> associacoes = new ArrayList<Associacao>();
6  public void incluir(String placa, Carro carro) {
7
8      }
9  public void pegarCarro(String placa) {
10
11     }
12  public void excluir(String placa) {
13
14     }
15  public void contemChave(String placa) {
16
17     }
18
19 }
```

Figura 1 | Implementação de um mapa.

A Figura 1 apresenta simplesmente a estrutura da criação de um mapa; portanto as verificações antes de incluir ou excluir uma associação dentro do mapa são necessárias.

Manipulação de associações de chaves em mapas com listas

A manipulação de associações em mapas com listas envolve operações para adicionar, remover, atualizar e recuperar informações por meio das chaves. Para adicionar um novo par chave-valor, é necessário criar um novo nó na lista encadeada, vinculando a chave ao valor correspondente. A atualização de um valor associado a uma chave requer a busca da chave desejada na lista e a substituição do valor atual pelo novo valor. Já a remoção de uma associação envolve a exclusão do nó correspondente à chave desejada. Explicaremos um pouco mais detalhado cada operação a seguir:

- 1. Adição de elementos:** ao adicionar um novo par chave-valor a um mapa, por exemplo, em um sistema de gerenciamento de clientes de uma empresa, cada cliente pode ser identificado por um número único de identificação. Ao adicionar um novo cliente, o sistema atribui um número de identificação e associa esse número aos detalhes do cliente, como nome, endereço e informações de contato.

ALGORITMOS E ESTRUTURA DE DADOS

2. **Atualização de valores:** por exemplo, em um sistema de reserva de voos, a atualização dos detalhes de um voo poderia ser realizada usando mapas com listas. Ao alterar a hora de partida de um voo, o sistema busca o voo desejado pelo seu código e atualiza os detalhes pertinentes, como o horário de partida, usando a chave associada.
3. **Remoção de elementos:** em um sistema de controle de estoque de uma loja, por exemplo, a remoção de itens no estoque pode ser feita usando um mapa com listas. Ao vender um produto, o sistema pode buscar o produto pelo código associado, diminuir a quantidade disponível e, se a quantidade chegar a zero, remover o produto do mapa.
4. **Recuperação de elementos:** por exemplo, em um sistema de gerenciamento de eventos, pode-se utilizar um mapa para associar os participantes a seus respectivos eventos. Ao consultar os participantes de um evento específico, o sistema busca pelo evento desejado usando sua chave associada e recupera os detalhes dos participantes vinculados a ele.

Um exemplo de codificação da operação de adição de elementos em um mapa está exibido na Figura 2. Observe:

```
 adiciona
1  public void adiciona(String placa, Carro carro) {
2      if (!this.contemChave(placa)) {
3          Associacao associacao = new Associacao(placa, carro);
4          this.associacoes.add(associacao);
5      }
6  }
```

Figura 2 | Adição de elemento em um mapa.

Esse código é bem eficiente, pois a inserção no fim de qualquer tipo de lista é muito mais rápida.

Essas operações exemplificam como os mapas com listas são utilizados em contextos profissionais, facilitando a manipulação e gestão de informações específicas por meio de chaves associativas.

Siga em Frente...

Validações de chaves em mapas com listas

Validar chaves em mapas com listas é essencial para garantir a integridade e unicidade das chaves. Antes de inserir uma nova chave, é importante verificar se ela já existe na estrutura para evitar duplicações. Essa validação também é necessária durante a busca por chaves específicas,

ALGORITMOS E ESTRUTURA DE DADOS

garantindo que a chave desejada esteja presente na estrutura antes de realizar operações relacionadas a ela. Detalhando um pouco mais este processo de validação, temos:

1. **Verificação de existência:** por exemplo, em sistemas de gerenciamento de estoque, ao receber um novo produto para adicionar ao inventário, é crucial validar se o código do produto já existe no sistema. Isso evita a duplicação de informações e erros no controle de estoque. Antes de adicionar um novo item, o sistema verifica se o código do produto já está associado a um produto existente.
2. **Garantia de unicidade:** em plataformas de registro de usuários, por exemplo, ao permitir que novos usuários se cadastrem, é essencial garantir que o e-mail de cada usuário seja único. Antes de adicionar um novo usuário ao sistema, a aplicação valida se o e-mail já está associado a outro usuário, mantendo a unicidade, isto é, único valor dos e-mails cadastrados.
3. **Formatos específicos:** em sistemas de validação de dados, como em formulários de cadastro, por exemplo, pode ser necessário validar se os números de telefone estão no formato correto antes de adicioná-los ao mapa. Isso garante que apenas números de telefone válidos sejam armazenados. Assim, ao adicionar um novo contato, o sistema verifica se o número de telefone segue o formato específico esperado.
4. **Controle de integridade:** por exemplo, em sistemas de gerenciamento de transações financeiras, a validação de chaves pode ser crucial para garantir a integridade dos dados. Antes de realizar uma transação, o sistema verifica se a chave associada à transação existe no banco de dados para garantir que a operação seja realizada apenas com informações válidas.

Na sequência, a Figura 3 apresenta um código em Java para verificar a existência de chaves em uma Mapa usando a estrutura *HashMap*.

ALGORITMOS E ESTRUTURA DE DADOS

```

Existe
1  public class VerificacaoExistenciaMapa {
2
3      public static void main(String[] args) {
4          // Criando um mapa para armazenar códigos de produtos e seus respectivos nomes
5          Map<Integer, String> inventario = new HashMap<>();
6
7          // Adicionando alguns produtos ao mapa
8          inventario.put(101, "Produto TesteA");
9          inventario.put(102, "Produto TesteB");
10         inventario.put(103, "Produto TesteC");
11
12         // Verificando se um código de produto já existe no inventário antes de adicionar
13         int novoCodigo = 102;
14         if (inventario.containsKey(novoCodigo)) {
15             System.out.println("O código " + novoCodigo + " já está associado a um produto existente: "
16                         + inventario.get(novoCodigo));
17         } else {
18             inventario.put(novoCodigo, "Novo Produto");
19             System.out.println("Novo produto adicionado ao inventário com o código " + novoCodigo);
20         }
21     }
22 }
23

```

Figura 3 | Verificação de existência de chaves.

Na Figura 3, criamos um mapa “inventario” no qual armazenamos códigos de produtos como chaves e os nomes dos produtos como valores. Ao tentar adicionar um novo produto com um código que já existe no mapa (“novo Código” = 102), verificamos com *containsKey* se essa chave já está presente no mapa, antes de realizar a operação. Se a chave já existe, exibimos uma mensagem informando sobre a existência do código e do produto associado a ele. Caso contrário, adicionamos o novo produto ao mapa.

Esperamos que, com os exemplos da realidade profissional aqui apresentados em conjunto com os códigos e explicações fornecidas, este conteúdo tenha ficado claro para você. Continue seguindo com seus estudos e sempre procurando colocar em prática para poder potencializar o aprendizado.

Vamos Exercitar?

Algoritmos e técnicas de programação – Loja de Eletrônicos

Descrição da situação-problema: vamos retomar nosso exemplo prático apresentado no início da aula! A loja de eletrônicos deseja aprimorar seu sistema de controle de estoque. Eles têm uma variedade de produtos, cada um com um nome, preço e quantidade disponível. O objetivo é criar um algoritmo que utilize um mapa com listas para gerenciar e manipular as informações de estoque de maneira eficiente.

ALGORITMOS E ESTRUTURA DE DADOS

Objetivo: desenvolver um algoritmo utilizando mapas com listas para gerenciar o estoque dos produtos da loja.

Descreva como você aplicaria os algoritmos por meio do conceito de mapas com lista para o desenvolvimento desta aplicação.

Esta resolução trata apenas de forma descritiva os passos para solução da situação-problema, não levando em conta que o estudante, neste momento, deva desenvolver os algoritmos. Diante disso, vamos à resolução.

- **Estrutura de dados:** utilizar um mapa onde as chaves são códigos únicos dos produtos e os valores são listas contendo informações como nome, preço e quantidade disponível.
- **Adicionar produto ao estoque:** ao receber um novo produto, criar uma entrada no mapa com o código como chave e uma lista contendo nome, preço e quantidade disponível como valor.
- **Remover produto do estoque:** localizar o produto pelo código e remover a entrada correspondente no mapa.
- **Atualizar quantidade em estoque:** acessar a lista do produto pelo código e atualizar a quantidade disponível.
- **Buscar informações de um produto:** procurar o produto pelo código no mapa e obter as informações associadas.
- **Listar todos os produtos:** percorrer o mapa e exibir as informações de todos os produtos.
- **Calcular valor total do estoque:** iterar sobre o mapa, multiplicar o preço pelo estoque de cada produto e somar os resultados.
- **Vantagens do uso de mapas com listas:** facilita a busca rápida de produtos pelo código, permite a expansão de informações associadas a cada produto e oferece eficiência no gerenciamento de grandes conjuntos de dados.
- **Exemplo de utilização:** aplicar os algoritmos desenvolvidos em um programa principal, realizando operações como adição, remoção, atualização e consulta de produtos.

O uso de mapas com listas nesse contexto facilita o gerenciamento do estoque, permitindo acessar rapidamente as informações dos produtos e realizar operações de forma eficiente. Esta abordagem é valiosa para empresas que lidam com diferentes tipos de itens e desejam um método organizado e flexível para gerenciar seu estoque.

Saiba mais

Para saber mais sobre pesquisas recentes de algoritmos ligados ao armazenamento associativo, acesse o artigo com o título: "*Uma comparação de métodos de redução de dimensionalidade utilizando índices de preservação da topologia*", de Cláudio J. F. de Medeiros e José Alfredo F. Costa, do Departamento de Engenharia Elétrica – Centro de Tecnologia, da Universidade Federal do Rio Grande do Norte. O trabalho apresenta comparações entre os métodos não supervisionados de redução de dimensionalidade, como análise de componentes principais,

ALGORITMOS E ESTRUTURA DE DADOS

projeção de Sammon, redes auto associativas, mapas auto-organizáveis, Isomap e LLE. Não deixe de conferir para compreender um pouco mais sobre este assunto.

MEDEIROS, C J. F; COSTA, J. A. F. [Uma comparação de métodos de redução de dimensionalidade utilizando índices de preservação da topologia](#). Congresso Brasileiro de Redes Neurais / Inteligência Computacional, Ouro Preto, 2009.

Bons estudos!

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

MEDEIROS, C J. F; COSTA, J. A. F. Uma comparação de métodos de redução de dimensionalidade utilizando índices de preservação da topologia. **Congresso Brasileiro de Redes Neurais / Inteligência Computacional**, Ouro Preto, 2009. Disponível em: https://sbic.org.br/wp-content/uploads/2016/12/174_CBRN2009.pdf. Acesso em: 29 nov. 2023.

SZWARCFITER, J L., MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos: com implementações em Pascal e C**. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 3

Mapas com Espalhamento

Mapas com espalhamento

Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo

ALGORITMOS E ESTRUTURA DE DADOS



para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Partida

Olá, estudante! Esperamos que, a partir do conteúdo desta aula, você possa explorar um mundo de eficiência e organização nos algoritmos!

A estrutura a se trabalhar neste conteúdo é: mapas com espalhamento. Trabalhamos com o conteúdo de espalhamento em outra unidade de ensino. Alguns conteúdos serão resgatados para que possamos prosseguir com os mapas com espalhamento.

Em nossa jornada pelo mundo dos algoritmos, os mapas com espalhamento se destacam como uma peça essencial, oferecendo uma compreensão mais profunda sobre como estruturas de dados podem otimizar processos complexos.

Durante esta aula, serão abordados os conteúdos, como os fundamentos de mapas com espalhamento, a manipulação de associações de chaves e as validações de chaves neles. Você também vai compreender como essa nova estrutura funciona e saberá a melhor maneira pela qual esta estrutura de dados pode auxiliar o profissional de desenvolvimento de sistemas na construção de algoritmos do seu trabalho.

Vamos para um exemplo prático? Imagine o seguinte problema: uma empresa de logística está buscando aprimorar seu sistema de distribuição de produtos para garantir entregas mais rápidas e eficientes aos clientes. Eles lidam com um grande volume de pedidos diários, e a otimização dos roteiros de entrega se tornou uma prioridade para melhorar a satisfação do cliente e reduzir custos operacionais. Você vai gerenciar esse processo. Como você o faria utilizando a estrutura de mapas com espalhamento?

Depois de explorar os princípios desta estrutura de dados e observar exemplos de como esses algoritmos funcionam, você estará pronto para criar soluções que resolvam desafios do mundo profissional utilizando também a estrutura de mapas com espalhamento.

Convidamos você a explorar esta nova estrutura de dados e a entender seu valor no desenvolvimento de algoritmos computacionais.

ALGORITMOS E ESTRUTURA DE DADOS

Vamos Começar!

Nesta aula, você vai compreender uma nova estrutura de dados, os mapas com espalhamento, bem como suas principais características e o funcionamento da manipulação de associações e validações de chaves nessa estrutura.

Mapas com espalhamento, ou *hash maps*, são estruturas de dados que permitem armazenar e recuperar informações de maneira eficiente (Cormen, 2022).

Introdução

Os mapas com espalhamento, também conhecidos como *hash maps* ou *hash tables*, são estruturas de dados fundamentais na ciência da computação, oferecendo uma forma eficiente de armazenar e recuperar informações. Sua utilidade se estende por uma vasta gama de aplicações, desde sistemas de bancos de dados até algoritmos de otimização e gerenciamento de recursos em sistemas computacionais.

Esses mapas funcionam com base no princípio da associação de chaves a valores. Imagine um grande arquivo no qual cada informação tem uma etiqueta única que a identifica. Essa etiqueta é a “chave” e a informação correspondente é o “valor”. O grande diferencial dos mapas com espalhamento está na capacidade de armazenar e recuperar essas informações de maneira extremamente rápida, independentemente do volume de dados.

A facilidade e rapidez com que essas estruturas nos permitem encontrar e manipular dados são essenciais para diversas aplicações práticas. Por exemplo, em sistemas de bancos de dados, eles são utilizados para indexar e recuperar informações rapidamente, minimizando o tempo de busca. Por exemplo, considere um banco de dados de uma loja online. Cada produto possui um código exclusivo. Para otimizar as consultas de pesquisa por produtos, o sistema pode usar um mapa com espalhamento, no qual as chaves seriam os códigos dos produtos e os valores seriam os detalhes desses produtos, como descrição, preço, tamanho, etc. Ao realizar uma busca por um produto com um determinado código, o mapa com espalhamento poderia direcionar rapidamente para os dados desse produto, reduzindo o tempo de busca e aumentando a performance na recuperação das informações.

Já em compiladores, os mapas com espalhamento são usados para armazenar símbolos e palavras-chave, agilizando o processo de compilação. Sua versatilidade se reflete também em sistemas de cache, nos quais são empregados para reter informações frequentemente acessadas, acelerando a recuperação desses dados.

Um exemplo que simula a busca por um produto em um mapa com espalhamento pode ser observado na Figura 1.

ALGORITMOS E ESTRUTURA DE DADOS

```
BuscaProduto
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class BuscaProduto {
5     public static void main(String[] args) {
6         // Criando um mapa com espalhamento para armazenar os produtos
7         Map<Integer, String> produtos = new HashMap<>();
8
9         // Adicionando produtos ao mapa (código do produto, nome do produto)
10        produtos.put(101, "Notebook");
11        produtos.put(202, "Smartphone");
12        produtos.put(303, "Fone de Ouvido");
13
14        // Realizando uma busca por um produto com código 101 (Notebook)
15        int codigoProduto = 101;
16        if (produtos.containsKey(codigoProduto)) {
17            String nomeProduto = produtos.get(codigoProduto);
18            System.out.println("Produto encontrado: " + nomeProduto);
19        } else {
20            System.out.println("Produto não encontrado para o código " + codigoProduto);
21        }
22    }
23 }
```

Figura 1 | Busca de um produto com mapas com espalhamento.

Neste exemplo, criamos um mapa com espalhamento (*HashMap*) que armazena produtos; a chave é o código do produto e o valor é seu nome. Em seguida, realizamos uma busca por um produto com o código 101. Se o produto for encontrado no mapa, ele será exibido; caso contrário, será exibida uma mensagem indicando que o produto não foi encontrado.

Operações com mapas

Um exemplo da busca foi exibido na Figura 1; porém, isso pode também ser realizado de outra maneira. Para verificar se uma chave existe no mapa, pode ser calculado o índice correto da Tabela, o qual pode ser buscado na Lista correspondente. Observe a Figura 2 para entender o procedimento.

ALGORITMOS E ESTRUTURA DE DADOS



```
BuscaPelaChave
1  public boolean contemChave(String placa) {
2
3      int indice = this.calculaIndiceDaTabela(placa);
4      List<Associacao> lista = this.tabela.get(indice);
5
6      for (int i = 0; i < lista.size(); i++) {
7
8          Associacao associacao = lista.get(i);
9
10         if (associacao.getPlaca().equals(placa)) {
11             return true;
12         }
13     }
14
15 }
```

Figura 2 | Verificação pela chave.

A manipulação das associações de chaves é realizada por meio de operações principais: inserção, busca, remoção e atualização de elementos. A inserção envolve a aplicação da função de *hash* à chave e o armazenamento do valor associado no índice correspondente na estrutura de dados. A busca utiliza a mesma função de *hash* para localizar a posição do elemento na tabela, permitindo a recuperação rápida do valor associado à chave. A atualização é similar à inserção, mas substitui o valor existente por um novo valor associado à chave.

A remoção, por sua vez, requer a localização do elemento na tabela por meio da função de *hash* e sua exclusão. Calcula-se o índice e em seguida procuramos a chave na Lista. Encontrada a chave, o próximo passo é remover a associação. Caso a chave não seja encontrada, o que pode ser feito é o lançamento de uma *Exception* (tratamento de erro), ou seja, uma mensagem tratada ao usuário. Vamos abordar um código de remoção na Figura 3.

ALGORITMOS E ESTRUTURA DE DADOS

```
RemovePelaChave
1  public void remove(String placa) {
2
3      int indice = this.calculaIndiceDaTabela(placa);
4      List<Associacao> lista = this.tabela.get(indice);
5
6      for (int i = 0; i < lista.size(); i++) {
7          Associacao associacao = lista.get(i);
8
9          if (associacao.getPlaca().equals(placa)) {
10              lista.remove(i);
11              return;
12          }
13      }
14
15      throw new IllegalArgumentException("A chave não existe");
16
17 }
```

Figura 3 | Remoção pela chave.

Observe que, na linha 15 da Figura 3, é realizado o tratamento de erro por meio de uma mensagem utilizando um “*throw new*”, prática comum para não deixar a execução do programa comprometida.

Siga em Frente...

Prosseguindo, vamos abordar agora o processo de inclusão de uma associação a uma chave.

Quando uma nova associação é adicionada, pode ocorrer que a chave correspondente já exista no Mapa. Nessa situação, é necessário remover a associação anterior antes de inserir a nova. Essa ação é essencial devido à restrição de o Mapa aceitar chaves duplicadas. Isso não é permitido. Veja um exemplo desse código na Figura 4.

ALGORITMOS E ESTRUTURA DE DADOS



AdicionaAssociacao

```
1  public void adiciona(String placa, Carro carro) {  
2  
3      if (this.contemChave(placa)) {  
4          this.remove(placa);  
5      }  
6  
7      int indice = this.calculaIndiceDaTabela(placa);  
8  
9      List<Associacao> lista = this.tabela.get(indice);  
10  
11     lista.add(new Associacao(placa, carro));  
12  
13 }  
14 }
```

Figura 4 | Incluindo uma associação para uma chave.

Portanto, como é observado na Figura 4, o método basicamente adiciona uma nova associação (ou substitui uma existente, se já houver uma chave igual) ao mapa, removendo a associação anterior caso já exista uma chave semelhante para garantir que as chaves sejam únicas no mapa.

Dentro da biblioteca do Java, existem duas maneiras de implementações com mapas que utilizam a técnica de espalhamento. São elas: *HashMap* e *HashTable*.

Um exemplo para a sua utilização pode ser observado na Figura 5.

ALGORITMOS E ESTRUTURA DE DADOS

```
ExemploHashMap
1  public class ExemploHashMap {
2
3      public static void main(String[] args) {
4
5          HashMap<String, Carro> mapa = new HashMap<String, Carro>();
6          mapa.put("ACB8764", new Carro("Exemplo1"));
7          System.out.println(mapa);
8          mapa.put("JHHabc1234", new Carro("Exemplo2"));
9          System.out.println(mapa);
10         mapa.put("JHH9254", new Carro("Exemplo3"));
11         System.out.println(mapa);
12
13         System.out.println(mapa.containsKey("ACB8764"));
14         System.out.println(mapa.get("ACB8764"));
15         mapa.remove("ACB8764");
16
17         System.out.println(mapa);
18
19     }
20 }
```

Figura 5 | Incluindo uma associação para uma chave.

Esse código Java na Figura 5 demonstra a utilização da estrutura *HashMap* para armazenar objetos da classe Carro associados a uma chave do tipo *String*.

Em conclusão, vimos que a utilização adequada dos mapas com espalhamento requer a compreensão desses fundamentos, manipulação eficiente das associações de chaves e implementação correta das validações, o que torna essa estrutura de dados uma ferramenta poderosa para resolver uma variedade de problemas computacionais.

Esperamos que tenha ficado claro para você os conceitos e exemplos da realidade profissional aqui apresentados, bem como a codificação e manipulação de informações por meio dos mapas com espalhamento.

Vamos Exercitar?

Vamos retomar nosso exemplo apresentado inicialmente? Uma empresa de logística está buscando aprimorar seu sistema de distribuição de produtos para garantir entregas mais rápidas

ALGORITMOS E ESTRUTURA DE DADOS

e eficientes aos clientes. Eles lidam com um grande volume de pedidos diários, e a otimização dos roteiros de entrega se tornou uma prioridade para melhorar a satisfação do cliente e reduzir custos operacionais.

Objetivo: desenvolver um sistema que utilize mapas com espalhamento para otimizar os roteiros de entrega, permitindo a rápida identificação das rotas mais eficientes para os entregadores.

Descreva como você aplicaria a solução com mapas com espalhamento para o desenvolvimento desta aplicação.

Esta resolução é uma análise dos requisitos necessários e propõe, de forma descriptiva, uma solução da situação-problema. Diante disso, vamos à resolução.

Na sequência, estão descritos os passos para implementar a solução da situação-problema:

1. **Passo 1:** desenvolvimento de uma estrutura de mapas com espalhamento para armazenar informações sobre rotas e localizações.
2. **Passo 2:** associação de cada endereço de entrega a informações relevantes (como histórico de tráfego, possíveis atrasos etc.) utilizando as chaves e valores no mapa.
3. **Passo 3:** utilização de algoritmos específicos para encontrar as rotas mais rápidas e eficientes com base nas informações armazenadas no mapa.

A aplicação dos mapas com espalhamento neste contexto de logística possibilitou uma gestão mais eficiente dos roteiros de entrega. Isso resultou em entregas mais rápidas, redução de custos operacionais e, consequentemente, em uma melhoria significativa na satisfação do cliente e credibilidade do sistema.

Saiba mais

Para saber mais sobre outras estruturas de dados, como a de “grafos”, acesse o trabalho: *“Abordagem baseada na arquitetura DIKW para construção de grafos de conhecimento semântico que integram dados IoT com fontes de dados heterogêneas”*, de Francisca Jamires da Costa, da Universidade Federal do Ceará (UFC). O trabalho aborda técnicas que viabilizam a construção de um grafo de conhecimento semântico para dados IoT, abrangendo diferentes níveis de dados. Não deixe de conferir para compreender um pouco mais sobre este assunto.

COSTA, F. J. [Abordagem baseada na arquitetura DIKW para construção de grafos de conhecimento semântico que integram dados IoT com fontes de dados heterogêneas](#).

Dissertação (Mestrado em Ciência da Computação). Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Fortaleza, 2023.

Bons estudos!

ALGORITMOS E ESTRUTURA DE DADOS

Referências

CORMEN, T. **Algoritmos – Teoria e Prática.** 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos.** São Paulo: Grupo GEN, 2013.

COSTA, F. J. **Abordagem baseada na arquitetura DIKW para construção de grafos de conhecimento semântico que integram dados IoT com fontes de dados heterogêneas.**

Dissertação (Mestrado em Ciência da Computação). Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Fortaleza, 2023. Disponível em:

https://repositorio.ufc.br/bitstream/riufc/74758/3/2023_dis_fjcosta.pdf. Acesso em: 27 nov. 2023.

SZWARCFITER, J. L. **Estruturas de dados e seus algoritmos.** 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C.** São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

Aula 4

Tratamentos em Armazenamento Associativo

Tratamento em armazenamento associativo



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

ALGORITMOS E ESTRUTURA DE DADOS

Ponto de Partida

Olá, estudante! Desejamos que esta aula seja enriquecedora, já que os assuntos abordados têm grande importância para o seu domínio em algoritmos e estruturas de dados.

Durante esta aula, vamos dar continuidade ao conteúdo de armazenamento associativo e abordar alguns detalhes com o intuito de aprofundar nosso aprendizado. Como já vimos, o armazenamento associativo também pode oferecer inúmeras vantagens para o processo correto do desenvolvimento de algoritmos.

Faremos uma introdução sobre o tema, com foco em tratamentos de armazenamento associativo, seus principais conceitos teóricos e tipos, incluindo o tratamento de colisões, buscas eficientes e armazenamento associativo multinível. Você também vai compreender como essas técnicas funcionam dentro dos algoritmos com exemplos implementados via código, além de poder observar de que maneira esta estrutura de dado pode auxiliar o profissional da área técnica de sistemas a criar algoritmos robustos com base nesses conceitos.

Vamos para um exemplo prático? Imagine o seguinte problema: em um centro de distribuição de produtos eletrônicos, a gestão de estoque é crucial para garantir a eficiência na entrega aos clientes. Para otimizar esse processo, um sistema de armazenamento multinível está sendo implementado. Nesse sistema, cada produto eletrônico é associado a uma chave única baseada no seu código de barras e é armazenado em diferentes níveis de estoque. O seu trabalho será desenvolver um método para gerenciar o estoque multinível, permitindo a rápida identificação da localização de um produto específico no armazém.

Após compreender os fundamentos dessa estrutura de dados e analisar casos práticos que demonstram o funcionamento desses algoritmos, você estará preparado para conceber soluções destinadas a resolver problemas do ambiente profissional, utilizando o armazenamento associativo e mapas correlacionais.

Encorajamos você a explorar esse recurso de organização de dados e a compreender sua relevância no desenvolvimento de algoritmos computacionais. Bons estudos!

Vamos Começar!

Nesta aula, vamos dar continuidade ao nosso estudo, abordando, desta vez, algumas particularidades dentro do armazenamento associativo. Vamos abordar colisões em mapas e o armazenamento associativo multinível, e continuaremos a tratar da manipulação de elementos nesta estrutura. Iniciemos o conteúdo.

O desenvolvimento das técnicas de tabelas de dispersão foi motivado por um caso bastante simples, porém importante (Szwarcfiter, 2020).

ALGORITMOS E ESTRUTURA DE DADOS

Se o número de chaves (n) corresponde exatamente ao número de compartimentos (m) e os valores das chaves variam de 0 a m - 1, é possível utilizar diretamente o valor de cada chave como seu índice na tabela. Nesse método, cada chave x é armazenada no compartimento x. Essa abordagem é amplamente utilizada na programação comercial, conhecida como acesso direto. A Figura 1 apresenta um exemplo visual desse método; os nós consistem apenas de seus valores-chave e as setas representam os índices das chaves (Szwarcfiter, 2020).

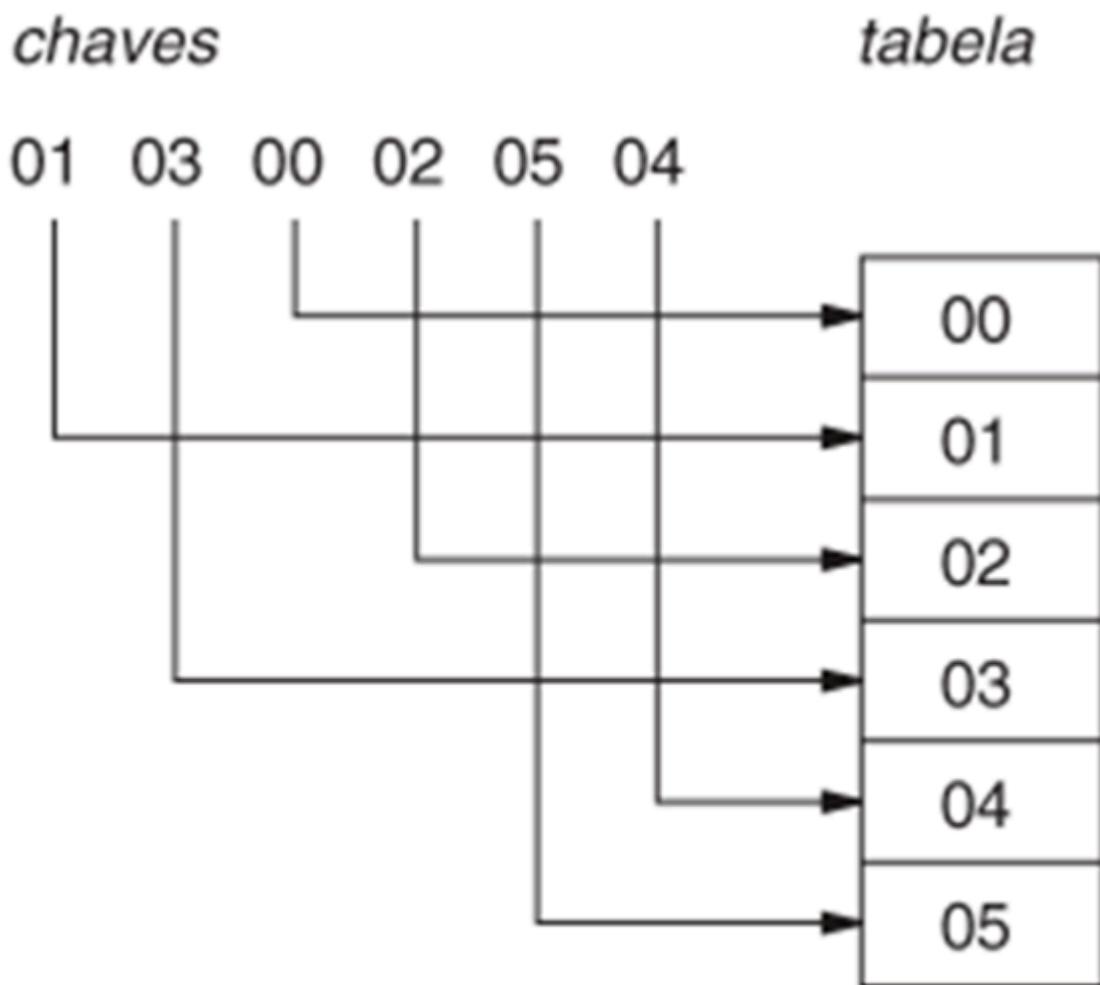


Figura 1 | Acesso direto. Fonte: Szwarcfiter (2022, p. 233).

Desta forma, as tabelas de dispersão ou espalhamento podem ser utilizadas em conjunto com o armazenamento associativo no desenvolvimento de soluções mais ágeis e precisas, dependendo da necessidade.

ALGORITMOS E ESTRUTURA DE DADOS

Tratamentos em armazenamento associativo

O armazenamento associativo oferece diversas possibilidades de tratamento de dados que são fundamentais para a organização e manipulação eficiente das informações. Por meio desse modelo, é possível realizar diferentes operações, como a associação de chaves a valores únicos, a busca de dados com excelente performance, a remoção de associações, verificação da existência de uma chave e gerenciamento do número de associações na estrutura (Szwarcfiter, 2020).

Esses tratamentos possibilitam uma gestão dinâmica e precisa dos dados armazenados, permitindo a criação de algoritmos robustos e eficazes para resolver uma ampla gama de problemas.

Colisões e técnicas de resolução em mapas

O tratamento de colisões é um aspecto essencial em estruturas de armazenamento associativo, como os mapas com espalhamento. Colisões ocorrem quando duas ou mais chaves diferentes são associadas à mesma posição de armazenamento após a aplicação da função de espalhamento.

Para resolver colisões, diversas técnicas são empregadas:

1. Encadeamento separado: cada posição da tabela de espalhamento é uma lista encadeada, à qual múltiplas chaves podem ser associadas. Quando uma colisão ocorre, a chave é inserida na lista correspondente à posição da tabela. Um exemplo da codificação desta técnica é exibido na Figura 2.

ALGORITMOS E ESTRUTURA DE DADOS

```
1 import java.util.*;  
2  
3 public class EncadeamentoSeparado {  
4     public static void main(String[] args) {  
5         // Tamanho da tabela de espalhamento  
6         int tamanhoTabela = 5;  
7  
8         // Criação da tabela de espalhamento com listas encadeadas  
9         List<String>[] tabelaEspalhamento = new LinkedList[tamanhoTabela];  
10  
11         // Inicialização das listas encadeadas em cada posição da tabela  
12         for (int i = 0; i < tamanhoTabela; i++) {  
13             tabelaEspalhamento[i] = new LinkedList<>();  
14         }  
15  
16         // Função de espalhamento simples para exemplificar  
17         // Neste exemplo, a chave é convertida em um índice  
18         String[] chaves = { "c1", "c2", "c3", "c4", "c5" };  
19  
20         for (String chave : chaves) {  
21             int indice = Math.abs(chave.hashCode()) % tamanhoTabela;  
22             tabelaEspalhamento[indice].add(chave);  
23         }  
24  
25         // Exibição da tabela de espalhamento  
26         for (int i = 0; i < tamanhoTabela; i++) {  
27             System.out.println("Posição " + i + ": " + tabelaEspalhamento[i]);  
28         }  
29     }  
30 }
```

Figura 2 | Código do encadeamento separado.

Neste exemplo, criamos uma tabela de espalhamento com encadeamento separado utilizando listas encadeadas para resolver colisões. As chaves são mapeadas para índices na tabela usando uma função simples (nesse caso, `hashCode()`). Cada chave é adicionada à lista encadeada correspondente à sua posição na tabela. Este é um exemplo básico para ilustrar o funcionamento do encadeamento separado em Java.

2. Endereçamento aberto: nesse método, quando há uma colisão, a busca é feita por outra posição na tabela de espalhamento, baseando-se em um algoritmo de sondagem, como *linear probing* (tentativa de encontrar a próxima posição vazia) ou *quadratic probing* (tentativa de encontrar a próxima posição vazia usando uma função quadrática). Um exemplo da codificação desta técnica é exibido na Figura 3.

ALGORITMOS E ESTRUTURA DE DADOS

```

EnderecamentoAberto
1  public class EncadeamentoAberto {
2      static final int TAMANHO_TABELA = 10;
3      static String[] tabelaEspalhamento = new String[TAMANHO_TABELA];
4
5      // Método de hash simples para este exemplo
6      static int hash(String chave) {
7          return Math.abs(chave.hashCode()) % TAMANHO_TABELA;
8      }
9
10     // Função para inserir elementos usando linear probing
11     static void inserir(String chave) {
12         int indice = hash(chave);
13         while (tabelaEspalhamento[indice] != null) {
14             indice = (indice + 1) % TAMANHO_TABELA;
15         }
16         tabelaEspalhamento[indice] = chave;
17     }
18
19     public static void main(String[] args) {
20         inserir("c1");
21         inserir("c2");
22         inserir("c3");
23         inserir("c4");
24
25         // Exibir a tabela de espalhamento
26         for (int i = 0; i < TAMANHO_TABELA; i++) {
27             System.out.println("Posição " + i + ": " + tabelaEspalhamento[i]);
28         }
29     }
30 }

```

Figura 3 | Código do endereçamento aberto.

Neste exemplo, a técnica de encadeamento aberto usando *linear probing* é empregada para inserir elementos na tabela de espalhamento. A função *hash()* calcula o índice da tabela no qual a chave deve ser inserida. Se a posição estiver ocupada, o código usa o método de *linear probing*, incrementando o índice até encontrar uma posição vazia para inserir a chave. Este é outro exemplo agora para ilustrar o encadeamento aberto em Java.

3. Re-hash: nesse método, se uma colisão ocorre, a função de espalhamento é aplicada novamente à chave, resultando em uma nova posição na tabela de espalhamento. Esse processo é repetido até encontrar uma posição sem colisão. Um exemplo da codificação desta técnica é exibido nas Figuras 4 e 5.

ALGORITMOS E ESTRUTURA DE DADOS

```

Rehash
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class ReHashExample {
5
6     static class TabelaEspalhamento {
7         private Map<Integer, String> tabela = new HashMap<>();
8
9         // Função de hash simples
10        private int hash(String chave) {
11            return Math.abs(chave.hashCode());
12        }
13
14        // Função para inserir elementos com tratamento de colisões usando re-hash
15        public void inserir(String chave) {
16            int hash = hash(chave);
17            while (tabela.containsKey(hash)) {
18                hash++; // Re-hash simples incrementando o valor da chave
19            }
20            tabela.put(hash, chave);
21        }
22
23        // Função para visualizar a tabela de espalhamento
24        public void visualizarTabela() {
25            for (Map.Entry<Integer, String> entry : tabela.entrySet()) {
26                System.out.println("Posição " + entry.getKey() + ":" + entry.getValue());
27            }
28        }
29    }
}

```

Figura 4 | Código do Re-hash – parte 1.

A Figura 5 será utilizada para exibir o método *main()* da classe mostrada na Figura 4, ou seja, ainda se trata da mesma classe.

```

Rehash
31     public static void main(String[] args) {
32         TabelaEspalhamento tabela = new TabelaEspalhamento();
33         tabela.inserir("chave1");
34         tabela.inserir("chave2");
35         tabela.inserir("chave3");
36         tabela.inserir("chave4");
37
38         tabela.visualizarTabela();
39     }
40 }

```

ALGORITMOS E ESTRUTURA DE DADOS

Figura 5 | Código do Re-hash – parte 2.

Este exemplo utiliza uma estrutura de mapa (*HashMap*) para simular uma tabela de espalhamento. A função inserir usa a técnica de *Re-hash* para lidar com colisões: se a chave já existe na tabela, o código incrementa o valor do *hash* até encontrar uma posição vazia, e então insere a chave nessa posição. Esse é um exemplo de implementação simplificado para demonstrar o conceito de *Re-hash* em Java.

A escolha da técnica de tratamento de colisões depende do contexto de aplicação, do tamanho da estrutura de dados e da eficiência desejada para operações de busca e inserção. Cada método tem suas vantagens e limitações, e a seleção adequada pode otimizar a eficiência do armazenamento associativo.

Siga em Frente...

Armazenamento associativo multinível

O armazenamento multinível é uma estrutura avançada que organiza dados em diferentes camadas, proporcionando velocidades de acesso variáveis e eficiência na busca de informações. Composto por múltiplos níveis, cada um com características específicas de acesso e capacidade, essa estrutura oferece uma solução otimizada para o gerenciamento de dados em sistemas computacionais.

O armazenamento multinível é uma arquitetura de armazenamento de dados em que cada nível contém tempos de acesso e capacidades distintas. Geralmente compreende níveis de cache, memória principal e armazenamento em disco, organizados de acordo com a frequência e urgência de acesso aos dados.

Suas principais características são:

- **Hierarquia de níveis:** divisão dos dados em camadas, permitindo acesso mais rápido nos níveis superiores e armazenamento mais extenso nos inferiores.
- **Otimização de performance:** dados frequentemente acessados são mantidos em níveis de acesso mais rápido, reduzindo o tempo de busca.
- **Eficiência de armazenamento:** alocação inteligente de dados, mantendo informações críticas em locais de acesso rápido e relegando informações menos utilizadas para níveis de acesso mais lento.

Imagine uma loja virtual de calçados, na qual o armazenamento multinível é implementado para gerenciar informações sobre os modelos disponíveis. Os calçados de alta rotatividade e maior demanda, como tênis esportivos ou modelos da moda, são mantidos nas camadas superiores

ALGORITMOS E ESTRUTURA DE DADOS

para acesso imediato. Enquanto isso, sapatos de estilos menos populares ou estoques extras são armazenados em camadas de acesso mais lento.

A solução envolve diretamente a configuração do armazenamento multinível, por meio do qual os calçados mais populares e frequentemente comprados pelos clientes são posicionados nas camadas superiores, garantindo um acesso ágil e eficiente para a gestão do estoque.

Em conclusão, podemos afirmar que o armazenamento multinível é fundamental em aplicações que requerem alta performance e eficiência na busca e acesso aos dados, sendo uma ferramenta essencial para otimização de operações em sistemas computacionais.

Por fim, esperamos que os assuntos abordados nesta aula tenham ficado claros para você. A partir de agora, é com você. Persista em seus estudos. Cada nova descoberta expande seu horizonte de conhecimento. Continue explorando e desvendando os fascinantes segredos da área de algoritmos e estruturas de dados. Bons estudos!

Vamos Exercitar?

Vamos retomar o nosso exemplo apresentado inicialmente! Em um grande armazém de produtos eletrônicos, um código de barras de um produto é associado a uma localização específica dentro do depósito. Esse armazém possui diversos níveis, prateleiras e setores. O objetivo é criar um sistema que permita aos funcionários localizar rapidamente qualquer produto apenas pelo código de barras, indicando exatamente em qual parte do armazém ele está localizado.

Objetivo: desenvolver um método para gerenciar o estoque multinível, permitindo a rápida identificação da localização de um produto específico no armazém.

Descreva como você aplicaria os algoritmos de armazenamento associativo multinível para o desenvolvimento desta aplicação.

Na sequência, estão descritos os passos para implementar a solução da situação-problema:

1. **Identificação da chave única:** o código de barras de cada produto será a chave única para a localização no armazém.
2. **Estrutura do mapa multinível:** crie uma estrutura de dados que represente o armazém, dividindo-o em diferentes níveis, corredores e prateleiras. Use uma estrutura hierárquica, como um mapa de mapas ou uma árvore, para armazenar essas informações.
3. **Associação chave-valor:** associe cada código de barras (chave) à localização do produto no armazém (valor). Por exemplo, o código de barras pode ser associado a um nível, corredor e prateleira específicos.
4. **Implementação do método de busca:** crie um método que, ao receber um código de barras como entrada, retorne a localização exata do produto no armazém, consultando a estrutura

ALGORITMOS E ESTRUTURA DE DADOS

de armazenamento multinível.

5. **Testes e validação:** realize testes extensivos para garantir que o sistema identifique corretamente a localização de cada produto com base no código de barras fornecido.
6. **Aprimoramento contínuo:** após a implementação inicial, avalie o desempenho do sistema, faça ajustes e melhorias conforme necessário para garantir a precisão e eficiência na localização dos produtos.

O armazenamento multinível pode ser aplicado em cenários complexos de logística e distribuição. A eficiência na gestão de estoque proporcionada por essa abordagem ajuda a minimizar erros e agilizar a localização de produtos, otimizando as operações e garantindo um serviço mais rápido e preciso aos clientes.

Saiba mais

Para saber mais sobre várias estruturas de dados em algoritmos, acesse o livro *Estruturas de dados e seus algoritmos*, do autor Jayme Luiz Szwarcfiter. O livro aborda assuntos como tabelas de dispersão ou espalhamento, estruturas autoajustáveis, listas de prioridades avançadas e tantas outras. Você pode ter este livro como um guia de consulta para aprender uma estrutura quando surgir a necessidade. Não deixe de conferir para compreender um pouco mais sobre este assunto.

SZWARCFITER, Jayme Luiz. [**Estruturas de dados e seus algoritmos**](#). 3^a ed. Rio de Janeiro: LTC, 2020.

Bons estudos!

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

SZWARCFITER, J. L. [**Estruturas de dados e seus algoritmos**](#). 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C**. São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.

ALGORITMOS E ESTRUTURA DE DADOS

Aula 5

Encerramento da Unidade

Videoaula de Encerramento



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Estudante, esta videoaula foi preparada especialmente para você. Nela, você irá aprender conteúdos importantes para a sua formação profissional. Vamos assisti-la?

Ponto de Chegada

Olá, estudante! Para desenvolver a competência desta Unidade, citada a seguir: “compreender os fundamentos de armazenamento associativo, e avaliar seu funcionamento e o porquê da sua utilização em sistemas de computação”, você deverá primeiramente conhecer os conceitos fundamentais abordados ao longo dela.

Um nível considerável de habilidade envolve a aptidão para compreender e reconhecer os contextos nos quais empregamos as técnicas envolvidas com mapas associativos em algoritmos e estruturas de dados. No decorrer da Unidade, investigamos sua definição, seus elementos constituintes, o modo como os mapas associativos são executados, estratégias para otimizar sua eficácia e aplicações em cenários reais de sistemas.

Mapas associativos são determinantes em algoritmos devido à sua capacidade de associar chaves a valores, permitindo acesso eficiente aos dados. Eles facilitam a busca, inserção e remoção de elementos, otimizando o desempenho em operações-chave. Além disso, são fundamentais para resolver problemas que exigem mapeamento rápido e flexível de informações.

ALGORITMOS E ESTRUTURA DE DADOS

De acordo com a evolução nos conteúdo da disciplina de algoritmos e estrutura de dados, podemos observar cenários da realidade profissional nos quais os mapas associativos podem ser aplicados. Você também adquiriu a habilidade de examinar a aplicação dessas estruturas em diferentes contextos.

Desenvolver tais habilidades representa o ponto de partida para a construção de estruturas de dados robustas, eficientes, e acima de tudo preparando-o para enfrentar novos desafios no desenvolvimento de sistemas. Ao aperfeiçoar sua capacidade de tomar decisões sobre quando e como implementar mapas associativos, você estará bem equipado para avançar em sua carreira tecnológica.

É Hora de Praticar!



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Convidamos você a colocar em prática o conhecimento adquirido nesta Unidade de ensino. Para isso, vamos abordar um Estudo de caso com o cenário descrito a seguir: **Melhoria na busca de contatos em um sistema de gerenciamento de clientes.**

ALGORITMOS E ESTRUTURA DE DADOS



Figura 1 | Busca de informações. Fonte: Shutterstock.

- **Cenário:** imagine um sistema de gerenciamento de clientes utilizado por uma empresa de vendas. A empresa possui um sistema atual, porém ele enfrenta desafios na busca eficiente de informações sobre os clientes. Portanto, a proposta é aprimorar a funcionalidade de busca utilizando algoritmos de mapas associativos.
- **Objetivo:** implementar mapas associativos para otimizar a busca por clientes no sistema de gerenciamento, proporcionando um acesso mais rápido e eficiente às informações do cliente.

Passos para resolução:

1. Para o início da resolução deste Estudo de caso, é necessário:
 1. Examinar a estrutura atual de armazenamento de dados dos clientes no sistema de gerenciamento.
 2. Identificar os gargalos na busca de informações.
2. Implementação de mapas associativos em C:
 1. Declare e inicialize um mapa associativo (*hash map*) para armazenar os dados dos clientes.
 2. Utilize a função *hash* para associar cada cliente a uma chave única no mapa.

ALGORITMOS E ESTRUTURA DE DADOS

3. Testes e verificação:

1. Insira dados de clientes no mapa.
2. Realize testes de busca para verificar a eficácia da implementação.

Crie sua solução algorítmica, de preferência na linguagem de programação C. Bom trabalho!

- Em que situações específicas os mapas associativos são mais vantajosos em comparação com outras estruturas de dados, e como essa escolha impacta o desempenho do algoritmo?
- Considerando a capacidade de os mapas associativos associar chaves a valores, como essa característica pode ser explorada para resolver problemas práticos e complexos no desenvolvimento de software?

Tente exercitar-se, empenhando-se em refletir sobre a responder a estas questões.

Sugestão de solução

O sistema agora utiliza um mapa associativo para buscar informações de clientes de forma mais eficiente. As funções de inserção e busca garantem um acesso rápido aos dados do cliente.

Na sequência, o código da proposta:

ALGORITMOS E ESTRUTURA DE DADOS

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_CLIENTS 1000

typedef struct {
    char nome[50];
    int idade;
    // Outros dados do cliente...
} Cliente;

typedef struct {
    Cliente* clientes[MAX_CLIENTS];
} MapaAssociativo;

// Função hash simples para exemplo
unsigned int hash(char* chave) {
    unsigned int hash = 0;
    for (int i = 0; i < strlen(chave); i++) {
        hash = hash * 31 + chave[i];
    }
    return hash % MAX_CLIENTS;
}

// Função para inserir cliente no mapa
void inserirCliente(MapaAssociativo* mapa, char* chave, Cliente* cliente) {
    int index = hash(chave);
    mapa->clientes[index] = cliente;
}

// Função para buscar cliente no mapa
Cliente* buscarCliente(MapaAssociativo* mapa, char* chave) {
    int index = hash(chave);
    return mapa->clientes[index];
}
```

Figura 2 | Código da solução em C.

Não esqueça que, para testar o algoritmo, você vai precisar da função *main()*, criando algumas *structs* no código.

ALGORITMOS E ESTRUTURA DE DADOS

Ao completar este Estudo de caso, o aluno terá adquirido conhecimentos práticos na implementação de mapas associativos para resolver desafios reais em sistemas de gerenciamento. A compreensão do uso dessas estruturas de dados, juntamente com a prática de implementação, oferecerá ao aluno habilidades valiosas na melhoria da eficiência de busca em aplicações práticas, destacando a importância dessas habilidades no desenvolvimento de software.

No infográfico a seguir, são apresentadas, resumidamente, algumas características em relação ao aprendizado dos mapas associativos, bem como conceitos, fundamentos, algumas vantagens e aplicações em que eles são utilizados.

Convidamos você a refletir sobre os conceitos apresentados neste infográfico.

ALGORITMOS E ESTRUTURA DE DADOS

MAPAS ASSOCIATIVOS **FUNDAMENTOS**

1

CONCEITOS

Armazena elementos na forma de pares chave-valor. Conhecido também como dicionário. Associa cada elemento a uma chave única.

2

FUNDAMENTOS

Unidades de chaves, eficiência nas operações, flexibilidade de dados, funções hash, agilidade na recuperação de dados

3

VANTAGENS

Implementação simples, diversas aplicações, adaptação a dados dinâmicos, resolução de problemas, busca eficiente

4

APLICAÇÕES

Manipulação de gráficos, processamento de linguagem natural, roteamento em redes, índices de banco de dados, banco de dados NoSQL

CORMEN, T. **Algoritmos – Teoria e Prática**. 3^a ed. Rio de Janeiro: LTC, 2022.
CORMEN, T. **Desmistificando Algoritmos**. São Paulo: Grupo GEN, 2013.

ALGORITMOS E ESTRUTURA DE DADOS

SZWARCFITER, J. L. **Estruturas de dados e seus algoritmos.** 3^a ed. Rio de Janeiro: LTC, 2020.

TENENBAUM, A. M. **Estrutura de dados usando em C.** São Paulo: Makron Books, 1995.

ZIVIANI, N. **Projeto de algoritmos:** com implementações em Pascal e C. 3^a ed. São Paulo: Cengage Learning, 2011.