

Unidade 3

Tipos Abstratos de Dados e Filas Prioritárias



Introdução

Olá Aluno (a), nessa unidade de ensino serão abordadas novas estruturas de dados para continuarmos nosso aprendizado. A disciplina de Análise e Projeto de Algoritmos, objetiva trazer diferentes conceitos e formas de desenvolvimento, sempre propondo soluções inteligentes.

Os temas das duas aulas dessa unidade de ensino serão: Tipos Abstratos de Dados ou simplesmente TAD e Filas Prioritárias. Serão apresentados os conceitos desses dois conteúdos, as suas principais características e, acima de tudo, como ele é utilizado dentro de um Algoritmo Computacional.

Em nosso estudo, também será construído exemplos em vários contextos que envolvam os TAD e as Filas Prioritárias. Para um desses contextos, vamos utilizar classes com métodos, de acordo com o paradigma orientado a objetos, que demonstre a utilização de diversos tipos abstratos de dados. Os algoritmos de Filas Prioritárias também serão exemplificados utilizando cenários reais, com usabilidade dessa técnica de resolução de problemas.

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

Objetivos

- Conhecer as ferramentas para análise e projeto de algoritmos;
- Conhecer e identificar os Tipos Abstratos de Dados;
- Desenvolver algoritmos utilizando os Tipos Abstratos de Dados;
- Conhecendo e compreendendo os algoritmos de Filas Prioritárias.

Conteúdo programático

Aula 01 – Tipos Abstratos de Dados - TAD.

Aula 02 – Filas Prioritárias.



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

Pode ser necessário instalar um aplicativo de leitura QRcode no celular

Aula 01 – Tipos Abstratos de Dados - TAD

Introdução

O objetivo dessa aula é abordar os Tipos Abstratos de Dados (TAD), trazendo o conceito e explicações práticas de como utilizar a lógica do TAD dentro do algoritmo desenvolvido. As explicações serão demonstradas utilizando exemplos didáticos para facilitar a absorção do conteúdo.

A base desse tipo de algoritmo é reduzir a informação necessária para a criação do método principal do programa. Vale lembrar que, geralmente, uma linguagem de programação possui um arquivo ou classe principal que é responsável pela chamada das outras sub-rotinas. A linguagem de programação Java é um exemplo entre o paradigma orientado a objeto que trabalha dessa forma.

Vamos a um exemplo prático de um cliente. Sem a utilização de uma estrutura algorítmica TAD, o cliente seria controlado dentro do algoritmo, por variáveis soltas como por exemplo (id, nome, cidade e uf). Essas variáveis seriam operadas de forma isolada e sem uma associação por um tipo ou classe qualquer.

Conceitualmente, utilizando a lógica de um tipo abstrato de dado, o programa passou a ser projetado conjuntamente, não sendo um id, nome, cidade e uf, porém simplesmente o tipo **cliente**. Dentro desse tipo, são incluídos todos os atributos que se referem ao cliente. Esse tipo cliente, é definido dentro de uma classe onde cada atributo terá um tipo específico como se fossem variáveis comuns.

Como já estudado na disciplina de Programação Orientada a Objetos, uma classe é composta pelo seu nome, seus atributos e suas operações, logo, o tipo cliente também terá operações que serão criadas dentro da classe.

Para maiores detalhes, o Quadro 1 a seguir, apresenta um comparativo entre as formas de abordagem do algoritmo. Na coluna 1, os dados serão controlados por variáveis isoladas, já na coluna 2, um tipo abstrato de dados é utilizado para agrupar as informações dentro do tipo **cliente**.

Quadro 1 – Comparativo entre as formas de abordagem

| Número | Tipo comum | Tipo Cliente |
|--------|---|--|
| 1 | Id | cliente: {int id} |
| 2 | Nome | nome: {String nome} |
| 3 | E-mail | email: { String email} |
| 4 | Endereço | endereço: { 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 |

Fonte: elaborada pelo autor (2021)

A utilização de modelos por meio do TAD, permitiu uma melhor compreensão dos algoritmos, porém acabou aumentando a complexidade das soluções, o que tornou imprescindível a construção da modelagem dos dados para qualquer tipo de sistema computacional.

Essa modelagem de dados nos remete ao paradigma da orientação a objetos. Portanto, a Programação Orientada a Objetos ou ainda, *Objetct Oriented Programming* (OOP), é a responsável pela evolução dos tipos abstratos de dados, fazendo com que esses algoritmos possam ser construídos sendo controlados de acordo com a política de desenvolvimento e implementação de classes e demais objetos desse paradigma.

A ideia dos tipos abstratos de dados é que realmente, os tipos primitivos ou os tipos simples para representar e manipular tipos de dados mais complexos não são suficientes. Sendo assim, é necessário trabalhar com novos tipos de dados que sejam capazes de suprir as exigências algorítmicas desses novos modelos de desenvolvimento de sistemas computacionais (CORMEN, 2012).

Indicação de Leitura

A obra Teoria e Prática de Cormen, apresenta uma boa base dos diferentes tipos de estruturas existentes no cenário de Algoritmos. Alguns outros conteúdos como: ordenação de vetores, como, por exemplo: (*quick sort*, *merge sort* e ordenação por *heap*), análise de algoritmos, recursividade, filas de prioridade, estrutura de dados elementares, como: (pilhas e filas) e muitas outras estruturas, são abordadas neste livro e trazem características e exemplos de aplicação desses conteúdos com uma boa didática para o seu aprendizado.

Disponível em: <https://docero.com.br/doc/nxxnx8v>. Acesso em: 9 jul. 2021.

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

Conceito de um TAD

Antes de começar a parte prática propriamente dita, vamos contextualizar o cenário de aplicação para este tipo abstrato de dado.

Sem dúvida, a automação residencial é uma área que cativa os usuários desse setor, que avança em produtos e serviços oferecidos. A automação de alguns eletrodomésticos ou mesmo luzes e portões eletrônicos, estão presentes hoje na residência de muitas pessoas de todas as partes do mundo. A comodidade de ligar ou desligar um aparelho remotamente por um aplicativo, ou ainda, por comandos de voz por meio de um assistente como a Alexa, atrai muito a atenção de usuários que apreciam esse tipo de tecnologia. No meio de tantas utilidades como conforto e entretenimento, algumas automações também se tornam fundamentais quando o assunto é a segurança. Por exemplo, existem sistemas de automação residencial que são programados para evitar acidentes causados pelo vazamento de gás. Os moradores podem até não estar em casa, porém havendo um vazamento de gás, um acidente gravíssimo pode acontecer. Portanto, o sistema se encarrega de cortar o abastecimento, abrir as janelas para ventilação dos ambientes e ainda avisar o proprietário por notificação em seu smartphone. Ainda assim, se pensarmos em termos de alimentação das informações, podemos construir um sistema computacional, que vai armazenando o histórico dos acontecimentos dentro da residência. Com essas

informações no banco de dados, é possível criar relatórios com indicadores, gráficos apresentando o percentual de irregularidades por dispositivos entre tantas outras possibilidades.

Para melhor entendimento da automação residencial ou a automação em si, assista ao vídeo abaixo que indicamos a você.

Indicação de Vídeo

Veja este vídeo do canal “*QUE CRIATIVO*”. O canal exhibe conteúdos sobre Tecnologia e novas tendências desse mercado. No vídeo é possível observar vários dispositivos como câmeras, sensores, robôs domésticos etc.

Você pode assistir o vídeo quantas vezes achar necessário para entender o trabalho dos elementos envolvidos nas automações.

Disponível em: <https://www.youtube.com/watch?v=MaY6DEHF4rY&t=408s>. Acesso em: 30 jun. 2021.

Exemplo prático de um TAD

Após a apresentação de todo o cenário da automação residencial e o que pode ser desenvolvido por meio dela, vamos abordar um exemplo prático para automatização de alguns itens dentro de uma residência.

Vamos começar por um eletrodoméstico muito utilizado todos os dias, a Televisão, ou simplesmente TV. Os comandos existentes em um controle remoto como mudança de canal, alteração no volume do aparelho ou a escolha de execução de um aplicativo específico como um *streaming*, podem ser automatizados via sistemas de aplicativos. Portanto, vamos começar pontuando alguns atributos desse aparelho que iremos definir em nosso exemplo. O quadro 2 exhibe esses atributos.

É importante lembrar que vamos abordar o paradigma orientado a objetos para toda a construção da lógica e codificação dos exemplos.

Quadro 2 – Atributos controlados

| Número | Tipo de Dado | Nome do Atributo |
|--------|--------------|------------------|
| 1 | string | marca |
| 2 | boolean | ligado |
| 3 | volume | int |
| 4 | canal | int |
| 5 | string | local |

Fonte: elaborada pelo autor (2021)

O quadro 2, exibe os atributos da classe da TV que será implementada. Essa classe terá um construtor e alguns métodos para controle do volume e do canal selecionado.

A classe criada com os atributos e construtor definidos é apresentada na Figura 1.

Figura 1 – Classe Televisao.java

A screenshot of a code editor with two tabs: 'Televisao.java' and 'UtilizarTV.java'. The 'Televisao.java' tab is active, showing the following Java code:

```
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
11         ligado = false;
12
13     }
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

Essa classe tem o objetivo de fazer todo o gerenciamento das operações que serão executadas pelo dispositivo eletrônico, em nosso exemplo, a TV.

Na primeira videoaula, será explicada a criação do local do projeto, bem como, a criação da classe “Televisao.java”, para posteriormente serem construídos e implementados os métodos que irão controlar a TV.

Videoaula 1

Acompanhe a videoaula, ela vai apresentar a criação do diretório da localização dos arquivos e a criação da classe “Televisao.java” com seus atributos e o seu construtor. O professor vai explicar o código na prática e passo a passo para demonstrar todo o processo.

Assista e estude esse vídeo para compreender a ideia do programa e a codificação passo a passo do exemplo a ser criado.



Videoaula 1

Utilize o QRcode para assistir!

Acompanhe a videoaula, ela vai apresentar a criação do diretório da localização dos arquivos e a criação da classe “Televisao.java” com seus atributos e o seu construtor.



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

Os métodos que serão controlados dentro do programa são: troca de canal e o aumento do volume da TV. O método da troca de canal será denominado como: **defineCanal()** e vai pedir o canal por parâmetro. Por meio desse valor, o programa vai testar se a TV está ligada e então troca o valor do canal para o valor passado por parâmetro.

A Figura 2, apresenta a criação da função `defineCanal()`:

Figura 2 – Função que define o canal da TV

```
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 }
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

O atributo *ligado* do tipo *boolean*, recebe o valor **true** quando a TV está ligada, sendo assim, ele verifica antes de “setar” o canal para o atributo.

O método do aumento do volume será denominado como: **aumentarVolume()**. Ele não tem parâmetro e é do tipo *void*. A verificação se a TV está ligada é feita da mesma maneira que o método *defineCanal()*. Após essa verificação, o método aumenta o volume da TV com incremento de passo 1. Isso significa que a cada vez que o comando é executado a TV aumenta em 1(um) o volume atual do aparelho. Se estava em 15, após a execução do método *aumentarVolume()*, o volume passa a ser 16.

A Figura 3, apresenta a criação da função *aumentarVolume()*

Figura 3 – Função que aumenta o volume da TV

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

Fonte: elaborado pelo autor via IDE Sublime (2021)

A construção de outros métodos também pode ser implementada se assim você deseja, como, por exemplo, outra função para diminuir o volume. Vamos demonstrar a criação dos dois métodos relatados na prática com o professor.

Videoaula 2

A próxima videoaula, vai abordar a construção do método para definir o canal da TV e do outro método para aumentar o volume da TV. O professor vai fazer na prática a construção dos dois métodos, explicando os tipos, passagens de parâmetros e todos os detalhes necessários para a implementação dos algoritmos.

Assista o vídeo se possível replicando o exemplo em seu ambiente de desenvolvimento visando compreender o processo da construção dos métodos, bem como, entender como eles irão funcionar. Lembrando que para este exemplo continuamos utilizando a linguagem de programação Java.



Videoaula 2

Utilize o QRcode para assistir!

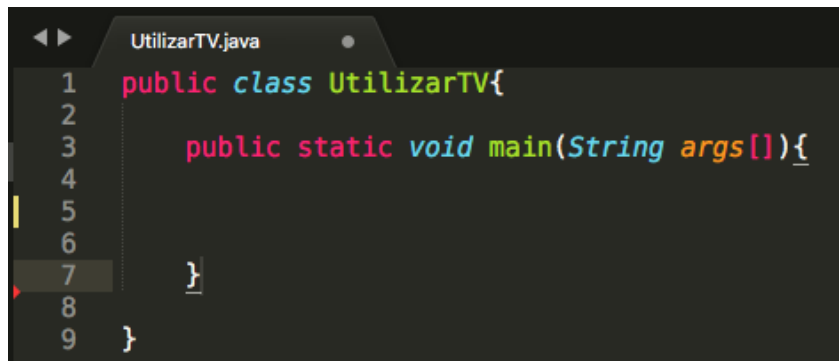
A próxima videoaula, vai abordar a construção do método para definir o canal da TV e do outro método para aumentar o volume da TV.



Para a finalização do exemplo de criação do nosso exemplo de automação, ainda está faltando a construção da logística da classe responsável por executar os métodos que estão na classe `Televisao.java`. O diretório para armazenar as duas classes também deve ser criado a fim de concentrar em um só local os arquivos compilados, ou seja, os arquivos com extensão **.class**.

Vamos primeiramente para a criação da classe `UtilizarTV.java`, dentro dessa classe o método `main()` será construído. Observe a Figura 4.

Figura 4 – Classe UtilizarTV.java

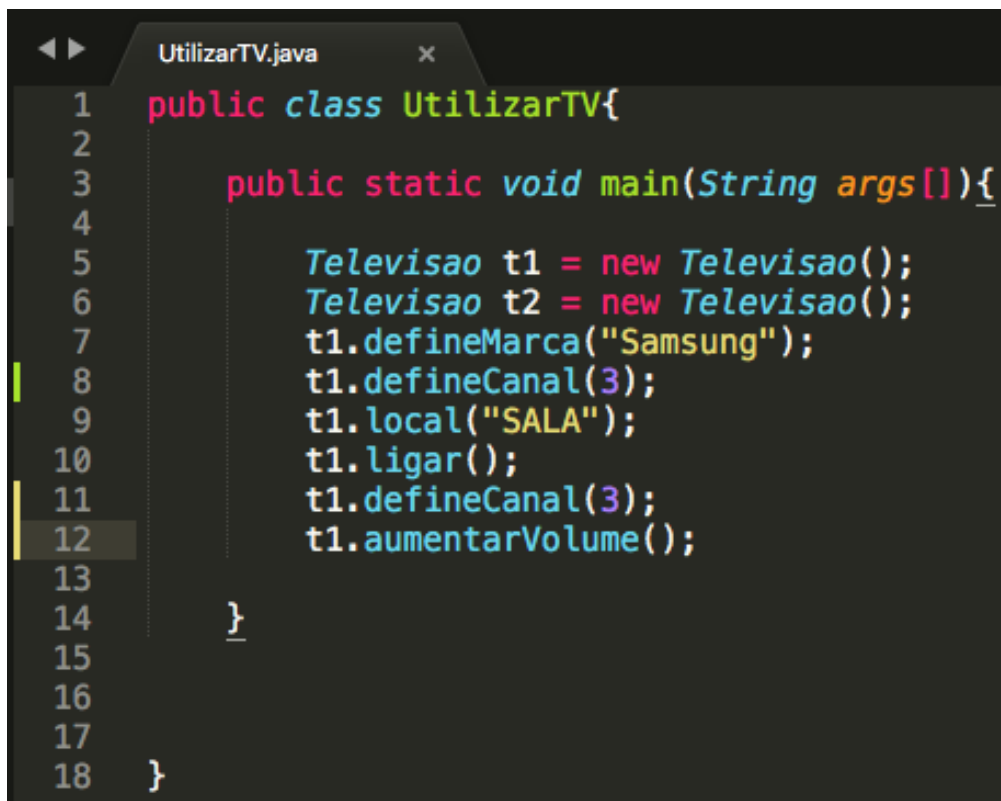


```
UtilizarTV.java
1 public class UtilizarTV{
2
3     public static void main(String args[]){
4
5
6
7     }
8
9 }
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

A partir da Figura 4, dentro do método principal podemos instanciar quantos aparelhos de TV quisermos, basta instanciá-los de forma independente. Após a instância, estamos aptos a executar os métodos definidos anteriormente. Na Figura 5 estamos executando um exemplo com os métodos para definir o canal e também aumentar o volume.

Figura 5 – Chamando métodos na classe UtilizarTV



```
UtilizarTV.java x
1 public class UtilizarTV{
2
3     public static void main(String args[]){
4
5         Televisao t1 = new Televisao();
6         Televisao t2 = new Televisao();
7         t1.defineMarca("Samsung");
8         t1.defineCanal(3);
9         t1.local("SALA");
10        t1.ligar();
11        t1.defineCanal(3);
12        t1.aumentarVolume();
13
14    }
15
16
17
18 }
```

Fonte: elaborado pelo autor via IDE Sublime (2021)

Videoaula 3

A próxima videoaula, vai abordar a criação da estrutura de diretórios e da classe UtilizarTV.java. Durante a criação, o professor vai apresentar o cenário e como realizar a instância e passar valores para os métodos que estão na classe Televisao.java.

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



Videoaula 3

Utilize o QRcode para assistir!

A próxima videoaula, vai abordar a criação da estrutura de diretórios e da classe UtilizarTV.java.



Na próxima Figura, há a exibição de uma estrutura de diretórios para trabalhar com o exemplo. A estrutura foi criada por meio de um diretório com as iniciais “TAD” e dentro do diretório estão todos os arquivos responsáveis pela prática. Os arquivos UtilizarTV.java e Televisao.java são os arquivos codificados com os métodos, instâncias etc. Os arquivos .class são oriundos do processo de compilação que acontece quando executamos o comando de compilação. Veja os arquivos na Figura 6.

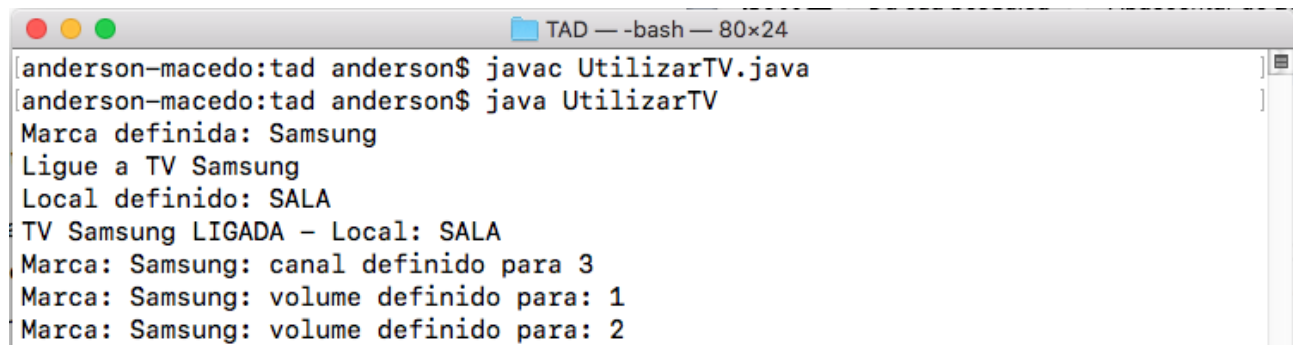
Figura 6 – Chamando métodos na classe UtilizarTV



Fonte: elaborado pelo autor via IDE Sublime (2021)

Para compilar os arquivos e gerar o `.class`, é necessário compilar por meio do comando: `javac + nome do arquivo + extensão`. Observe a Figura 7.

Figura 7 – Compilando os arquivos



```
TAD — -bash — 80x24
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
```

Fonte: o autor (2021)

Um outro meio para se compilar arquivos em Java ou em outra linguagem de programação fora de um ambiente de desenvolvimento é por meio de comandos no *prompt* do Sistema Operacional. A Figura 7 apresenta o comando para compilar o código em Java. Logo depois o código é executado por meio do comando: `java UtilizarTV`.

Aula 02 – Filas Prioritárias

Introdução

Nessa aula, o tema que será abordado são Filas Prioritárias. Nesse primeiro momento vamos fundamentar o conceito de Filas e Filas Prioritárias. Portanto, a Fila é uma estrutura de dado que tem por característica o FIFO ou *First In First Out*, que quer dizer que o primeiro elemento a entrar na fila é o primeiro a sair (CORMEN, 2012). O algoritmo de fila pode ser representado com exemplos de vários tipos de filas que encontramos em nosso dia a dia. Por exemplo, a fila de um banco ou a fila de um caixa de supermercado. Observe uma dessas filas na Figura 1.

Figura 1 – Fila de pessoas

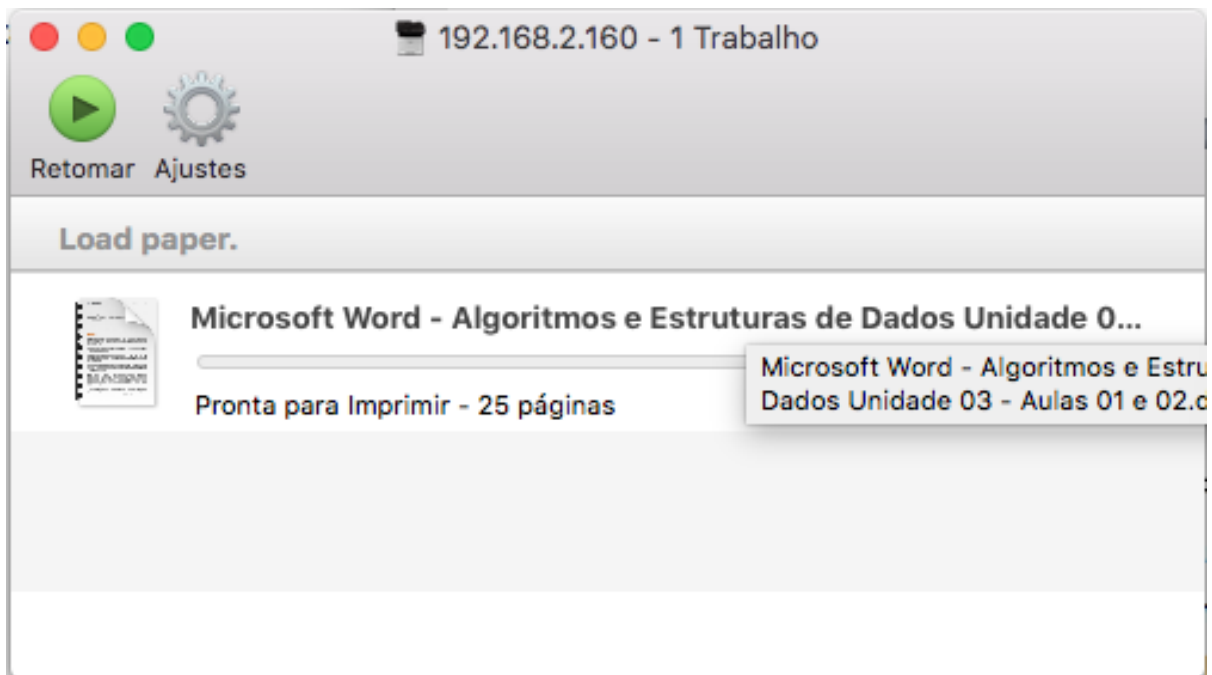


Fonte: <https://www.shutterstock.com/>

Do ponto de vista computacional podemos citar como exemplo dessa estrutura, uma fila de impressão em um equipamento compartilhado. Desse modo, se vários usuários enviarem impressão para esse equipamento, elas serão controladas por meio dessa fila que respeita o mesmo tipo de característica FIFO relatada acima.

No caso de a impressora de rede estar disponível no momento em que se envia um arquivo para a fila de impressão, o programa verifica alguns passos. A impressora faz a leitura do arquivo, verifica a disponibilidade de papel na bandeja e se tudo estiver validado, autoriza o início da impressão. Do contrário, se a impressora estiver ocupada, imprimindo outro arquivo que chegou antes na fila, a fila de impressão vai adicionar o próximo arquivo na fila e assim sucessivamente (CORMEN, 2012). A Figura 2 exibe uma fila de impressão sendo gerenciada pelo programa do sistema operacional.

Figura 2 – Fila de impressão

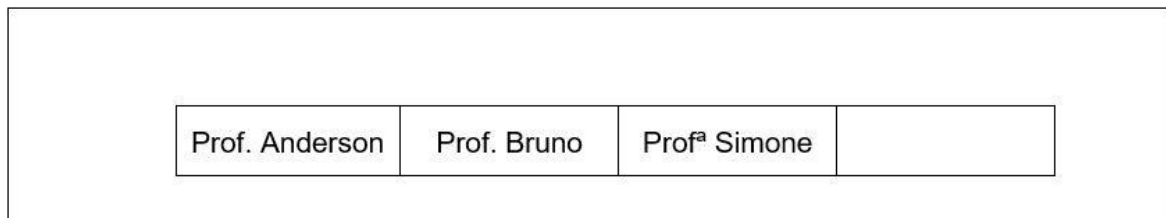


Fonte: elaborado pelo autor via SO: MAC-OS (2021)

Todas as vezes que necessitamos incluir uma pessoa, arquivo, produto ou qualquer tipo de item em uma fila, excluindo-se desta toda a prioridade possível, geralmente, a inclusão sempre será feita no final da fila (CORMEN, 2012).

Em um outro exemplo clássico de estrutura de dados, o Array, a Figura 3 apresenta uma fila de pessoas identificadas pelo seu nome.

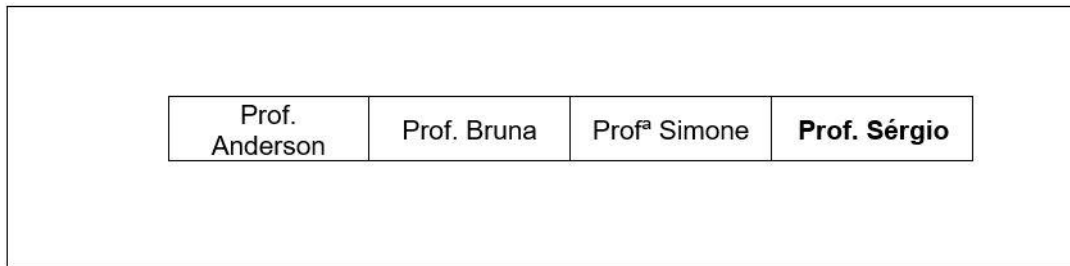
Figura 3 – Fila de *strings*



Fonte: elaborada pelo autor (2021)

De acordo com a Figura 3, se desejarmos inserir o nome do professor “Sérgio” na fila, ele será inserido após o último elemento, para tanto, observe a Figura 4.

Figura 4 – Fila de strings



Fonte: elaborada pelo autor (2021)

Fila Prioritária

Agora vamos abordar essa outra estrutura algorítmica. A Fila Prioritária é uma estrutura que possui um conjunto de elementos, porém esses elementos podem possuir prioridades entre eles. Como possíveis operações dentro de uma fila prioritária estão as operações de inserção e remoção. Essas operações precisam ser gerenciadas da melhor forma, pois agora não estamos mais lidando com filas simples do tipo FIFO, mas sim com filas de prioridades, ou seja, mesmo que o elemento seja inserido na fila por último, ele pode passar na frente dos outros elementos de acordo com a sua prioridade (CORMEN, 2012).

Um exemplo que se encaixa bem na questão de prioridade é a de emergência hospitalar. Quantas pessoas são acometidas de acidentes domésticos, de trânsito, mal súbito ou até mesmo algo mais simples como uma infecção de urina, porém senão for tratada, poderá evoluir a gravidade do quadro do paciente. O fato é, que para controlar o pronto socorro de um hospital, não é nada fácil. O meio mais eficaz é através de uma fila. Porém, a complexidade é grande, pois há mais do que uma fila com várias especialidades médicas e também níveis de prioridades diferentes. Por exemplo, pessoas idosas, gestantes e lactantes já tem uma preferência inicial sobre as demais pessoas, além disso, pessoas que chegam por ambulâncias vítimas de acidentes em estado grave também tem preferência. Podemos resumir os atendimentos entre duas categorias principais, que são: emergência e urgência. A diferença entre essas duas categorias é que a emergência acontece quando há risco eminente de vida. Já a urgência, acontece quando o caso ainda não é tão grave, pois diz respeito a uma ameaça em um futuro próximo, porém se não for

tratada pode evoluir para uma emergência. Tudo isso é verificado na triagem, processo que acontece quando o paciente da entrada no pronto socorro.

Descartando essas duas categorias, o paciente pode ser investigado por um enfermeiro ou médico que faz esse primeiro atendimento. Na Figura 5, você pode conferir algumas categorias de atendimento em um pronto socorro.

Figura 5 – Protocolo de Manchester

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

Fonte: Hospital Santa Cruz (2021)

A fonte da Figura se refere ao Hospital Santa Cruz da cidade com o mesmo nome que fica no Rio Grande do Sul. O Hospital utiliza a classificação de risco pelo Protocolo de Manchester que inclui a cor laranja não utilizada anteriormente.

O Hospital de Santa Cruz a exemplo de muitos outros no Brasil utiliza essa tabela de classificação para classificar o nível de prioridade que cada paciente vai receber.

O Algoritmo

O Algoritmo de Filas Prioritárias pode ser implementado de várias formas, algumas funcionam bem para inserção de elementos na fila e outras para remoção de elementos na fila. A ideia é mesclar essas duas formas em um único algoritmo, tendo assim uma lógica ideal para este tipo de algoritmo.

Uma das formas de representação de uma fila prioritária, é trabalhar com uma lista linear encadeada, fazendo com que os elementos fiquem ordenados por prioridade decrescente. Dessa maneira, na implementação, considerando uma ordem aleatória dos seus elementos, para inserir um novo elemento na fila, basta inseri-lo no final da fila, porém, para remover um elemento da fila de prioridade respeitando a prioridade é claro, é preciso varrer a lista para buscar o elemento com maior prioridade. O próximo passo é remover esse elemento e colocar no seu lugar um outro elemento.

Indicação de Leitura

Para melhor compreensão deste tópico, você pode estudar o livro de CORMEN. O capítulo 6 “Ordenação por Heap”, apresenta a estrutura do algoritmo de filas prioritárias abordando o processo de criação e manipulação dos elementos. Vale a pena conferir.

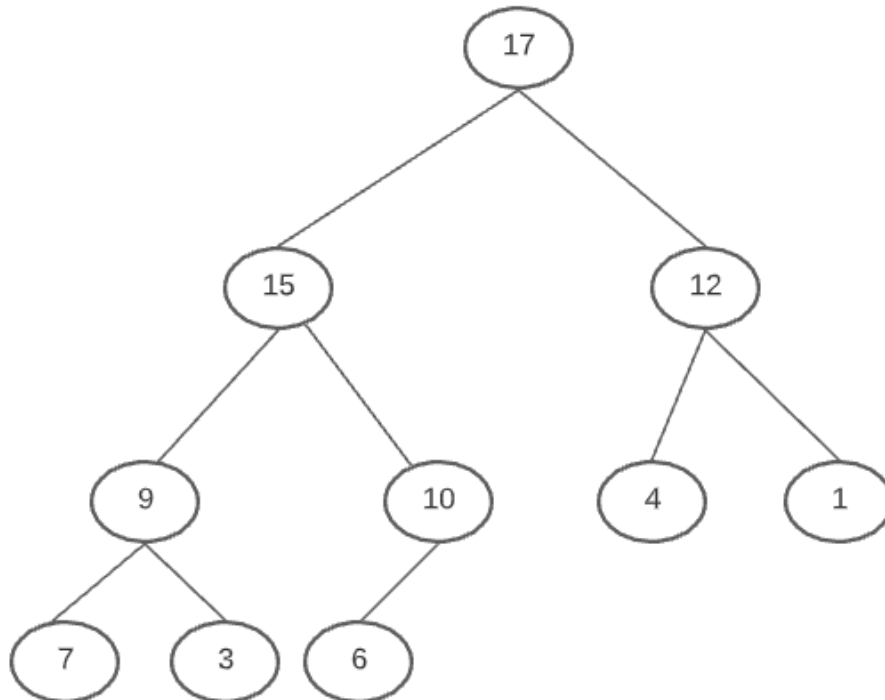
CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. Rio de Janeiro: Campus, 2012. 926 p.

Implementação utilizando “*Heap*”

Heap, é uma estrutura de árvore binária onde cada nó, com exceção dos nós folha, tem prioridade igual ou maior que seus filhos. Em nosso exemplo, vamos considerar que apenas o último nível da árvore pode ficar incompleto, e quando for este o caso, os níveis incompletos devem ser alocados a esquerda da árvore.

Observe a Figura 6, que mostra uma árvore utilizando os algoritmos *heap* com os elementos respeitando a prioridade dos nós pais sobre os nós filhos.

Figura 6 – Árvore Binária



Fonte: elaborado pelo autor (2021)

Observe que na Figura 6, a árvore construída no esquema gráfico mantém a prioridade dos nós pais sobre os nós filhos. Exemplo: o nó 17 é pai dos nós 15 e 12, portanto tem prioridade sobre seus filhos, por esse motivo tem um número maior.

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

Videoaula 1

Nessa videoaula, será abordado o exemplo para a criação da Árvore Binária apresentada na Figura 6, sendo demonstrado por meio da linguagem de programação Java. O professor vai implementar na prática a criação das estruturas de dados.

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



Videoaula 1

Utilize o QRcode para assistir!

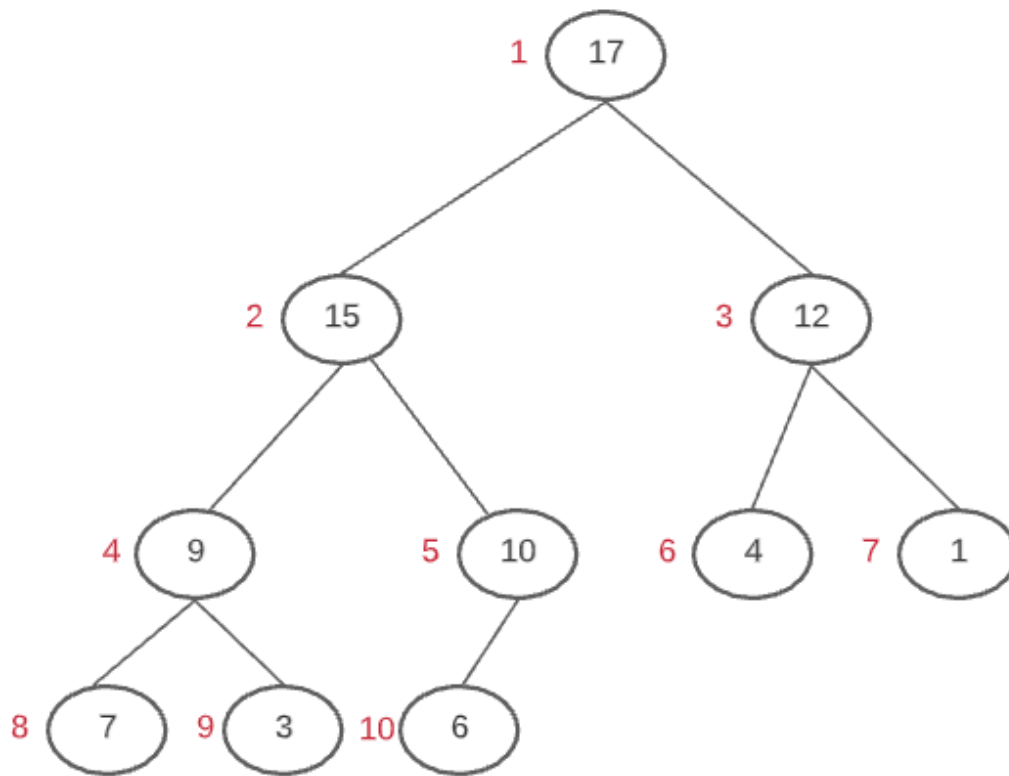
Nessa videoaula, será abordado o exemplo para a criação da Árvore Binária apresentada na Figura 6, sendo demonstrado por meio da linguagem de programação Java.



Numerando os nós

Continuando nosso exemplo, vamos agora numerar cada nó da nossa árvore. Essa ordenação vai se dar por ordem de níveis crescentes da esquerda para a direita. A Figura 7 apresenta como ficará a estrutura da árvore após a numeração dos nós.

Figura 7 – Árvore Binária com numeração dos nós



Fonte: elaborado pelo autor (2021)

Como podemos observar, a Figura 7 exibe destacado em vermelho a ordem de cada nó. O próximo passo é criar um vetor para representar um “*heap*”. Observe a Figura 8.

Figura 8 – Árvore Binária com numeração dos nós

| Vetor: heap | | | | | | | | | | |
|-------------|----|----|---|----|---|---|---|---|----|----|
| 17 | 15 | 12 | 9 | 10 | 4 | 1 | 7 | 3 | 6 | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

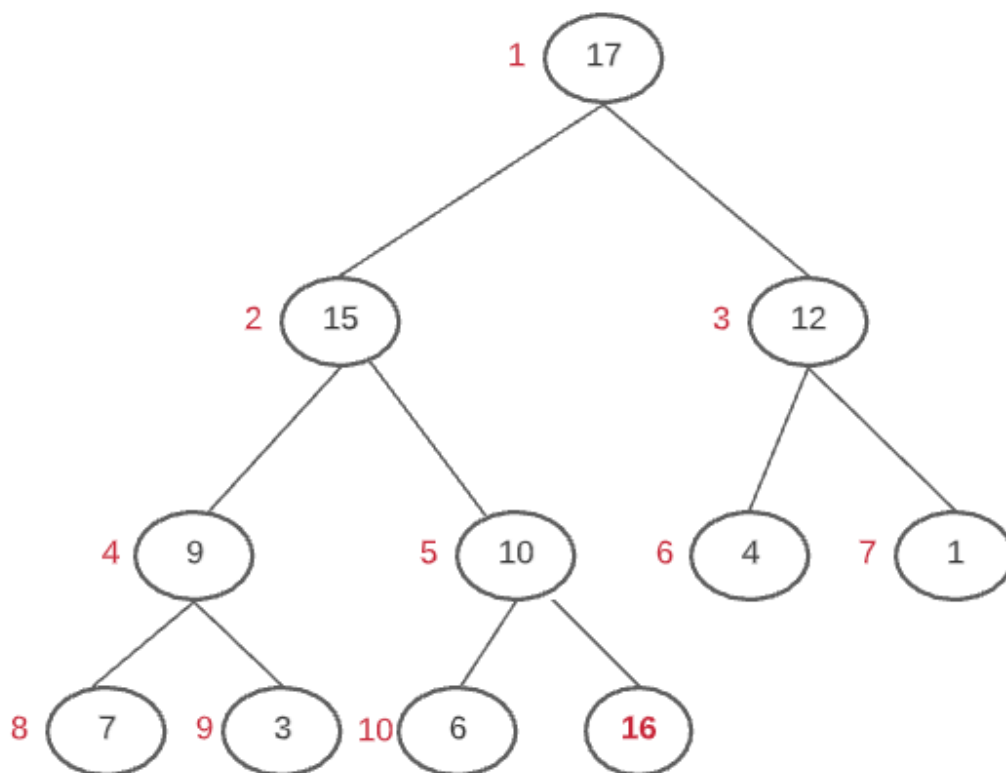
Fonte: elaborado pelo autor (2021)

Inserir um novo elemento

A Figura 8 apresenta o vetor *heap* com a última posição em branco. Nessa posição, vamos inserir um novo elemento. O elemento a ser inserido será o número 16 (dezesesseis). O próximo passo é construir o algoritmo para colocá-lo em ordem de prioridade na árvore.

Para inserir o elemento com uma determinada prioridade, primeiramente é criado um novo elemento no fim do vetor. Em consequência disso, a prioridade atual do *heap* é afetada. Para corrigir a prioridade inicia-se um processo similar ao de ordenação de vetores, ou seja, se o valor que está sendo inserido no final do vetor for maior que o valor de seu pai, então os elementos trocam de lugar. Esse procedimento se repete até que o novo elemento esteja no lugar correto na árvore. A Figura 9 demonstra a entrada do elemento 16 no *heap*.

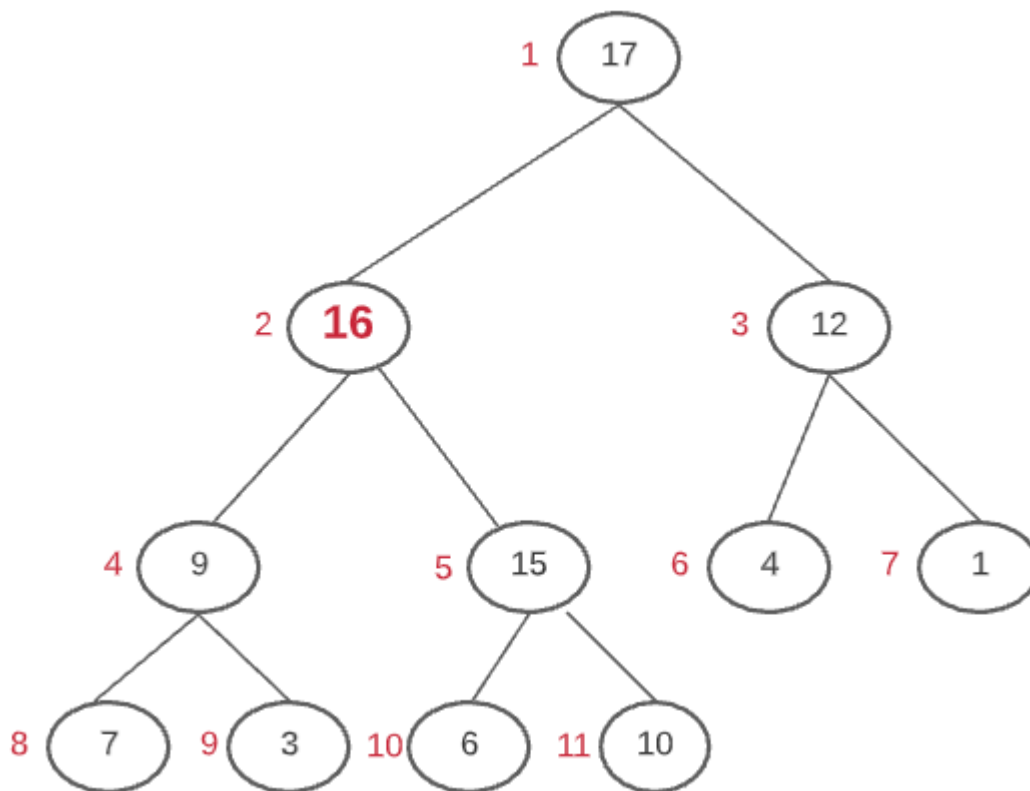
Figura 9 – Árvore Binária – inserção de um novo elemento



Fonte: elaborado pelo autor (2021)

A Figura 10 apresenta o elemento já posicionado corretamente de acordo com sua prioridade, após a aplicação de verificação e mudanças dos elementos.

Figura 10 – Árvore Binária – elemento corretamente posicionado



Fonte: elaborado pelo autor (2021)

Videoaula 2

Nessa segunda videoaula, será abordada a construção do método que vai corrigir a ordem do elemento inserido no vetor. O professor vai implementar os códigos passo a passo e explicar a execução do algoritmo.

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

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



Videoaula 2

Utilize o QRcode para assistir!

Nessa segunda videoaula, será abordada a construção do método que vai corrigir a ordem do elemento inserido no vetor.

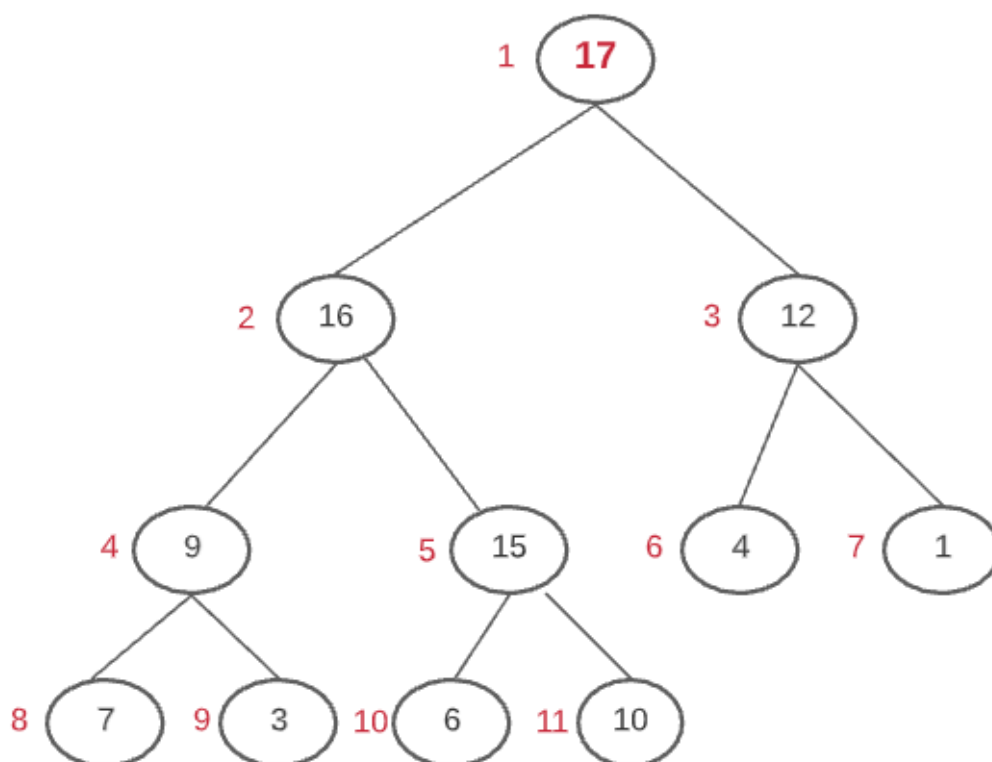


Remover um elemento

A remoção de um elemento é bem mais simples de ser controlada do que a inserção. O elemento a ser removido será sempre o de maior prioridade. Lembrando que estamos trabalhando com o desenvolvimento de algoritmos em filas prioritárias.

Nesse sentido, o valor a ser removido de acordo com a Figura 11 é o elemento 17 que está destacado. Observe a Figura 11 a seguir.

Figura 11 – Árvore Binária – elemento corretamente posicionado



Fonte: elaborado pelo autor (2021)

Videoaula 3

Nessa terceira videoaula, será abordada a construção do método que exclui um elemento do vetor. O professor vai construir a codificação para resolver o caso.

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



Videoaula 3

Utilize o QRcode para assistir!

Nessa terceira videoaula, será abordada a construção do método que exclui um elemento do vetor.



Encerramento da Unidade

Chegamos ao fim desta Unidade de Ensino após caminhar por conteúdos importantes para o seu desenvolvimento acadêmico até aqui. A Unidade 3 apresentou o conteúdo dos tipos abstratos de dados, trabalhando com o paradigma da orientação a objetos em um cenário da automação industrial. Também foi abordado o algoritmo de filas prioritárias demonstrando a inserção e remoção de elementos em uma árvore binária.

Após a assimilação dos conteúdos abordados nesta Unidade, você estará apto a prosseguir estudando um pouco mais sobre a Análise e Projeto de Algoritmos nas próximas Unidades de Ensino.

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

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

Referências

- CORMEN, T. H. **Desmistificando algoritmos**. Rio de Janeiro: Campus, 2012. 926 p.
- CORMEN, T. H. *et al.* **Algoritmos: teoria e prática**. Rio de Janeiro: Campus, 2012. 926 p.
- DEITEL, H. M.; DEITEL, P. J. **Java: como programar**. 8. ed. São Paulo: Pearson, 2010.
- FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**. 2. ed. São Paulo: Makron Books, 2000. 195 p.
- JANDL JUNIOR, P. **Java: guia do programador**. São Paulo: Novatec, 2007. 681 p.
- LAUREANO, M. **Estrutura de dados com algoritmos e C**. São Paulo: Brasport, 2012.
- PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados: com aplicações em Java**. São Paulo: Pearson, 2016.
- RECURSIVIDADE. *In*: DICIO, Dicionário Online de Português. Porto: 7Graus, 2021. Disponível em: <https://www.dicio.com.br/recurividade/>. Acesso em: 20 jul. 2021.
- SANTANA, A. L. **Técnicas de Programação**: Curso Técnico em Informática. Colatina: IFES, 2011.
- XAVIER, G. F. C. **Lógica de programação**. São Paulo: Senac, 2018. 322 p.



UNIFIL.BR