

**EAD**  
**UNISANTA**

## **ESTRUTURA DE DADOS**

Me. Antonio Carlos Marques do Amaral Guerra

**GUIA DA  
DISCIPLINA**

**Objetivo:**

A disciplina Estrutura de Dados tem como objetivo geral, apresentar a importância das diferentes estruturas de dados e suas operações básicas. Explicar de forma clara e através de exemplos, a importância do uso de estruturas estáticas e dinâmicas.

**Introdução:**

Como sabemos, a memória de um computador compõe-se de uma sequência de bytes, controlada pelo Sistema Operacional. Nesta memória são colocados todos os programas executados pelo computador, que são compostos por instruções e dados, respectivamente o código programado e as variáveis a serem utilizadas.

O Sistema Operacional mantém o controle de quais partes da memória estão disponíveis e quais estão ocupadas. Toda vez que um programa é executado, o Sistema Operacional reserva o espaço suficiente para as variáveis deste programa.

Em uma estrutura de dados estática, os bytes necessários são alocados antes que a execução do programa inicie na mesma ordem em que são declaradas. Portanto, em tempo de codificação devem ser definidas as variáveis com os respectivos tipos, tamanhos e quantidades necessárias para a devida reserva de espaço em memória.

Já em uma estrutura dinâmica, veremos que a alocação do espaço em memória ocorre em tempo de execução. À medida que for necessário o armazenamento de uma variável, o espaço em memória é alocado e quando não for mais necessário, os bytes são liberados

## 1. Estruturas Estáticas

As variáveis estáticas são alocadas antes que o programa entre em execução. Ao ser criado o programa, deve ser definido o tipo, tamanho e quantidade de variáveis necessários para que o Sistema Operacional aloque espaço da memória disponível para as tais variáveis.

### 1.1 Estrutura Array

Um Array é um tipo de Estrutura de Dados estática, também chamado de Variável Composta Homogênea. Destrinchando esse termo, podemos dizer que é uma Variável, tanto que tem uma definição com um único nome, é composta, pois consegue armazenar mais de um dado e é homogênea pois os dados que armazena são de um único tipo. Por exemplo, conseguimos armazenar uma quantidade pré-definida de números de R.A. (Registros Acadêmicos) de alunos, nomes, notas, etc.

#### 1.1.1 Vetores

Inicialmente veremos a estrutura array unidimensional, que precisam de apenas um índice para especificar um elemento do conjunto. Arrays unidimensionais também são chamados de Vetores. (Figura 1)

vetor:	dado-1	dado-2	dado-3	dado-4	dado-5	dado-6	dado-7	dado-8	dado-9	dado-10
Índice:	0	1	2	3	4	5	6	7	8	9

Figura 1: Array unidimensional (vetor)

O índice é um número inteiro ou uma variável utilizada como referência para individualizar um dos dados integrantes do array. É indicado entre colchetes todas as vezes que a variável array for referenciada. O valor desse item, naturalmente, estará limitado à quantidade de posições definidas no array. No pseudocódigo utilizaremos a posição dos vetores como índice, iniciando na posição 0 (zero).

### 1.1.1.1 Utilização de vetores

Para facilitar a compreensão da funcionalidade, vamos ver um pequeno exemplo de armazenamento de dados, utilizando um array de 10 elementos, similar ao da Figura 1 acima. Neste exemplo, vamos ver um algoritmo que objetiva armazenar os números de matrículas de 10 alunos, que devem ser digitados. Ao final deve ser emitida listagem com os números de matrículas menores que 100.000.

#### Algoritmo vetor Contagem

```
variáveis
    IND: inteiro
    VETMATR: vetor [0..9] inteiro
início
para IND de 0 até 9 faça
    escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")
    leia (VETMATR[IND])
fim_para

para IND de 0 até 9 faça
    se VETMATR[IND] < 100000
        escreva (VETMATR[IND])
    fim_se
fim_para
fim
```



## Entendendo

Na definição das variáveis, VETMATR foi o nome dado ao nosso array, ou vetor. Poderia ser qualquer outro nome, assim como IND, nome atribuído para a variável que será utilizada como índice do array, isto é, a indicação do posicionamento do referido dado. Ao se declarar o array, o Sistema Operacional já reserva a sequência de bytes necessária para o armazenamento das dez matrículas que serão digitadas.

Esta é uma limitação das estruturas estáticas, pois se um dia for necessário o armazenamento de mais do que 10 números de matrículas, o programa fonte deverá ser alterado para que passe a reservar um espaço maior de armazenamento. Isso torna o programa pouco flexível e mais susceptível a manutenções futuras.

Mas, perceba como seria pior se não existisse o recurso da utilização do array e os dados digitados tivessem que ficar armazenados. Neste simples caso teríamos que definir 10 variáveis de armazenamento com nomes diferentes, o que impediria de utilizarmos as

estruturas de repetição, tanto para o armazenamento quanto para a apresentação das matrículas menores que 100.000. Ficaria algo assim:

**variáveis**

**MATR-01, MATR-01, MATR-01, MATR-01, MATR-01, MATR-01, MATR-01, MATR-01, MATR-01,  
MATR-01: inteiro**

**início**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-01)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-02)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-03)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-04)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-05)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-06)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-07)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-08)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-09)**

**escreva ("DIGITE O NÚMERO DE MATRÍCULA : ")**

**leia (MATR-10)**

**se MATR-01 < 100000**

**escreva (MATR-01)**

**fim\_se**

**se MATR-02 < 100000**

**escreva (MATR-02)**

**fim\_se**

**se MATR-03 < 100000**

**escreva (MATR-03)**

**fim\_se**

**se MATR-04 < 100000**

**escreva (MATR-04)**

**fim\_se**

**se MATR-05 < 100000**

**escreva (MATR-05)**

**fim\_se**

**se MATR-06 < 100000**

**escreva (MATR-06)**

**fim\_se**

**se MATR-07 < 100000**

**escreva (MATR-07)**

**fim\_se**

**se MATR-08 < 100000**

**escreva (MATR-08)**

**fim\_se**

```
se MATR-09 < 100000
    escreva (MATR-09)
fim_se
se MATR-10 < 100000
    escreva (MATR-10)
fim_se

fim
```

Horrível, não?

Agora, como acredito que você tenha entendido, vou deixar uma pergunta para você. Imagine que ao invés de 10 fossem 100 matrículas. Na solução utilizando o array seria apenas mudar sua definição para “[0..99]” e nas duas estruturas de repetição mudar para “de 0 até 99”, sem acrescentar uma linha de codificação.

E na solução sem utilizar o array, como ficaria? Essa eu vou deixar para você fazer, pois nem quero pensar numa solução dessas.

Mas, é só isso que dá para fazer com um array? Logicamente que não!

Vamos considerar que uma revendedora de veículos possua além da loja matriz (0), mais 50 lojas filiais numeradas de 01 a 50 e queira contabilizar a quantidade de vendas efetuadas e o valor total em cada loja.

Assim utilizaremos um vetor com 51 contadores para contar as vendas e a cada venda registrada será adicionado um no contador correspondente. Também teremos outro vetor para totalizar o valor vendido em cada loja. Ao final serão listadas as quantidades e os valores totais das vendas efetuadas em cada loja.

#### Algoritmo vetorRevendedorasVeículos

```
variáveis
    N: inteiro
    VAL: real
    VETQUANT: vetor [0..50] inteiro
    VETVALOR: vetor [0..50] inteiro

início

para N de 0 até 50 faça    // inicializando todos os vetores com zeros
    VETQUANT[N] = 0
    VETVALOR[N] = 0
```

```

fim_para
faça
    escreva ("Digite o número da loja ou 99 para terminar: ")
    leia (N)
    se N < 51
        VETQUANT[N] = VETQUANT[N] + 1
        escreva ("Digite o valor da venda: ")
        leia (VAL)
        VETVALOR[N] = VETVALOR[N] + VAL
    senão
        se N <> 99
            escreva ("Números de lojas válidos: de 0 a 50")
        fim-se
    fim-se
enquanto (N <> 99)

para N de 0 até 50 faça
    escreva ("LOJA " N " – Quant. = " VETQUANT[N] " Valor total = R$ " VETVALOR[N])
fim_para

fim

```



## Entendendo

Analisando o pseudocódigo acima vemos que nas duas estruturas de repetição onde foi utilizado o PARA, a navegação pelo array ocorreram de forma sequencial da primeira à última posição. Já na estrutura onde utilizamos o FAÇA/ENQUANTO o acesso ao array ocorreu de forma aleatória pois nada obriga que a digitação do número da loja seja feita de forma crescente. Se no período em questão uma determinada loja não efetuou uma venda sequer, a posição correspondente permanecerá zerada tanto no vetor que controla a quantidade quanto no que controla o valor total.

Agora vamos adaptar o exercício anterior para considerar que os dados de entrada ao invés de serem digitados, estejam em um arquivo de dados com as vendas previamente cadastradas, que será lido para calcularmos as quantidades e totais de vendas por loja. No arquivo teremos o Número da Loja, o Produto vendido, o valor da venda e a data da venda.

Vamos solicitar qual o mês a ser totalizado e ler o arquivo, contando e totalizando os valores de cada venda realizada no mês.

### Algoritmo vetorRevendedorasVeículos2

variáveis

N: inteiro

MÊS\_TOT: inteiro



**VETQUANT:** vetor [0..50] inteiro

**VETVALOR:** vetor [0..50] inteiro

**VENDAS:** registro

início

N\_LOJA: INTEIRO

PRODUTO: caractere

VALOR: real

DIA\_VENDA: inteiro

MÊS\_VENDA: inteiro

ANO\_VENDA: inteiro

fim\_registro

início

para N de 0 até 50 faça // inicializando todos os vetores com zeros

VETQUANT[N] = 0

VETVALOR[N] = 0

fim\_para

faça

escreva ("Digite o mês a ser totalizado: ")

leia (MÊS\_TOT)

abrir VENDAS

enquanto .não. EOF faça

leia (N\_LOJA , VALOR, MÊS\_VENDA)

se MÊS\_VENDA = MÊS\_TOT

VETQUANT[N\_LOJA] = VETQUANT[N\_LOJA] + 1

VETVALOR[N\_LOJA] = VETVALOR[N\_LOJA] + VALOR

fim\_se

fim\_enquanto

fechar VENDAS

para N de 0 até 50 faça

escreva ("LOJA " N\_LOJA " – Quant. = "VETQUANT[N] " Valor total = R\$" VETVALOR[N])

fim\_para

fim



## Entendendo

Ao utilizarmos um arquivo de dados, o comando Abrir faz com que o Sistema Operacional localize o arquivo e o disponibilize para a utilização pelo nosso programa e o comando Fechar libera sua utilização. A cada leitura, um registro de dados com os dados de uma venda é apresentado substituindo a digitação do exemplo anterior. O arquivo VENDAS foi definido na seção de variáveis com os dados de cada registro. Perceba que neste segundo exercício não fizemos a validação se o número da loja é entre 0 e 50, pois esse tipo de validação conceitualmente deve ser feito quando o arquivo de vendas foi gerado, para garantir a integridade do dado.



### 1.1.1.2 Mãos à Obra

Para cada exercício deste Guia, você encontra uma solução no último Capítulo, mas tente fazer sem olhar a solução apresentada. Como você sabe, em programação não há uma única solução correta, mas após concluir o exercício verifique a solução apresentada para verificar se a sua solução também está correta.

#### Exercício 1.1.1.2-A

E se quisermos ao final do Algoritmo **vetorRevendedorasVeículos**, além de listar as quantidades e os valores totais das vendas efetuadas em cada loja, quisermos também identificar qual ou quais as lojas que tiveram o maior valor de venda?

Bem.... Fica aí um desafio que sei que você será capaz de resolver. Se achar que não consegue, releia os tópicos anteriores, mas é muito importante você pelo menos tentar resolver o desafio proposto.

#### Exercício 1.1.1.2-B

Fazer o pseudocódigo para um programa que leia um arquivo sequencial contendo o mês de nascimento dos alunos cadastrados e imprima a quantidade de aniversariantes por mês, utilizando vetor.

Resultado Esperado:

Quantidade de Aniversariantes por mês

01 = xxx  
02 = xxx  
03 = xxx  
04 = xxx  
05 = xxx  
06 = xxx  
07 = xxx  
08 = xxx  
09 = xxx  
10 = xxx

11 = xxx

12 = xxx

### 1.1.1.3 Pesquisa Sequencial em um vetor

Quando temos um vetor com grande quantidade de dados já preenchidos e queremos procurar um determinado elemento que pode constar ou não, a solução mais imediata é procurarmos um a um em cada posição desse vetor e somente concluiremos que o elemento procurado não consta, quando chegarmos no último sem encontrá-lo.

Imagine, por exemplo, um professor que após a aplicação de uma prova escrita do ENEM, tenha em mãos 200 provas impressas cada uma com o número de matrícula dos alunos, na ordem em que as provas foram concluídas e entregues. Se o professor quiser buscar a prova de um determinado aluno, provavelmente irá verificar se o número de matrícula procurado é o da primeira prova, posteriormente o da segunda, terceira, até encontrar ou chegar na última e possivelmente concluir que a prova desse aluno não consta naquele conjunto de provas. Continuando a localização das provas de outros alunos, quando encontra, pode ser que tenha sido logo a primeira, pode ser justamente a última ou encontrá-la em qualquer posição do maço de provas.



## Importante

Podemos concluir que quando o valor procurado consta no vetor, são feitas em média 100 comparações, OK?

Já, os números procurados e não encontrados exigem 200 comparações.

Esse é um exemplo de uma Pesquisa Sequencial em um vetor não ordenado. E como ficaria um pseudocódigo para essa pesquisa?

#### Algoritmo vetorPesquisaSequencial

variáveis

I, MATRPROC: inteiro

VETPROVAS: vetor [0..199] inteiro // Já preenchido com números de matrículas

início

escreva ("Digite a matrícula a ser procurada: ")

leia (MATRPROC)

```

I = 0
enquanto (I < 200) .e. (MATRPROC <> VETPROVAS[I]) faça
    I = I + 1
fim_enquanto

se I = 200
    escreva ("Matrícula não encontrada")
senão
    escreva ("Matrícula encontrada na posição " I)
fim_se

fim

```



## Entendendo

Perceba que na rotina de repetição ENQUANTO/FAÇA utilizada para a Pesquisa Sequencial, apenas ocorre a navegação do primeiro ao último elemento e na própria condição do ENQUANTO temos as duas situações que fazem o loop parar e por estarem ligadas com o conector E, a navegação só continua enquanto as duas condições estiverem sendo atendidas. Na saída do loop é verificada qual das condições provocou a descontinuidade da repetição, concluindo se a matrícula consta ou não no vetor.

Mas, e se os dados constantes no vetor estiverem classificados em ordem crescente? Podemos otimizar a busca sequencial simplesmente interrompendo a pesquisa ao identificar um valor maior do que o procurado. Vamos entender a mudança no pseudocódigo.

### Algoritmo vetorOrdenadoPesquisaSequencial

variáveis

I, MATRPROC: inteiro

VETPROVAS: vetor [0..199] inteiro // Já preenchido com matrículas em ordem crescente

início

escreva ("Digite a matrícula a ser procurada: ")

leia (MATRPROC)

I = 0

enquanto (I < 200) .e. (MATRPROC < VETPROVAS[I]) faça

I = I + 1

fim\_enquanto

se (I = 200) .ou. (MATRPROC > VETPROVAS[I])

```

    escreva ("Matrícula não encontrada")
senão
    escreva ("Matrícula encontrada na posição " I)
fim_se
fim

```



## Entendendo

Veja que a primeira mudança foi na condição do ENQUANTO trocando o "<>" por "<". Agora o loop é interrompido não só quando encontrar o valor procurado, mas também quando encontrar um valor maior. Assim, mesmo que logo na primeira posição do vetor o valor seja maior do que o procurado, o loop nem é iniciado, concluindo-se que o valor procurado não se encontra no vetor. A outra alteração foi incluir na identificação da mensagem a ser apresentada a possibilidade da conclusão de que a Matrícula não consta no vetor, mesmo sem tê-lo percorrido inteiro.

No caso de o conteúdo do vetor estar ordenado, a diferença de performance é quando o valor procurado não consta no vetor, que se apresenta de forma semelhante ao valor encontrado. Podemos estar procurando um valor menor que o primeiro elemento do vetor e já parar de procurar, mas também se procurarmos um valor maior do que o constante no final do array, teremos que comparar com os 200 elementos do array.



## Importante

Podemos concluir que quando o valor procurado consta no vetor, são feitas em média as mesmas 100 comparações do caso anterior. Já, com o array ordenado, também na situação de busca de um valor inexistentes são realizadas em média 100 comparações.

Vamos ver um outro exemplo de Pesquisa Sequencial:

Fazer uma busca num texto de no máximo 100 caracteres constante em um array e apresentar a quantidade de letras A digitadas nesse texto.

```

Algoritmo vetorContaLetraA
    variáveis
        CONT, LET: inteiro

```

```

TEXTO: vetor [0..99] caractere // Já preenchido com um texto
início

CONT = 0
para LET de 0 até 99 faça
    se TEXTO[LET] = "A"
        CONT = CONT + 1
    fim_se
fim_para
escreva ("Quantidade de A que aparece no texto:" CONT)

fim

```

#### 1.1.1.4 Pesquisa Binária em um vetor

Esse tipo de pesquisa só se aplica a vetores previamente ordenados. É muito eficiente principalmente quando utilizados em vetores com grande quantidade de elementos.

Neste método, o vetor sempre é dividido em dois, originando daí a sua denominação. A primeira comparação é feita com o elemento do meio do vetor. Não sendo o elemento procurado, elimina-se uma das metades do vetor, pois se a ordem é crescente e o valor procurado é menor do que o encontrado no vetor, sabemos que se constar no array, o valor procurado estará na primeira metade e se for maior estará na metade seguinte. Seguem-se comparações sucessivas ao elemento do meio do segmento onde pode estar o elemento procurado. Assim, a cada comparação feita, metade do vetor é eliminada.

Podemos utilizar o exemplo do Algoritmo vetorOrdenadoPesquisaSequencial, que já considerava o vetor ordenado de forma crescente, para exemplificar a Pesquisa Binária.

#### Algoritmo vetorPesquisaBinária

```

variáveis
    I, MATRPROC, PRI, ULT, MED: inteiro
    VETPROVAS: vetor [0..199] inteiro // Já preenchido com matrículas em ordem crescente
início

    escreva ("Digite a matrícula a ser procurada: ")
    leia (MATRPROC)

    PRI = 0
    ULT = 199
    MED = 99
    enquanto PRI <= ULT .e. (MATRPROC <> VETPROVAS[MED]) faça
        se MATRPROC < VETPROVAS[MED]
            ULT = MED - 1
        senão

```

```

        PRI = MED + 1
    fim_se
    MED = (PRI + ULT) / 2
fim_enquanto

se MATRPROC <> VETPROVAS[MED]
    escreva ("Matrícula não encontrada")
senão
    escreva ("Matrícula encontrada na posição " MED)
fim_se

fim

```

Na pesquisa binária com 200 posições no array, sempre dividindo ao meio faremos apenas 7 (sete) comparações no pior cenário que é quando procura um valor inexistente. Se o valor constar, será encontrado em até 7 comparações.

E se ao invés de 200 elementos fossem 400, quantas comparações seriam necessárias?

Se você respondeu 14 não tenha vergonha, pois muitos devem ter respondido igual, mas está errado.



## Importante

Seriam apenas 8 (oito) comparações, pois logo na primeira cortaria de 400 para 200 e aí já vimos que precisaria apenas de mais 7 comparações.

Isso justifica termos começado a definição da Pesquisa Binária como sendo "muito eficiente principalmente quando utilizados em vetores com grande quantidade de elementos".

### 1.1.1.5 Mãos à Obra

#### Exercício 1.1.1.5-A

Retomando o exemplo dado através do Algoritmo vetorContaLetraA, desenvolva a alteração para, ao invés de contar apenas quantas letras A constam no texto, contar a quantidade de cada letra do alfabeto.

Dica:

Para isso, utilize um novo array contendo cada letra do alfabeto e repita a busca no texto para o alfabeto todo.

[illegible]

ALFABETO[0..25]																												
A	B	C	D	E																						X	Y	Z
0	1	2	3	4	5	.....																					24	25

### 1.1.2 Matrices

Agora que você já sabe as vantagens e facilidade de usar um vetor, se você precisar armazenar as médias obtidas pelos 50 alunos de uma classe certamente irá optar por usar um vetor de 50 posições. Agora, e se ao invés da média, você quiser armazenar suas 8 notas obtidas pelos 50 alunos para calcular as médias? Seria uma boa ideia usar um vetor de 8 posições para cada um dos 50 alunos? Sim, seria.

Mas aí não vai complicar muito? É possível fazer isso?

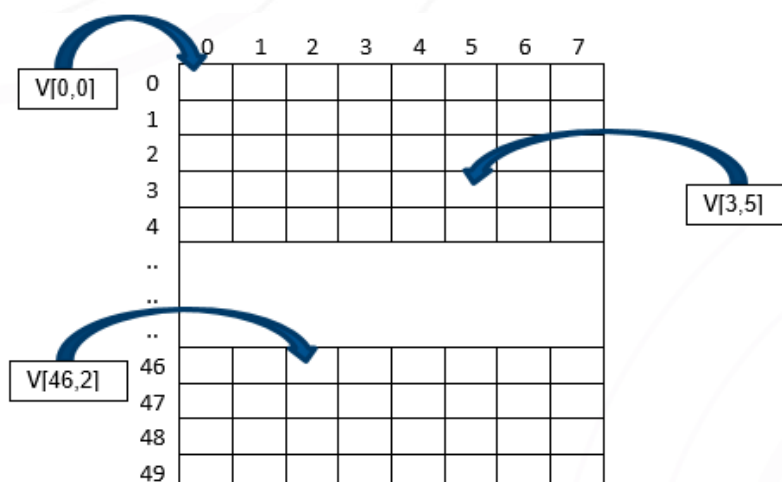
Sim, claro. E você mesmo já deve ter usado algo parecido quando abriu uma planilha Excel. É só separar 8 colunas em 50 linhas. Você estará usando um vetor de 50 linhas, uma para cada aluno e nelas usando um vetor para as 8 notas.

Esse “vetor de vetores” é chamado de Matriz, ou de Array Bidimensional.

E é possível mais do que duas dimensões? Bem, essa resposta você vai receber dependendo da linguagem de programação que será utilizada, mas veja, no próprio Excel trabalhamos com uma terceira dimensão (plan1, plan2, plan3,...).

Assim como ao se referir a uma célula de um vetor precisamos utilizar um índice. Em um Array Bidimensional são necessários dois índices, um para cada dimensão.





### 1.1.2.1 Utilização de Matrizes

Então, para provar que é fácil, vamos ver como fica o pseudocódigo do exemplo citado acima:

#### Algoritmo MatrizNotas

**variáveis**

I, J: inteiro

SOMA: real

NOMES: vetor [0..49] caractere

NOTAS: vetor [0..49, 0..7] real

MEDIAS: vetor [0..49] real

**Início**

//OBTENÇÃO DAS NOTAS

**para** I = 0 até 49 **faça**

**escreva** ("Informe o nome do aluno ", I )

**leia** (NOMES[I])

  SOMA = 0

**para** J de 0 até 7 **faça**

**escreva** ("Digite a ", J+1, "ª nota do aluno ", nomes[I], ": ")

**leia** (NOTAS[I, J])

    SOMA = SOMA + NOTAS[I, J]

**fim\_para**

  MEDIAS[I] = SOMA / 8

**fim\_para**

//APRESENTAÇÃO DOS RESULTADOS

**para** I = 0 até 49 **faça**

**escreva** ("Aluno: ", nomes[I], "Notas: ")

**para** J de 0 até 7 **faça**

**escreva** (NOTAS[I, J])

**fim\_para**

**se** MEDIAS[I] >= 7

**escreva** ("Aprovado com média: ", medias[I])

```

senão
    escreva ("Reprovado com média: ", medias[I] )
fim_se
fim_para
fim

```

### 1.1.2.2 Mãos à Obra

#### EXERCÍCIO 1.1.2.2-A

Com base no exercício feito no item referente à Pesquisa Sequencial onde foi apresentado o **Algoritmo vetorContaLetraA**, desenvolva um pseudocódigo que faça a contagem de letras A, agora em uma matriz de 100 colunas e 40 linhas.

#### EXERCÍCIO 1.1.2.2-B

Agora, de preferência sem olhar o algoritmo correspondente no Vetor, faça a contagem de cada letra do alfabeto na nossa matriz de 100 colunas e 40 linhas.

### 1.1.3 Métodos de Ordenação

Imagine uma enorme biblioteca, com milhares de livros dispostos nas prateleiras ou uma farmácia com inúmeras caixas de remédios, como seria difícil a localização de um exemplar se não estivessem dispostos numa determinada ordem. Se os livros de uma coleção forem numerados, podem estar ordenados de forma crescente pelo número impresso na capa ou pelo código de tombamento de uma biblioteca. Mas você pode preferir ordenar os livros de sua estante em ordem alfabética de título do livro para facilitar a localização.



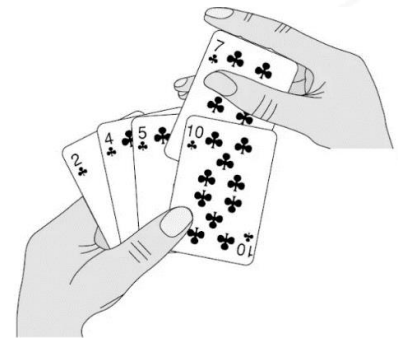
Agora imagine que você fez recebeu dezenas de livros, que estão em ordem totalmente aleatória, e quer organizá-los numa determinada ordem. Bem, você já deve estar imaginando um método para essa tarefa, que não necessariamente seja o mesmo de outro colega e ambos dessem o mesmo resultado. A diferença é que um pode dar menos trabalho ou ser mais rápido do que o outro.

Em um vetor, vimos que se os elementos estiverem dispostos em uma ordem conhecida, podemos utilizar algoritmos mais eficientes. Mas se estiverem dispostos de

forma aleatória, podemos usar um método de ordenação, alterando a disposição dos seus elementos. Existem vários métodos tradicionais, dentre os quais veremos alguns, mas se você for pesquisar mais, encontrará diversos outros métodos.

### 1.1.3.1 Método Insertion Sort

O Insertion Sort ou seja, ordenação por inserção, cria uma partição do vetor inicialmente apenas com o primeiro elemento e vai aumentando essa partição até completar o vetor, mantendo os elementos dessa partição ordenados, inserindo um a um os elementos na posição correta para manter a ordem desejada. Quando encontra um valor menor que os anteriores, desloca os elementos da partição já ordenada para liberar o lugar correto desse novo valor.



Podemos comparar o método com a forma que normalmente um jogador arruma as cartas do baralho em sua mão. Ao abrir suas cartas elas estão desordenadas e uma a uma vai localizando o seu lugar entre as demais, até que todas as cartas estejam na ordem desejada.

Vamos simular passo a passo o método da ordenação por Inserção para ordenar, como exemplo, os algarismos 4 2 3 5 1:

PASSO 1:  $V[0] > V[1] \rightarrow \text{TROCA}$

0	1	2	3	4	AUX
4	2	3	5	1	2

0	1	2	3	4	AUX
4	4	3	5	1	2

0	1	2	3	4	AUX
2	4	3	5	1	2

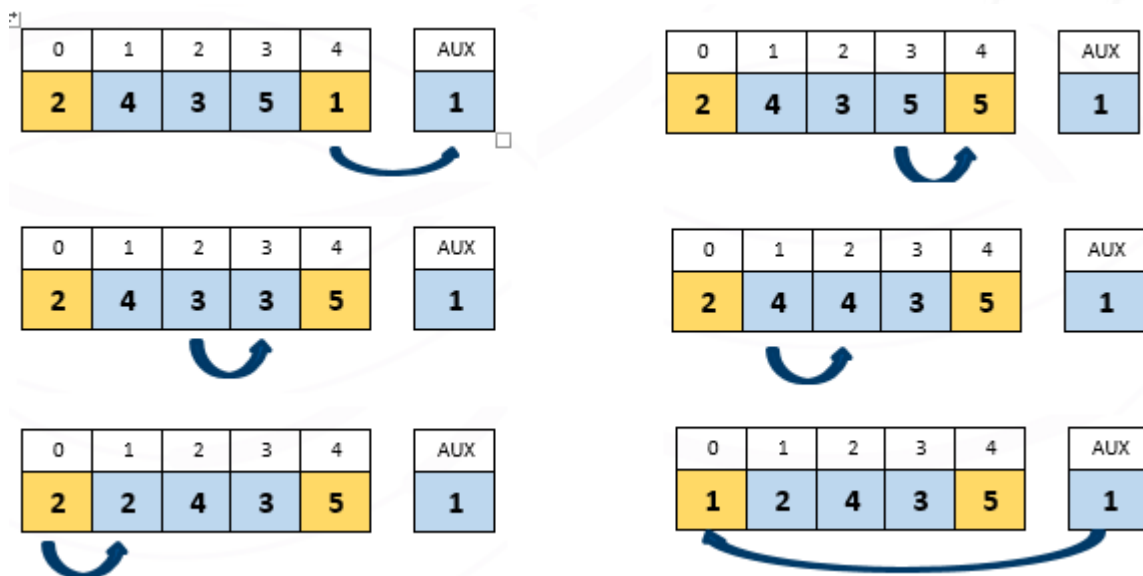
PASSO 2:  $V[0] < V[2] \rightarrow$  MANTÉM

0	1	2	3	4
2	4	3	5	1

PASSO 3:  $V[0] < V[3] \rightarrow$  MANTÉM

0	1	2	3	4
2	4	3	5	1

PASSO 4:  $V[0] > V[4] \rightarrow$  TROCA DESLOCANDO ATÉ ENCONTAR SEU LUGAR



PASSO 5:  $V[1] < V[2] \rightarrow$  MANTÉM

0	1	2	3	4
1	2	4	3	5

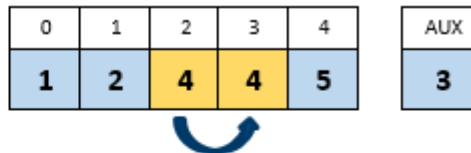
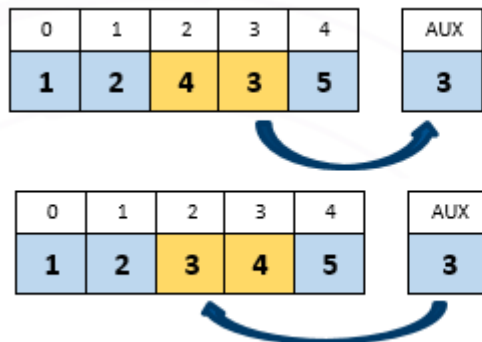
PASSO 6:  $V[1] < V[3] \rightarrow$  MANTÉM

0	1	2	3	4
1	2	4	3	5

PASSO 7:  $V[1] < V[4] \rightarrow$  MANTÉM

0	1	2	3	4
1	2	4	3	5

PASSO 8:  $V[2] > V[3] \rightarrow$  TROCA



PASSO 9:  $V[2] < V[4] \rightarrow$  MANTÉM

0	1	2	3	4
1	2	3	4	5

PASSO 10:  $V[3] < V[4] \rightarrow$  MANTÉM

0	1	2	3	4
1	2	3	4	5

Veja como fica simples o pseudocódigo para uma Pesquisa por Inserção em um vetor com 50 ocorrências:

### Algoritmo InsertionSort

variáveis

I, J, AUX: inteiro

VET: vetor [0..49] inteiro // Já preenchido em ordem aleatória

início

para I de 1 até 49 faça

AUX = VET[I]

J = I - 1

enquanto (J >= 0) .e. (VET[J] > AUX) faça

VET[J+1] = VET[J]

J = J - 1

fim\_enquanto

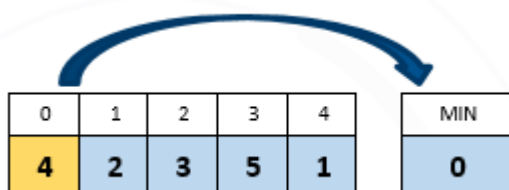
$VET[J+1] = AUX$   
 fim\_para  
 fim

### 1.1.3.2 Método Selection Sort

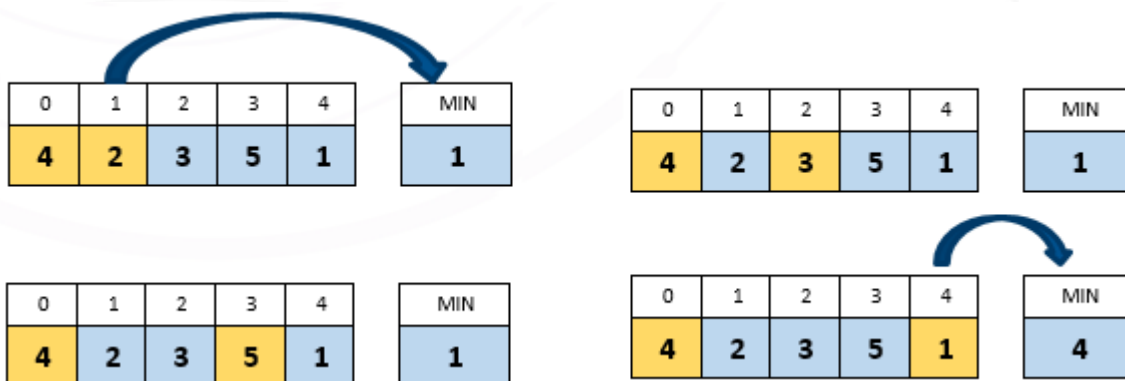
Para ordenar de forma crescente, define uma posição do vetor sequencialmente da esquerda para a direita e analisa o conteúdo de todos os demais até o final do vetor em busca da posição que contém o menor valor. Se for menor do que o conteúdo da posição definida, troca os conteúdos de posição, garantindo que todas as posições à sua esquerda estejam ordenadas com os menores valores do array. Repete esse procedimento até que todos os valores estejam ordenados.

Vamos ver uma simulação de ordenação pelo Selection Sort:

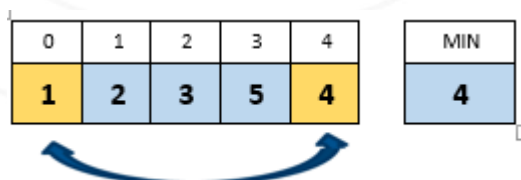
PASSO 1: Define Posição 0 e guarda índice



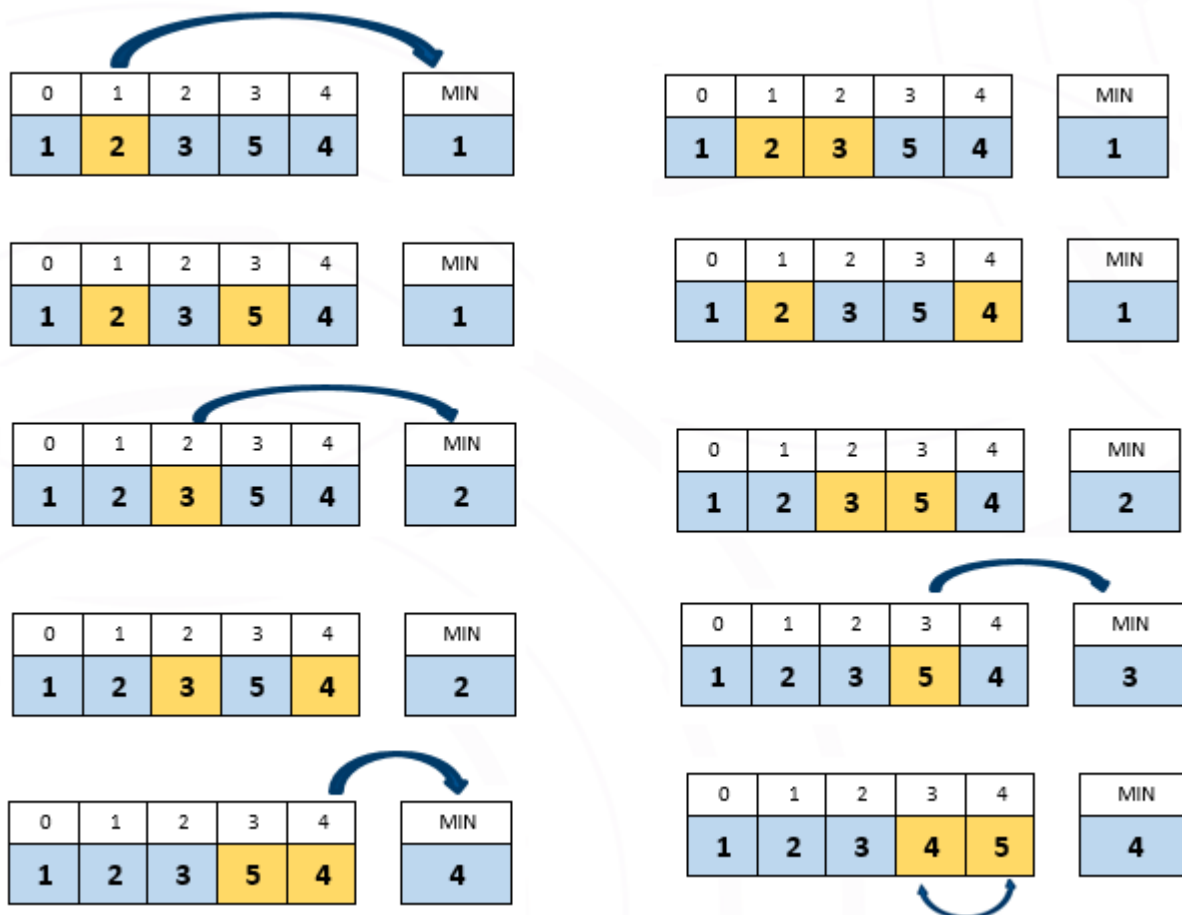
PASSO 2: Procura por conteúdo menor até o final do vetor e guarda índice do menor



PASSO 3: Quando encontra um valor menor do que o da posição definida, troca os conteúdos. Nesse momento o índice 0 contém o menor valor do array.



PASSO 4: Repete o processo definindo sequencialmente o próximo índice para conter o menor valor entre as demais posições do array.



E será que é difícil fazer o pseudocódigo para o Selection Sort?

Que nada, veja só como fica para ordenar um vetor com 50 elementos:

### Algoritmo SelectionSort

variáveis

I, J, MIN, AUX: inteiro

VET: vetor [0..49] inteiro // Já preenchido em ordem aleatória

início

para I = 0 até 48 faça

    MENOR = I

    para J = I+1 até 49 faça

        se VET[J] < VET[MIN]

            MIN = J

    fim\_se



```

fim_para
se MIN < I
    AUX = VET[I]
    VET[I] = VET[MIN]
    VET[MIN] = AUX
fim_se
fim_para
fim

```



## Pesquisar

Além dos dois métodos de ordenação aqui apresentados, existem diversos outros, como o BUBBLE SORT, o MERGE SORT, o QUICK SORT, etc., com diferentes níveis de eficiência e complexidade.

Pesquise também os níveis de complexidade de cada método pesquisado e dos métodos apresentados como exemplo.

Desperte a curiosidade e o espírito de pesquisador que existe em você para se aprofundar no assunto.

### 1.1.4 Listas Estáticas

Certamente você já teve contato com muitas listas. Lista Telefônica, Lista de presentes de Natal, Lista compras do supermercado, Lista nominal de alunos, etc. Sim, qualquer lista pode ser implementada de forma estática num array, mas para tal, precisa ser dimensionada na quantidade máxima de elementos. Considere que a quantidade máxima de uma turma é de 40 alunos e você implemente uma Lista em ordem alfabética. Naturalmente, alguns alunos podem sair dessa turma e outros podem entrar e você deve alterar a Lista, mantendo os nomes em ordem. Como você já conhece a estrutura array, deve ter percebido que para incluir um novo aluno que não seja o nome maior na ordem alfabética, todos os maiores precisam ser deslocados para abrir espaço para a inclusão. De forma semelhante, a retirada de um nome provoca o deslocamento dos posteriores para preencher o espaço vazio.


Mas, dependendo da situação, podem ser usadas algumas Listas com restrições de entrada e saída, como veremos a seguir:

#### 1.1.4.1 Estrutura Pilha

A estrutura de dados Pilha também é conhecida como **LIFO**, por ser acrônimo de **Last In, First Out**, isto é, o último que entra é o primeiro que sai (UEPS).



A denominação Pilha é por lembrar um empilhamento, por exemplo de containers, onde todo container a ser incluído ocupa o topo da pilha e ao saírem, precisa ser tirado sempre o que está no topo da pilha. Embora os dados sejam armazenados na memória, não sobrepostos como o exemplo dos containers, a estrutura segue essa mesma regra.

Um exemplo de aplicação que você certamente conhece é a função Desfazer  do Office ou CTRL Z. O Sistema Operacional “empilha” as alterações e a função ao “desempilhar” ao ser acionada várias vezes, traz de volta o que foi alterado por último, penúltimo, e assim por diante.

Como a regra só permite acessar o último dado incluído para encadear o próximo ou para retirar, a estrutura só precisa conhecer a posição do último nó incluído.

A Figura abaixo, retirada de livro disponível na Biblioteca Virtual, representa a estrutura de dados Pilha.

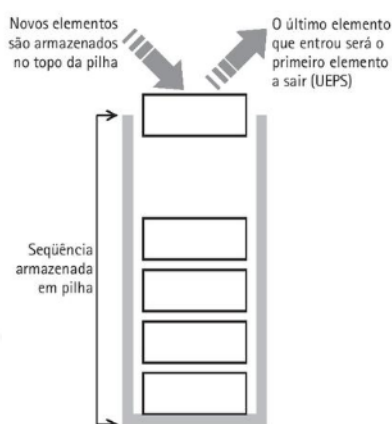


FIGURA - retirada do livro Lógica de Programação e Estrutura de Dados de Sandra Puga e Gerson Rissetti

Um exemplo clássico de utilização de Pilhas é o Jogo Torre de Hanoi.

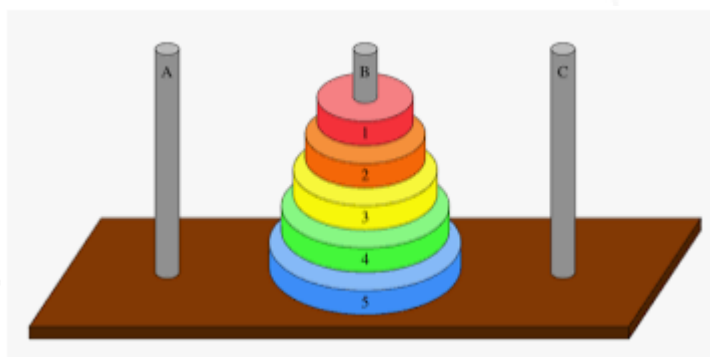
A Torre de Hanói é um quebra-cabeça que consiste em uma base contendo três pinos, em um dos quais são dispostos alguns discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O problema consiste em passar todos os discos de um pino para um outro, usando um dos pinos como auxiliar, de maneira que um disco

maior nunca fique em cima de outro menor e as movimentações sejam feitas um disco de cada vez. O objetivo é concluir a transferência dos discos com a menor quantidade de movimentações possível.

Esse jogo foi inspirado numa lenda onde eram utilizados 64 discos. Você pode construir esse jogo com peças físicas ou digitalmente, mas certamente não com uma quantidade tão grande como consta na lenda. Bem, quanto à lenda, eu vou deixar você matar a curiosidade através da busca na internet, onde você também encontrará sites inclusive com o jogo disponibilizado para você praticar o seu raciocínio lógico.

É interessante observar que o número mínimo de "movimentos" para conseguir transferir todos os discos da primeira estaca à terceira é  $2^n - 1$ , sendo  $n$  o número de discos.

Portanto, para solucionar a Torre de Hanói com 5 discos, que é a quantidade mais tradicional, são necessários 31 movimentos. Com 7 discos, são necessários 127 movimentos, com 15 discos, o mínimo será 32.767 movimentos, e com 64 discos, como diz a lenda, são necessários 18.446.744.073.709.551.615 movimentos.

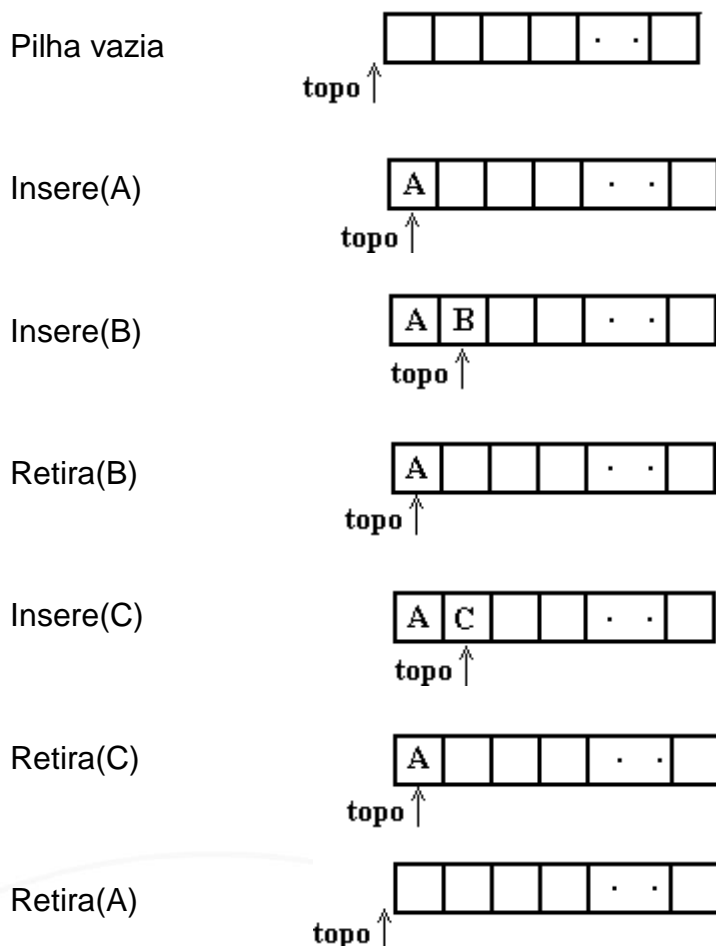


## Pesquisar

Faça uma busca na internet com o termo Torre de Hanoi. Você vai encontrar inclusive programas de jogos disponíveis, com lógica recursiva ou não. Programas prontos para você jogar escolhendo a quantidade de discos ou programas que mostram a solução do jogo.

Se você se animar e tiver habilidade para tal, tente criar o seu jogo Torre de Hanoi.

Inicialmente veremos a implementação de uma Pilha numa estrutura estática (vetor), onde a variável **Topo**, que indica a posição do último elemento inserido, é adicionado em 1 no momento da inserção (função **PUSH**) e decrescido de 1 quando ocorre a retirada de um elemento da Pilha (função **POP**). Em estrutura estática na função **Push** é necessária a verificação se a Pilha não está com sua capacidade esgotada.



## EXEMPLO DE UTILIZAÇÃO DE PILHA ESTÁTICA

Conferir uma expressão matemática e indicar se as aberturas de parêntesis, chaves e colchetes estão coerentes com os respectivos fechamentos.

### Algoritmo ChecaExpressãoEstática

variáveis

**I, IND:** inteiro

**RESULTADO:** caractere

**EXPRESSAO:** vetor [0..49] inteiro // Já preenchido com uma expressão matemática

```

PILHA: vetor [0..19]
Início
IND = 0
RESULTADO = "OK"
I = 0
Enquanto i < 49 .e. RESULTADO = "OK" faça
    se EXPRESSAO[i] = '{'
        PILHA[IND] = '}'
        IND = IND + 1
    fim_se
    se EXPRESSAO[i] = '['
        PILHA[IND] = ']'
        IND = IND + 1
    fim_se
    se EXPRESSAO[i] = '('
        PILHA[IND] = ')'
        IND = IND + 1
    fim_se

    se EXPRESSAO[i] = '}' .ou. EXPRESSAO[i] = ']' .ou. EXPRESSAO[i] = ')'
        se IND = 0
            RESULTADO = "INCORRETA"
        senão
            se EXPRESSAO[i] = PILHA[IND]
                IND = IND - 1
            senão
                RESULTADO = "INCORRETA"
            fim_se
        fim_se
    fim_se

    i = i + 1

fim_para
se IND > 0
    RESULTADO = " INCORRETA"
fim_se
escreva (RESULTADO)
fim

```

#### 1.1.4.2 Estrutura Fila

A Estrutura de Dados Fila também é conhecida como **FIFO**, por ser acrônimo de **First In, First Out**, isto é, o primeiro que entra é o primeiro que sai (PEPS).



A denominação Fila é por lembrar um enfileiramento, por exemplo, de pessoas para tomar vacina, da forma que toda fila de pessoas civilizadas deveria ser. Quem entra na fila ocupa a posição após o último que entrou e a saída deve ser na ordem de entrada. Em uma estrutura dinâmica a disposição na memória não obedecerá a ordem sequencial, mas a ordem é a mesma, como se as pessoas pegassem uma senha e se espalhassem aguardando a sua vez.

Um exemplo de aplicação é o controle de documentos para impressão, onde os documentos são impressos na ordem de chegada.

Como a entrada de dados em uma estrutura Fila não corresponde ao mesmo lado da saída, a estrutura precisa conhecer o endereço do nó que está há mais tempo e também do último nó incluído.

A Figura abaixo, retirada de livro disponível na Biblioteca Virtual, representa a estrutura de dados Fila.

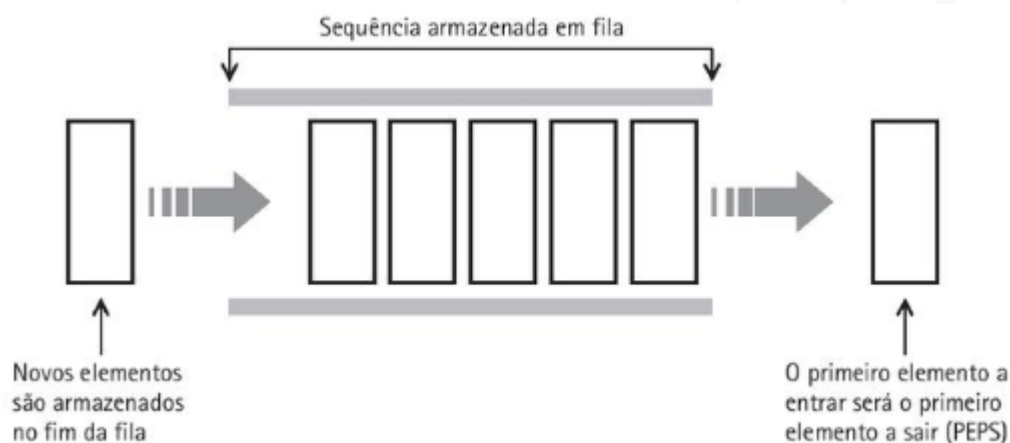


FIGURA - retirada do livro Lógica de Programação e Estrutura de Dados de Sandra Puga e Gerson Rissetti

### 1.1.5 Simulado

#### EXERCÍCIO 1.1.5-A

(Questão retirada de Prova do Enade) Considerando a execução do algoritmo abaixo, responda ao que se pede nos itens a e b.

```

01 algoritmo Vetores
02 variáveis
03     vetA[1..10], vetB[1..10], i: inteiro
04 início
05     para i de 1 até 10 faça
06         vetB[i] = 0
07         se resto(i,2) = 0
08             vetA[i] = i
09         senão
10             vetA[i] = 2 * i
11         fim_se
12     fim_para
13     para i de 1 até 10 faça
14         enquanto vetA[i] > i faça
15             vetB[i] = vetA[i]
16             vetA[i] = vetA[i] - 1
17         fim_enquanto
18     fim_para
19 fim_algoritmo

```

- a) Apresente os dados dos vetores vetA e vetB ao término da execução da linha 12.
- b) Apresente os dados dos vetores vetA e vetB ao término da execução da linha 19.

**Ao final da execução da linha 12:**

Vetor A										
Posição	1	2	3	4	5	6	7	8	9	10
Valor										

Vetor B										
Posição	1	2	3	4	5	6	7	8	9	10
Valor										

**Ao final da execução da linha 19:**

Vetor A										
Posição	1	2	3	4	5	6	7	8	9	10
Valor										

Vetor B										
Posição	1	2	3	4	5	6	7	8	9	10
Valor										

### EXERCÍCIO 1.1.5-B

(Questão retirada de Prova do Enade) No processo de pesquisa binária em um vetor ordenado, os números máximos de comparações necessárias para se determinar se um



elemento faz parte de vetores com tamanhos 50, 1.000 e 300 são, respectivamente, iguais a

- a) 5, 100 e 30.
- b) 6, 10 e 9.
- c) 8, 31 e 18.
- d) 10, 100 e 30.
- e) 25, 500 e 150.

### EXERCÍCIO 1.1.5-C

(Questão retirada de Prova Prefeitura Municipal de Dorés do Indaiá - Técnico em Informática - 2021)

A matriz em algoritmos é uma variável composta homogênea multidimensional. Ela é formada por uma sequência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome), e alocadas sequencialmente na memória.

Uma variável do tipo matriz precisa de:

- a) Um índice para cada uma de suas dimensões.
- b) Várias variáveis do mesmo nome.
- c) Uma variável composta em cada dimensão criada.
- d) Pelo menos duas variáveis com o mesmo número em cada dimensão criada.

## 2. Estruturas Dinâmicas Lineares

### 2.1 Definição

Como vimos, em uma estrutura de dados estática, implementada em um Array (Vetor ou Matriz), a disposição dos dados na memória é dada de forma contínua e o endereço de memória de um determinado elemento é especificado por um ou mais índices, do array. A quantidade de dados e a localização na memória são pré-determinados, o que pode em determinadas situações ocasionar um “estouro” da estrutura.

Por exemplo, para armazenar dados de uma loja que tenha 30 filiais, se for criado um array de 30 posições será suficiente, mas se um dia for criada mais uma filial, o programa tem que ser alterado, redimensionando o tamanho do vetor. Podemos então prever o crescimento e definir o vetor, digamos com 50 ocorrências. Isso vai gerar uma alocação ociosa de memória e não deixamos de correr o risco de um dia ser inaugurada a 51ª loja e o array não ser suficiente.

Já, as **estruturas dinâmicas** não apresentam esse problema, pois a quantidade de memória é alocada à medida que novos elementos (nós) forem incluídos e liberada quando o nó for excluído da estrutura. A memória não é alocada previamente e não necessariamente os dados estarão dispostos fisicamente de forma contínua.

Cada nó da estrutura dinâmica além do atributo para armazenar o dado, possui também um atributo do tipo ponteiro que “aponta” para o endereço do dado seguinte, formando um encadeamento entre os nós da estrutura. Lembrando que Ponteiro é um tipo de variável que armazena um endereço de memória.

Aqui, o índice não será um número inteiro e sim o endereço de memória indicado por um ponteiro. Toda estrutura dinâmica tem pelo menos um ponteiro para indicar o endereço do nó inicial e no caso de a estrutura estar vazia, o conteúdo desse ponteiro será NULL.

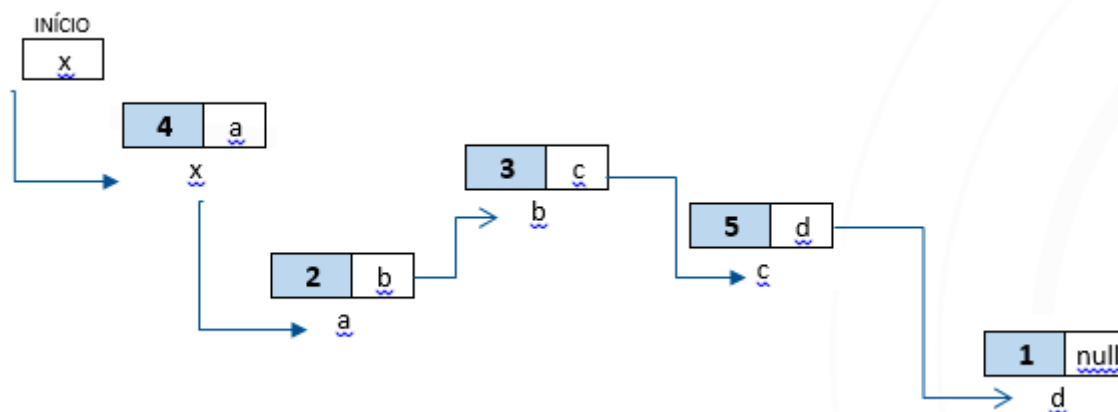
A Figura abaixo mostra um comparativo de um conjunto de dados representados em uma estrutura estática Vetor e em uma estrutura dinâmica, onde as letras minúsculas estão representando endereços aleatórios de memória.

## ESTRUTURA ESTÁTICA

VETOR [0..4]

4	2	3	5	1
---	---	---	---	---

## ESTRUTURA DINÂMICA



### 2.1.1 O Ponteiro *THIS*

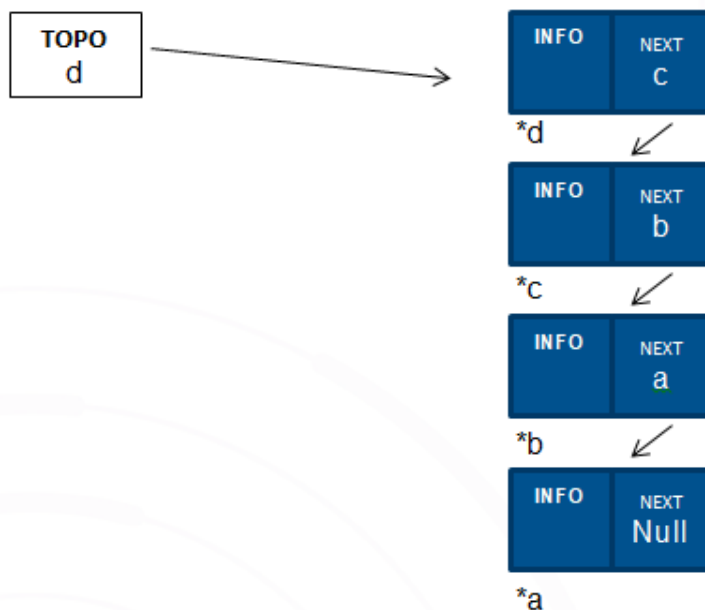
Ao ser criado um novo objeto que será inserido numa Pilha, será necessário conhecer o endereço de memória onde está armazenado este objeto. Esse endereço estará acessível através do ponteiro "**this**", que é uma das características dos objetos em C++ e algumas outras linguagens que suportam orientação a objetos. Ele é um membro inerente a todos os objetos que instanciamos em programas escritos em C++, não precisando ser definido com as demais variáveis.

## 2.2 Estrutura PILHA Dinâmica

### 2.2.1 Definição

Em uma Pilha Dinâmica cada nó contém dois atributos, um para armazenar o dado aqui denominado INFO e outro, do tipo Ponteiro, para armazenar o endereço de memória do nó seguinte, que será chamado NEXT. O ponto de vista é a partir do Topo, isto é, quando o nó do topo da pilha sair, qual será o próximo topo. Com isso o primeiro nó incluído na Pilha estará na base e seu campo NEXT será NULL.

A classe Pilha terá um atributo do tipo Ponteiro normalmente denominado TOPO, que apontará para o endereço do último objeto inserido na Pilha. Quando a estrutura estiver vazia, isto é, sem dados armazenados, o conteúdo de TOPO será NULL.



### 2.2.2 Construção de uma Pilha Dinâmica

Ao ser criada a classe Pilha, o ponteiro TOPO será inicializado com NULL, indicando que a Pilha está vazia.

**Cria Pilha**  
**TOPO = Null**

### 2.2.3 Inserção numa Pilha Dinâmica (PUSH)

A função de “empilhamento”, também denominada *Push*, ficará responsável pelo encadeamento desse novo objeto fazendo com que a Pilha o reconheça como um novo integrante dessa estrutura. A Classe, através do seu ponteiro Topo, indicará seu endereço de memória e caberá ao novo integrante indicar o endereço do elemento que até então era o topo da pilha.

Assim, após ter sido gerado um novo objeto, o ponteiro THIS automaticamente conterá o endereço ocupado. O trecho correspondente à inserção pode ser representado como:

```
cria objeto
escreva ("Informe o valor a ser inserido na Pilha ")
leia (INFO[THIS])
NEXT[THIS] = TOPO
TOPO = THIS
```

#### 2.2.4 Exclusão numa Pilha Dinâmica (POP)

A função de “desempilhamento”, também denominada *Push*, ficará responsável não só por retirar o objeto da Pilha, mas também por disponibilizar o endereço por ele ocupado para que o sistema operacional possa utilizá-lo para outra finalidade. Essa é uma grande vantagem das estruturas dinâmicas, que ocupam apenas a memória necessária para os dados armazenados no momento.

Como a regra da pilha é a retirada necessariamente do último que entrou (LIFO), será retirado o elemento que está armazenado no topo, não sendo necessário solicitar qual o valor a ser retirado da pilha. A única preocupação ao ser acionada a função POP é que a Pilha não esteja vazia. Vejamos o trecho do pseudocódigo correspondente ao desempilhamento:

```
se TOPO = Null
    escreva ("ARQUIVO VAZIO")
senão
    AUX = TOPO
    TOPO = NEXT[TOPO]
    deleta[AUX]
fim_se
```

#### 2.2.5 Listagem do conteúdo de uma Pilha Dinâmica

Para listar o conteúdo armazenado em uma Pilha, só é apresentado o valor do atributo INFO de cada nó, pois os ponteiros TOPO e NEXT servem apenas para o encadeamento desses nós, não sendo apresentados na listagem.

O acesso se dará do Topo até a base da pilha, como veremos no trecho do pseudocódigo correspondente:

```
se TOPO = NULL
    escreva ("ARQUIVO VAZIO")
senão
    AUX = TOPO
```

```

    enquanto AUX <> Null faça
        escreva (INFO[AUX])
        AUX = NEXT[AUX]
    fim_enquanto
fim_se

```

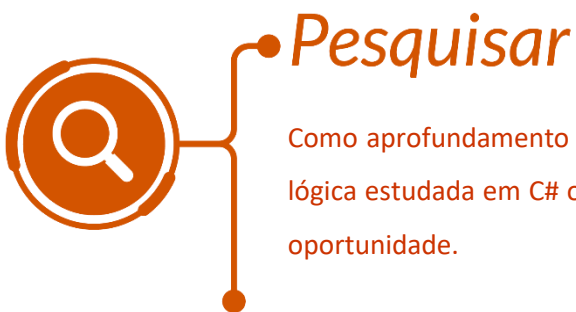
### 2.2.6 Quantidade de elementos de uma Pilha Dinâmica

Se ao invés de listar o conteúdo armazenado em uma Pilha, quisermos apenas conhecer a quantidade de nós constantes na estrutura, teremos que percorrer toda a pilha, do topo até a base, contando e ao final apresentaremos o total obtido, como veremos no trecho do pseudocódigo correspondente:

```

CONT = 0
AUX = TOPO
enquanto AUX <> NULO faça
    CONT = CONT + 1
    AUX = NEXT[AUX]
fim_enquanto
escreva ("Quantidade de elementos da Pilha = " CONT)

```



Como aprofundamento do estudo, se você tem condições de implementar a lógica estudada em C# ou em outra linguagem que domine, é uma excelente oportunidade.

### 2.2.7 Mãos à Obra

#### EXERCÍCIO 2.2.7-A

Com base no Algoritmo ChecaExpressãoEstática apresentado no exemplo de utilização de Pilha Estática, alterar para utilização de uma Pilha Dinâmica. A funcionalidade será a mesma, mudando apenas a estrutura de dados não mais utilizando o array.

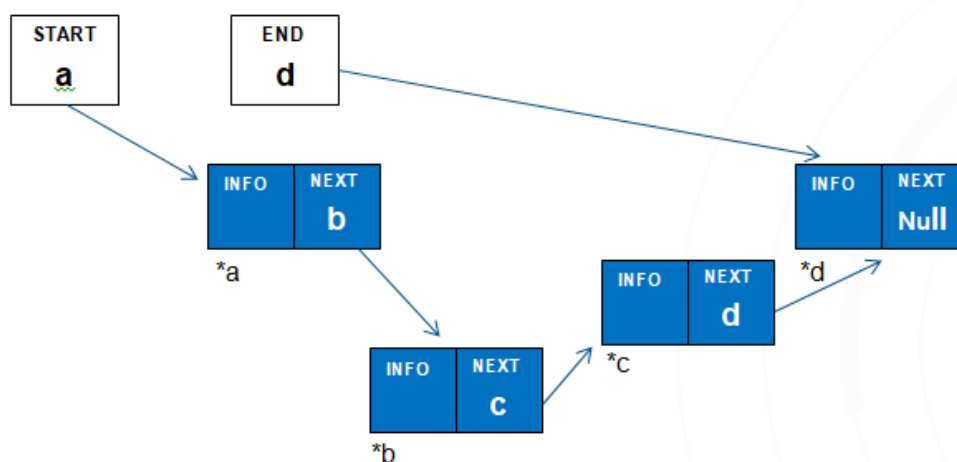
## 2.3 Estrutura FILA Dinâmica

### 2.3.1 Definição

Em uma estrutura dinâmica a disposição na memória não obedecerá a ordem sequencial, mas a ordem é a mesma, como se as pessoas pegassem uma senha e se espalhassem aguardando a sua vez.

Aqui a estrutura precisa de dois atributos do tipo Ponteiro. Um para armazenar o endereço do primeiro a sair que aqui será denominado START e outro para indicar o endereço do último que entrou, que chamaremos de END.

Assim como na Pilha, em uma Fila Dinâmica cada nó contém dois atributos, um para armazenar o dado (INFO) e outro, do tipo Ponteiro, para armazenar o endereço de memória do nó seguinte (NEXT). O ponto de vista é a partir do início. Com isso o nó que está no endereço apontado pelo END tem o campo NEXT com NULL.



### 2.3.2 Construção de uma Fila Dinâmica

Ao ser criada a classe Fila, os ponteiros START e END serão inicializados com NULL, indicando que a Fila está vazia.

**Cria Fila**  
**START = Null**  
**END = Null**

### 2.3.3 Inserção numa Fila Dinâmica

A função de “enfileiramento” ficará responsável pelo encadeamento desse novo objeto fazendo com que a Fila o reconheça como um novo integrante dessa estrutura.

Como a regra de ingresso numa Fila é sempre após o último, será utilizado o ponteiro END para ser acessado o nó que está no fim da fila e alterar o seu atributo NEXT, que até então estava com NULL para passar a apontar para o novo integrante da Fila. O endereço desse novo integrante é encontrado no THIS, como visto na inserção em uma Pilha.



Além do encadeamento através do NEXT do nó que até então era o último, o Ponteiro END da classe também precisa ser atualizado, para que em uma próxima inserção o encadeamento seja feito no novo último da fila. Repare que em uma Fila populada, o Ponteiro START não sofre alteração, pois a movimentação ocorre apenas no final da fila. Porém, se a Fila estiver vazia, o novo integrante além de ser o último da fila, também será o primeiro e nesse caso a Classe precisa reconhecê-lo também através do START. O trecho correspondente à inserção pode ser representado como:

```

Cria objeto
escreva ("Informe o valor a ser inserido na Fila ")
leia (INFO[THIS])
NEXT[THIS] = Null
se START = Null      // indica que a fila estava vazia
    START = THIS
senão
    NEXT[END] = THIS
fim_se
END = THIS

```

#### 2.3.4 Exclusão em uma Fila Dinâmica

A função de exclusão de uma Fila ficará responsável não só por retirar o objeto da Fila, mas também por disponibilizar o endereço por ele ocupado para que o sistema operacional possa utilizá-lo para outra finalidade, assim como ocorre no desempilhamento.

Como uma Fila só permite a saída do primeiro, será utilizado o ponteiro START para ser acessado o nó que está no início da fila. Antes de deletar esse nó, precisamos do endereço que está em seu NEXT para alterar o conteúdo de START.

Caso o nó que está sendo retirado seja o único existente na estrutura, a Fila ficará vazia. O ponteiro Start receberá o Null, que é o conteúdo de NEXT do nó retirado, mas precisamos manter a coerência do conteúdo de End, colocando Null nesse ponteiro também. Vejamos o trecho do pseudocódigo correspondente ao desempilhamento:

```

se START = NULL
    escreva ("ARQUIVO VAZIO")
senão
    AUX = START
    START = NEXT[START]
    se START = Null      // indica que a fila ficará vazia
        END = Null
    fim_se
    deleta[AUX]
fim_se

```

### 2.3.5 Listagem do conteúdo de uma Fila Dinâmica

Para listar o conteúdo armazenado em uma Fila, o acesso se dará do START até END, como veremos no trecho do pseudocódigo correspondente:

```
se START = NULL
    escreva ("ARQUIVO VAZIO")
senão
    AUX = START
    enquanto AUX <> Null faça
        escreva (INFO[AUX])
        AUX = NEXT[AUX]
    fim_enquanto
fim_se
```

### 2.3.6 Mãos à Obra

#### EXERCÍCIO 2.3.6-A

Com base nos exemplos dados e nos exercícios anteriores, elaborar o pseudocódigo de um programa que mantenha uma Fila Dinâmica que armazenará os nomes de clientes de uma clínica ainda não atendidos, na ordem de chegada, com as Opções de Inclusão, Remoção e Listagem.

No início apresentar o menu abaixo:

OPÇÕES:

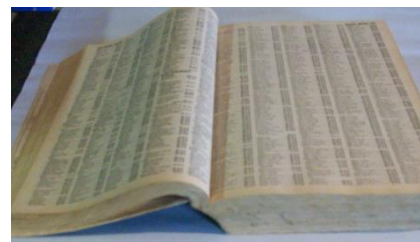
- (1) INCLUIR CLIENTE
- (2) EXCLUIR CLIENTE
- (3) LISTAR CLIENTES

O que deseja fazer? \_\_\_\_

## 2.4 Estrutura LISTA ENCADEADA

### 2.4.1 Definição

As Estruturas de Dados Pilha e Fila, vistas até agora, são tipos de Listas com restrições na entrada e saída. Mas imagine uma Lista Telefônica ou uma Lista de Presença de uma aula em ordem alfabética que pode ser alterada para incluir um novo aluno ou retirar algum aluno da lista.



Logicamente não poderemos restringir que só entre no final da lista ou só saia o primeiro. A estrutura denominada Lista não tem restrições como as demais e quando implementadas de forma dinâmica são chamadas de Listas Encadeadas, pois teremos o mesmo tipo de encadeamento visto nas anteriores utilizando o atributo NEXT.

As alterações em uma estrutura Lista têm um comportamento muito semelhante ao da Fila, mas agora temos que identificar o posicionamento do elemento em questão, pois uma inclusão pode ser no final, como na Fila, mas pode ser no início ou entre dois elementos já constantes na Lista. A retirada também não fica restrita ao primeiro. A implementação em uma estrutura estática, implica no deslocamento dos demais elementos do array, para abrir o espaço no local adequado para inclusão ou para o preenchimento do local onde se encontrava o elemento eliminado.

A estrutura LISTA utiliza os controles START e END de forma similar à FILA e seus objetos têm estrutura semelhante, com o campo INFO e o NEXT do tipo Ponteiro. Quando a estrutura estiver vazia, o conteúdo de START e de END será NULL e o nó que está no endereço apontado pelo END também tem o campo NEXT com conteúdo NULL.

### 2.4.2 Inserção numa Lista Encadeada

A função de Inserção ficará responsável pela identificação do posicionamento adequado para a inclusão e o encadeamento desse novo objeto fazendo com que a Lista o reconheça como um novo integrante dessa estrutura.

Vamos considerar uma Lista com valores em ordem crescente. Inicialmente deve ser identificado o posicionamento e chamar o módulo para inclusão na posição identificada. Se a Lista estiver vazia, ao incluir um primeiro elemento, devem ser atualizados os dois

ponteiros da Lista, o START e o END. Como a Lista não tem restrição na inclusão, se a Lista não estiver vazia, devem ser previstas as diferentes situações para que o conteúdo da Lista permaneça em ordem crescente, como veremos no pseudocódigo a seguir:

#### INCLUIR-PRIMEIRO-LISTA

```
NEXT[THIS] = Null  
START = THIS  
END = THIS
```

#### INCLUIR-NO-INÍCIO

```
NEXT[THIS] = START  
START = THIS
```

#### INCLUIR-NO-FINAL

```
NEXT[THIS] = Null  
NEXT[END] = THIS  
END = THIS
```

#### INCLUIR-NO-MEIO

```
AUX = START  
enquanto INFO[THIS] > INFO[NEXT[AUX]] faça  
    AUX = NEXT[AUX]  
fim_enquanto  
NEXT[THIS] = NEXT[AUX]  
NEXT[AUX] = THIS
```



## Entendendo

O pseudocódigo foi desenvolvido com chamadas de módulos para facilitar a visualização das rotinas de forma independente. Naturalmente o bloco de comandos podia estar no local da chamada correspondente, até mesmo por serem módulos muito simples, mas a técnica de modularização é altamente recomendável para facilitar o entendimento do programa.

Observe que no módulo INCLUIR-NO-MEIO optamos por comparar o valor a ser incluído com o INFO do nó seguinte ao AUX para interrompermos o loop no nó anterior, isto é, com valor imediatamente inferior ao que será incluído, pois é nele que será feito o encadeamento para o novo elemento.

Compare os três comandos existentes no módulo INCLUIR-NO-FINAL com a Inclusão da Fila. São os mesmos que serão executados quando a Fila não estiver vazia.

Nada mais lógico, já que na Fila a inclusão só ocorre no final. A inclusão numa Fila e Lista vazias, o comportamento também é semelhante.

### 2.4.3 Exclusão em uma Lista Encadeada

Em comparação com a Fila, a diferença é a não restrição em sair apenas o primeiro. Pode ser retirado qualquer elemento da Lista. Então, temos que receber a informação de qual elemento deve ser retirado e identificar seu posicionamento, eliminá-lo e manter o encadeamento entre os nós de forma coerente com a sua saída.

O trecho do pseudocódigo a seguir segue o mesmo conceito de modularização visto na inclusão:

```

se START = Null                                     // indica que a lista está vazia
    escreva ("ARQUIVO VAZIO")
senão
    escreva ("Informe o valor a ser excluído da Lista")
    leia (EXCL)
    se EXCL = INFO[START]
        chama <EXCLUIR-PRIMEIRO-LISTA>
    senão
        se EXCL = INFO[END]
            chama <EXCLUIR-FINAL-LISTA>
        senão
            chama <EXCLUIR-MEIO-LISTA>
        fim_se
    fim_se
fim_se

```

#### EXCLUIR-PRIMEIRO-LISTA

```

AUX = START
START = NEXT[START]
deleta[AUX]
se START = Null                                     // indica que a fila ficará vazia
    END = Null
fim_se

```

#### EXCLUIR-FINAL-LISTA

```

AUX = START
enquanto NEXT[AUX] <> END faça                       // localizar o penúltimo
    AUX = NEXT[AUX]
fim_enquanto
NEXT[AUX] = Null
deleta[END]
END = AUX

```

**EXCLUIR-MEIO-LISTA**

```
AUX = START
enquanto EXCL <> INFO[NEXT[AUX]] faça
    AUX = NEXT[AUX]
fim_enquanto
AUX2 = NEXT[AUX]
NEXT[AUX] = NEXT[NEXT[AUX]]
deleta[AUX2]
```

**2.4.4 Listagem do conteúdo de uma Lista Encadeada**

Para listar o conteúdo armazenado em uma Lista, o comportamento é exatamente o mesmo da listagem da Fila, iniciando do Start até o End:

```
se START = NULL
    escreva ("ARQUIVO VAZIO")
senão
    AUX = START
    enquanto AUX <> Null faça
        escreva (INFO[AUX])
        AUX = NEXT[AUX]
    fim_enquanto
fim_se
```

**2.4.5 Mãos à Obra****EXERCÍCIO 2.4.5-A**

Com base nos exemplos dados e nos exercícios anteriores, elaborar o pseudocódigo de um programa que mantenha uma Lista Encadeada que armazenará os nomes de alunos matriculados em uma turma em ordem alfabética, com as Opções de Inclusão, Remoção e Listagem.

No início apresentar o menu abaixo:

OPÇÕES:

- (1) INCLUIR UM ALUNO
- (2) EXCLUIR UM ALUNO
- (3) EMITIR LISTAGEM DE ALUNOS

O que deseja fazer? \_\_\_\_

## 2.5 Estrutura LISTA DUPLAMENTE ENCADEADA

### 2.5.1 Definição

Analisando o pseudocódigo da Lista Encadeada, vimos que na exclusão do último nó da Lista, mesmo tendo o seu endereço no ponteiro END, tivemos que fazer uma Busca Sequencial para localizar o penúltimo desde o START, para podermos alterar o campo NEXT do penúltimo para Null e alterar o END.

Na estrutura que veremos agora essa situação será simplificada, pois além do NEXT que faz o encadeamento para o próximo da Lista, teremos um outro atributo em cada objeto para fazer o encadeamento para o anterior. Chamaremos esse novo ponteiro de PREV.

Veremos que nos pseudocódigos correspondentes, apesar do cuidado com a manutenção do duplo encadeamento, teremos algumas facilidades além do caso citado da exclusão do último. Na inclusão no meio não precisaremos comparar com o INFO do posterior ao AUX, pois agora se pararmos o AUX no imediatamente maior que o novo elemento, teremos como voltar ao endereço anterior usando o PREV de AUX. Também teremos a flexibilidade de listar o conteúdo da Lista tanto na ordem crescente quanto na decrescente, o que não seria tão simples na Lista de encadeamento simples.

### 2.5.2 Inserção numa Lista Duplamente Encadeada

Em comparação ao que vimos na Lista de encadeamento simples, na inserção de dados numa Lista Duplamente Encadeada, acrescentamos a manutenção do encadeamento no sentido inverso através dos atributos PREV de cada objeto.

Acredito que esses comentários sejam mais facilmente observados entendendo e comparando os pseudocódigos a seguir.

```
escreva ("Informe o valor a ser inserido na Lista")
leia (INFO[THIS])
se START = Null                                     // indica que a fila está vazia
    chama <INCLUIR-PRIMEIRO-DUPLA>
senão
    se INFO[THIS] < INFO[START]
        chama <INCLUIR-NO-INÍCIO-DUPLA>
    senão
        se INFO[THIS] > INFO[END]
```



```

        chama <INCLUIR-NO-FINAL-DUPLA>
      senão
        chama <INCLUIR-NO-MEIO-DUPLA>
    fim_se
  fim_se
fim_se

```

#### INCLUIR-PRIMEIRO-DUPLA

```

NEXT[THIS] = Null
PREV[THIS] = Null
START = THIS
END = THIS

```

#### INCLUIR-NO-INÍCIO-DUPLA

```

PREV[THIS] = Null
PREV[START] = THIS
NEXT[THIS] = START
START = THIS

```

#### INCLUIR-NO-FINAL-DUPLA

```

PREV[THIS] = END
NEXT[THIS] = Null
NEXT[END] = THIS
END = THIS

```

#### INCLUIR-NO-MEIO

```

AUX = START
enquanto INFO[THIS] > INFO[AUX] faça
    AUX = NEXT[AUX]
fim_enquanto
NEXT[THIS] = AUX
PREV[THIS] = PREV[AUX]
NEXT[PREV[AUX]] = THIS
PREV[AUX] = THIS

```

Opcionalmente poderíamos manter a lógica utilizada no módulo equivalente para a Lista de encadeamento simples, com as adaptações correspondentes:

```

AUX = START
enquanto INFO[THIS] > INFO[NEXT[AUX]] faça
    AUX = NEXT[AUX]
fim_enquanto
NEXT[THIS] = NEXT[AUX]
PREV[THIS] = AUX
NEXT[AUX] = THIS
PREV[NEXT[THIS]] = THIS

```



### 2.5.3 Exclusão em uma Lista Duplamente Encadeada

Na exclusão de um dado da Lista Duplamente Encadeada manteremos os mesmos cuidados na manutenção da integridade do encadeamento nos dois sentidos. Também, dependendo da localização do elemento a ser removido, teremos alguns cuidados específicos a tomar para manter a integridade do encadeamento.

#### EXCLUIR-PRIMEIRO-DUPLA

```
AUX = START
START = NEXT[START]
PREV[START] = Null
deleta[AUX]
se START = Null      // indica que a lista ficará vazia
    END = Null
fim_se
```

#### EXCLUIR-FINAL-DUPLA

```
NEXT[AUX] = Null
AUX = PREV[END]
deleta[END]
END = AUX
```

#### EXCLUIR-MEIO-LISTA

```
AUX = START
enquanto EXCL <> INFO[AUX] faça
    AUX = NEXT[AUX]
fim_enquanto
PREV[NEXT[AUX]] = PREV[AUX]
NEXT[PREV[AUX]] = NEXT[AUX]
deleta[AUX]
```

### 2.5.4 Listagem do conteúdo de uma Lista Duplamente Encadeada

Na Lista Duplamente Encadeada temos a facilidade de apresentar o conteúdo na ordem crescente ou decrescente. Na ordem crescente não há alteração em relação à Lista encadeada simplesmente. Podemos, portanto, dar a opção no momento da implementação da listagem.

```
se START = NULL
    escreva ("ARQUIVO VAZIO")
senão
    escreva ("Informe o valor a ordem de listagem (CRESCENTE / DECRESCENTE)")
    leia (ORDEM)
    se ORDEM = "CRESCENTE"
        chama <LISTA-ORDEM-CRESCENTE>
```

```
senão
    se ORDEM = "DECRESCENTE"
        chama <LISTA-ORDEM-DECRESCENTE>
    senão
        escreva ("ORDEM INVÁLIDA")
    fim_se
fim_se
```

#### LISTA-ORDEM-CRESCENTE

```
AUX = START
enquanto AUX <> Null faça
    escreva (INFO[AUX])
    AUX = NEXT[AUX]
fim_enquanto
```

#### LISTA-ORDEM-DECRESCENTE

```
AUX = END
enquanto AUX <> Null faça
    escreva (INFO[AUX])
    AUX = PREV[AUX]
fim_enquanto
```

### 2.5.5 Mãos à Obra

#### EXERCÍCIO 2.5.5-A

Adaptar o Exercício 2.4.5-A para utilizar uma Lista Duplamente Encadeada implementando na Opção de Emissão de Listagem, se deseja a listagem na ordem Alfabética Crescente ou Decrescente.

No início apresentar o menu abaixo:

#### OPÇÕES:

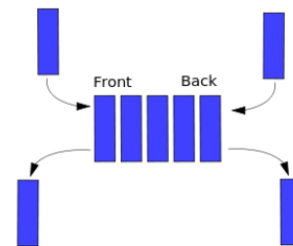
- (1) INCLUIR UM ALUNO
- (2) EXCLUIR UM ALUNO
- (3) EMITIR LISTAGEM DE ALUNOS (ordem crescente)
- (4) EMITIR LISTAGEM DE ALUNOS (ordem decrescente)

O que deseja fazer? \_\_\_\_

## 2.6 Variações da Estrutura LISTA

### 2.6.1 Estrutura DEQUE

O Deque é uma implementação de Lista com restrição semelhante a uma Fila, porém com entrada e saída de dados permitidos nas suas duas extremidades. Um novo nó pode ser inserido na frente do primeiro ou após o último, mas nunca no meio. A exclusão de um nó se dá nas mesmas condições.

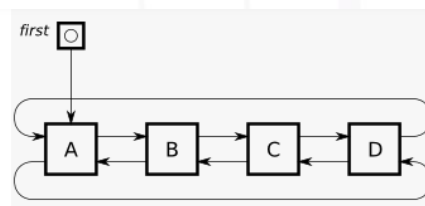


Sua denominação se dá através do termo em inglês Double Ended Queue, ou seja, Fila de Dupla Extremidade.

Os pseudocódigos são os mesmos utilizados na Lista, exceto os que permitem alteração no meio, podendo ser implementado com encadeamento simples ou duplo.

### 2.6.2 Estrutura LISTA CIRCULAR

Uma possível implementação de Listas de encadeamento simples ou duplo, de acordo com a utilização, são as chamadas Listas Circulares.



Para sua implementação basta ao invés de colocar Null no NEXT[END], colocar o endereço de START e no caso da Lista Circular Duplamente Encadeada também trocar o Null de PREV[START] pelo endereço de END.

## 2.7 Simulado

### EXERCÍCIO 2.7-A

(Questão retirada de Prova IF-SP - 2019 - IF-SP – Informática) Abaixo tem-se uma tabela que ilustra o conjunto de nós de uma lista duplamente encadeada, contendo o total de 5 nós.

elemento	anterior	proximo	conteúdo
1	3	4	?
2		5	?
3	5	1	?
4	1		?
5	2	3	?

Ao imprimir a estrutura na ordem correta, o conteúdo apresentado será I – F – S – P – 2019, dessa forma, assinale a alternativa que contém os dados que preenchem, corretamente, a coluna “conteúdo”, de cima para baixo.

- a) P – I – S – 2019 – F
- b) 2019 – P – S – F – I
- c) S – P – I – F – 2019
- d) S – I – F – 2019 – P

### EXERCÍCIO 2.7-B

(Questão retirada de Prova IFRJ - Analista de Tecnologia da Informação – 2021)

Analise as afirmações a seguir a respeito de pilhas:

I - Novos elementos entram, no conjunto, exclusivamente, no topo da pilha.

II - O único elemento que pode sair da pilha em um dado momento, é o elemento do topo.

III - as Pilhas são conhecidas como LIFO (last in, first out), isto é, o último a entrar é o último a sair.

Estão corretas as afirmações:

- a) I e II.
- b) I e III.
- c) II e III.
- d) I, II e III.

## EXERCÍCIO 2.7-C

(Questão retirada de Prova do Enade)

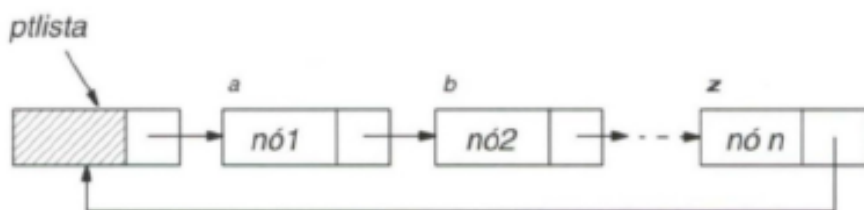
O coordenador geral de um comitê olímpico solicitou a implementação de um aplicativo que permita o registro dos recordes dos atletas à medida que forem sendo quebrados, mantendo a ordem cronológica dos acontecimentos, e possibilitando a leitura dos dados a partir dos mais recentes.

Considerando os requisitos do aplicativo, a estrutura de dados mais adequada para a solução a ser implementada é

- A** o deque: tipo especial de lista encadeada, que permite a inserção e a remoção em qualquer das duas extremidades da fila e que deve possuir um nó com a informação (recorde) e dois apontadores, respectivamente, para os nós próximo e anterior.
- B** a fila: tipo especial de lista encadeada, tal que o primeiro objeto a ser inserido na fila é o primeiro a ser lido; nesse mecanismo, conhecido como estrutura FIFO (*First In – First Out*), a inserção e a remoção são feitas em extremidades contrárias e a estrutura deve possuir um nó com a informação (recorde) e um apontador, respectivamente, para o próximo nó.
- C** a pilha: tipo especial de lista encadeada, na qual o último objeto a ser inserido na fila é o primeiro a ser lido; nesse mecanismo, conhecido como estrutura LIFO (*Last In – First Out*), a inserção e a remoção são feitas na mesma extremidade e a estrutura deve possuir um nó com a informação (recorde) e um apontador para o próximo nó.
- D** a fila invertida: tipo especial de lista encadeada, tal que o primeiro objeto a ser inserido na fila é o primeiro a ser lido; nesse mecanismo, conhecido como estrutura FIFO (*First In – First Out*), a inserção e a remoção são feitas em extremidades contrárias e a estrutura deve possuir um nó com a informação (recorde) e um apontador, respectivamente, para o nó anterior.
- E** a lista circular: tipo especial de lista encadeada, na qual o último elemento tem como próximo o primeiro elemento da lista, formando um ciclo, não havendo diferença entre primeiro e último, e a estrutura deve possuir um nó com a informação (recorde) e um apontador, respectivamente, para o próximo nó.

## EXERCÍCIO 2.7-D

(Questão retirada de Prova 2016 - Prefeitura de Juiz de Fora - MG - Programador) Em programação, mais especificamente na parte que trata de estrutura de dados, existem os conceitos de listas encadeadas. A imagem a seguir representa um tipo de lista encadeada. Qual é o tipo dessa lista?



- a) Lista de encadeamento reverso.
- b) Lista de encadeamento duplo.
- c) Pilha.
- d) Árvore Binária.
- e) Lista circular.

### EXERCÍCIO 2.7-E

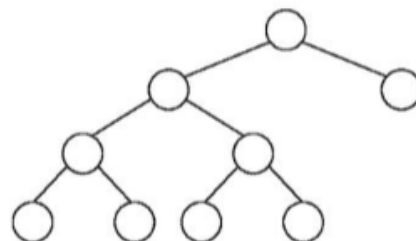
(Questão retirada de Prova 2012 - TJ-RJ - Analista Judiciário - Análise de Sistemas) O algoritmo conhecido como busca binária é um algoritmo de desempenho ótimo para encontrar a posição de um item em

- a) uma árvore B
- b) uma Lista encadeada ordenada
- c) uma árvore de busca binária
- d) um heap binário
- e) um vetor ordenado

### 3 ESTRUTURA DINÂMICAS NÃO LINEARES – ÁRVORES

#### 3.1 Definição

As estruturas de dados vistas até aqui são de organização **Linear**. O percurso nessas estruturas passa por uma trilha única com mão única ou dupla.



As estruturas Árvores são de organização **Não Linear**, permitindo caminhos diferentes como se numa estrada encontrássemos uma bifurcação ou uma rotatória com diversas saídas.

Inicialmente trataremos das denominadas Árvores Binárias. Essa denominação é em razão de, a cada passo dado na estrutura, termos no máximo duas opções de caminhos, mas a seguir veremos árvores n\_árias, que permitem diversas opções, a cada passo dado.

Um exemplo de aplicação que você está acostumado a usar é a árvore de diretórios ou pastas de arquivos do seu computador, como mostra a figura retirada do livro disponível na biblioteca virtual.



FIGURA - retirada do livro Lógica de Programação e Estrutura de Dados de Sandra Puga e Gerson Rissetti

Alguns termos utilizados no tratamento de estruturas de dados precisam ser conhecidos para um bom entendimento do que será estudado a seguir:

**Nó Raiz** – É o nó do topo da árvore, sendo o único que não tem antecessor. Seu endereço de memória é o único controlado pela estrutura. Todos os demais nós derivam do nó raiz;

**Nó Pai** – O nó pai é o que antecede outro nó;



**Nós Filhos** – São os sucessores imediatos de um determinado nó (Pai);

**Nós Irmãos** – São nós que possuem um mesmo pai;

**Nós Folhas** – São os nós que não possuem filhos;

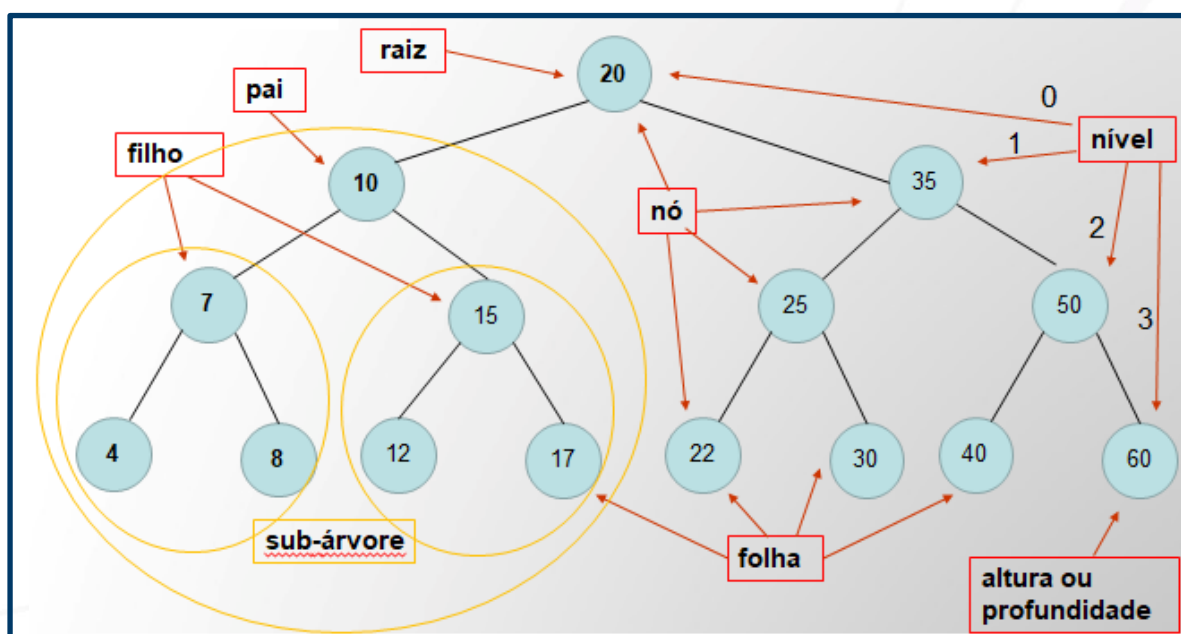
**Nível de um Nó** – É o número de passos necessários a partir do Nó Raiz até chegar no nó em questão. O nível do nó raiz é 0 (zero);

**Grau de um nó** – É a quantidade de Filhos do nó;

**Altura ou Profundidade** – A Altura de uma árvore, também denominada de Profundidade, é o maior número de nível de um nó constante da árvore;

**Grau de um nó** – É a quantidade de Filhos do nó;

**Sub-árvore** – É o conjunto de nós descendentes de qualquer nó da árvore, o qual é considerado a raiz da sua sub-árvore.

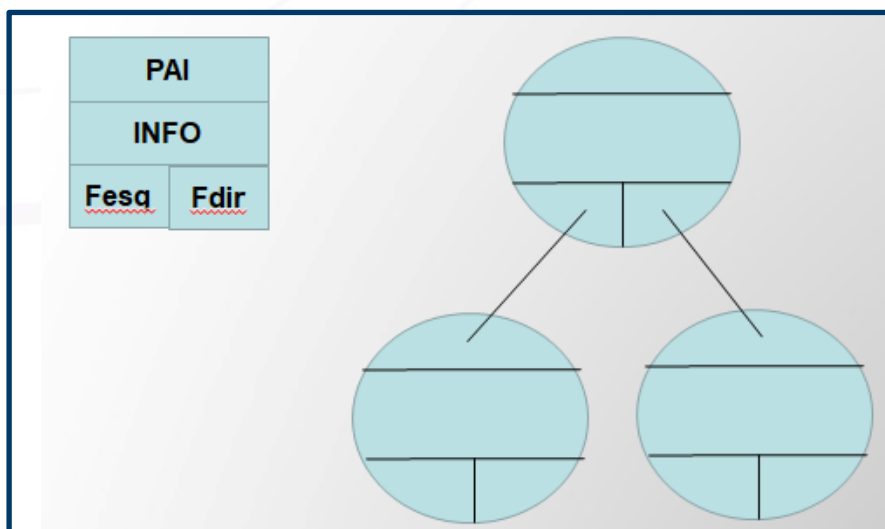


## 3.2 Estrutura ÁRVORE BINÁRIA

### 3.2.1 Definição

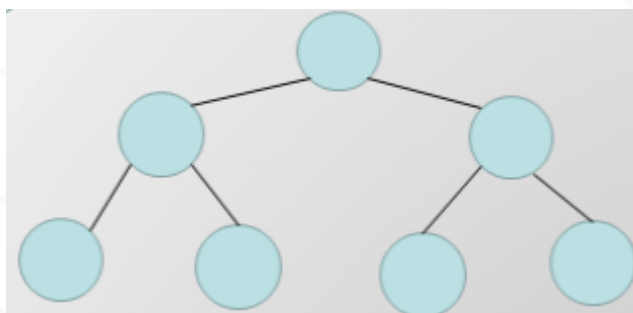
São estruturas não lineares onde cada nó possui no máximo dois filhos. Cada nó é definido com atributos, além do INFO para armazenar o dado, mais 3 ponteiros. O primeiro chamamos de PAI que apontará para o nó antecessor e como vimos, apenas o nó raiz estará com Null. Teremos também mais dois Ponteiros que indicam os endereços dos Filhos, que chamamos de FESQ (Filho da Esquerda) e FDIR (Filho da Direita).





Para simplificar a representação dos nós de uma árvore os nós comumente são representados por círculos e os ponteiros por traços interligando os nós, como visto na Figura anterior.

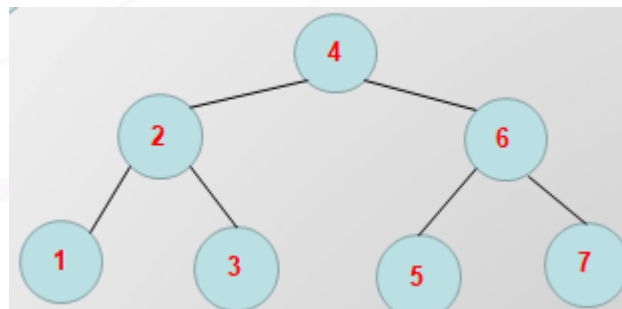
Para que uma Árvore Binária seja eficiente, seus dados precisam estar ordenados, pois só assim será possível a escolha do percurso para chegar a um determinado nó. Mas como ordenar o conteúdo dos nós de uma árvore binária, por exemplo, com o formato abaixo? Se os dados de cada nó forem os números de 1 a 7, qual o conteúdo de cada nó para que a árvore binária esteja em ordem crescente?



Pare. Pense um pouco sobre as questões acima. Qual a disposição dos números de 1 a 7 na árvore acima?

E aí? Pensou?

Chegou na solução abaixo?



Complicou? Como chegamos nessa solução?

Vamos com calma. A regra é: Cada nó tem que possuir a sub-árvore da esquerda apenas com valores menores do que ele e a sub-árvore da direita apenas com valores maiores. Veja o nó raiz. Sua sub-árvore da esquerda é composta por 2, 1 e 3 e sua sub-árvore da direita é composta por 6, 5 e 7. O nó com conteúdo 2 tem o 1 à esquerda e o 3 à direita. Isso precisa ocorrer com TODOS os nós da árvore. Isso vai ficar mais fácil de entender vendo o passo a passo da criação de uma Árvore Binária.

Essa ordenação caracteriza a **ABB - Árvore Binária de Busca**, também denominada BST – Binary Search Tree. Essa ordem precisa ser garantida no momento da inclusão de um novo nó.

### 3.2.2 Inclusão em Árvore Binária

Quando a árvore está vazia, ao ser incluído o primeiro elemento, este assumirá a raiz da árvore. Observe que nessa situação, o único nó constante na árvore será Raiz e Folha simultaneamente.

Todo nó quando incluído na árvore binária entra como uma Folha, isto é, os atributos FESQ e FDIR com Null. No momento da inclusão deve ser traçado o percurso partindo do nó raiz e seguindo para a esquerda ou direita de acordo com o valor do novo nó em comparação com os nós constantes na árvore. Podemos considerar como consistência a situação de o valor a ser incluído já existir na árvore, dando mensagem correspondente. O percurso terminará quando, seguindo o caminho de busca do valor a ser incluído, encontrarmos um nó apontamento com NULL na direção onde o novo valor estaria. Esse será o Pai do novo nó e para a inclusão, o endereço do Pai é atualizado no ponteiro

correspondente do novo nó (THIS) e o endereço THIS colocado no ponteiro FESQ ou FDIR dependendo se o INFO do THIS é maior ou menor que o INFO do Pai, mantendo a regra de ordenação.

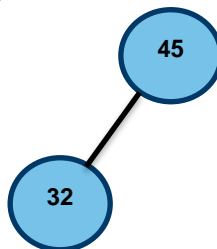
Vamos fazer passo a passo a simulação da inclusão dos valores seguintes, na ordem apresentada, partindo de uma estrutura vazia: 45, 32, 87, 55, 40, 82.

Passo 1 - Incluir o 45:



O valor 45 foi colocado em INFO[THIS] e os ponteiros PAI[THIS], FESQ[THIS] e FDIR[THIS] ficam com Null. O endereço THIS é colocado no ponteiro RAIZ da classe, que até então estava com Null.

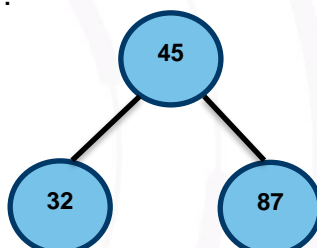
Passo 2 - Incluir o 32:



Ao ser criado um novo nó, seu endereço fica acessível no THIS. O valor 32 é colocado em INFO[THIS] e Null em FESQ e FDIR, pois todo novo nó entra na árvore como uma Folha.

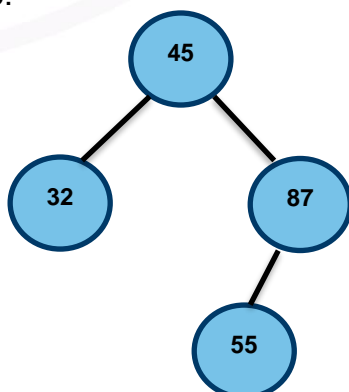
Comparamos o novo valor com o valor do nó raiz. Como é menor, o 32 deve ficar à esquerda do nó raiz, mas como o FESQ é Null, o novo nó será o Filho da esquerda do 45. Para tal efetua-se o encadeamento colocando o valor de THIS no FESQ do pai e o endereço do pai, que nesse caso é o nó raiz, no atributo PAI[INFO].

Passo 3 -: Incluir o 87:



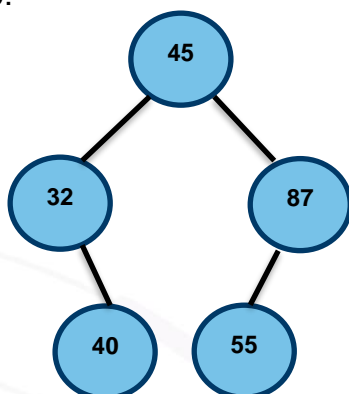
Na comparação do valor do novo nó com o valor do nó Raiz, definimos que o 87 será o Filho da Direita do 45, pois o nó raiz estava com FESQ = Null.

Passo 4 - Incluir o 55:



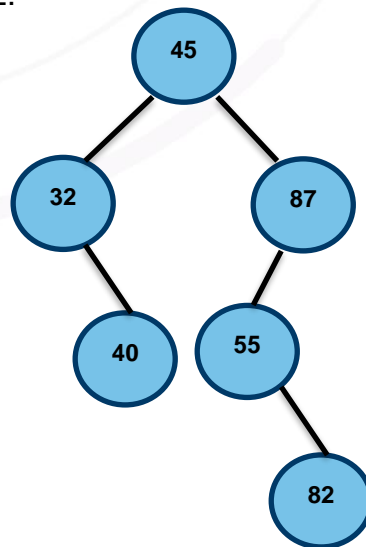
O nó com o valor 55 ficou como Filho da Esquerda do nó 47 após compararmos o novo valor com o 45 e por ser maior caminhamos para a direita. Como o FESQ do nó raiz estava preenchido com o endereço do nó 87, demos um passo para o nível 1 e comparamos o 55 com o 87, por ser menor e termos encontrado Null no FESQ do nó 87 o identificamos como PAI do novo elemento. Para implementar essa filiação é só fazer o encadeamento.

Passo 5 - Incluir o 40:



De forma semelhante ao passo anterior, foi feita a comparação do 40 com o 45 e dado um passo para o nível 1 seguindo o endereço do FESQ. Ao comparar o 40 com o 32, por ser maior e FDIR estar com Null ficou identificado que o novo nó será o Filho da Direita do 32.

Passo 6 - Incluir o 82:



Com essa inclusão, a árvore passou a ter altura 3, pois o 82 foi comparado com o 45 e por ser maior foi comparado com o 87. Por ser menor, foi comparado com o 55 e como é maior foi incluído no nível 3 da árvore, como Filho da Direita do 55.

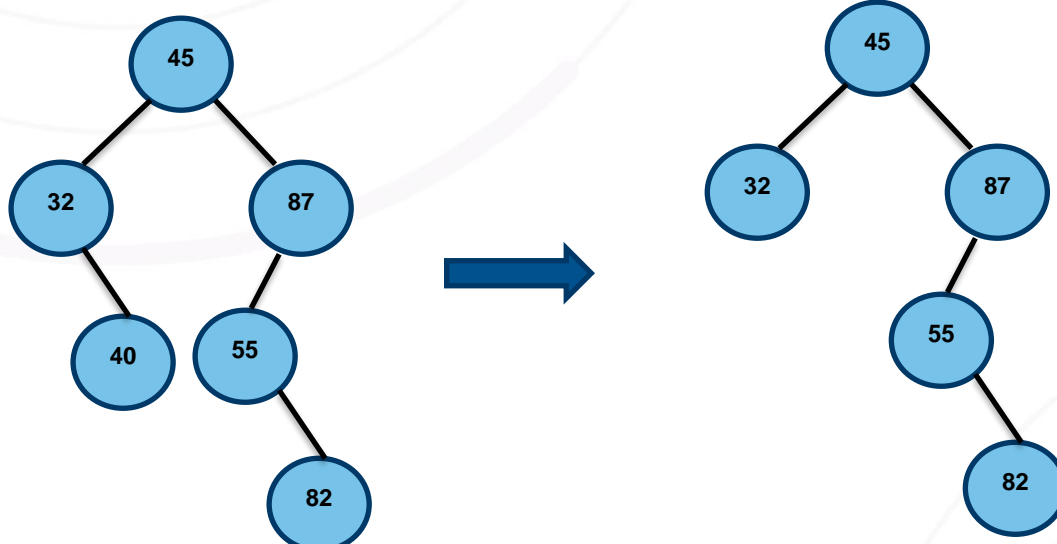
### 3.2.3 Deleção em Árvore Binária

A retirada de um nó de uma árvore binária pode ocorrer em 3 situações distintas que requerem tratamentos diferentes quanto ao nó a ser retirado:

- 1- O nó é uma folha;
- 2- O nó possui apenas um Filho;
- 3- O nó possui dois Filhos.

Na primeira situação, quando o nó não possuir filhos a situação é a mais simples. É só arrancar a folha, isto é, navegar para o nó Pai e colocar Null no Ponteiro Filho (FESQ ou FDIR) que apontava para o nó a ser deletado. Ao fazer isso, o nó deixa de pertencer à árvore, pois partindo da raiz, não há mais como chegar nesse nó. Porém, devemos deletar o nó Folha para liberar o espaço por ele ocupado.

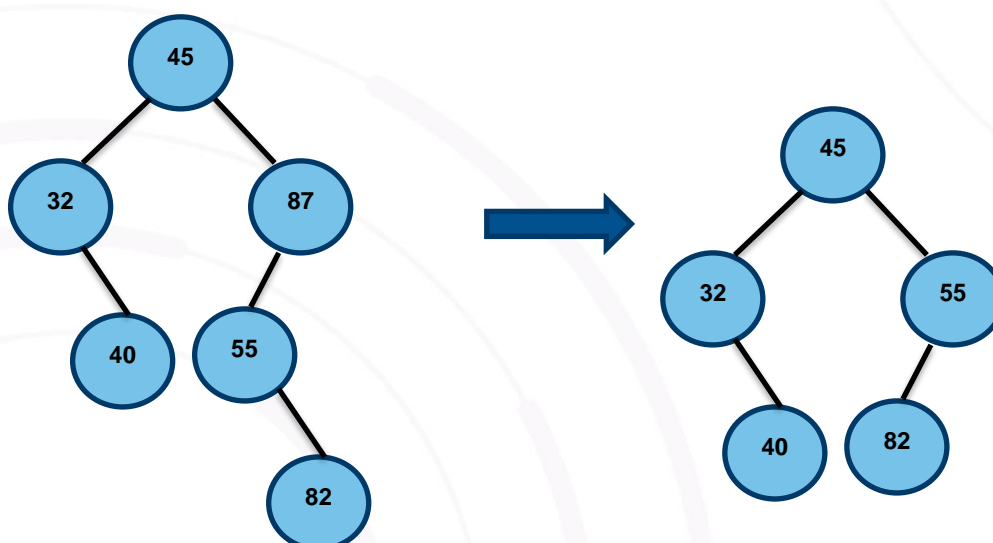
Na Figura abaixo, por exemplo, seria a deleção do 40:



Na segunda situação, quando o nó possui apenas um Filho, o Filho tomará o lugar do Pai, que será excluído. Nesse caso o Filho e toda a sub-árvore de descendentes dele diminuem um nível.

Para implementar essa deleção, o endereço do Filho é colocado no Pai do que será deletado e o endereço do Pai será colocado no Filho. Concluída a alteração do encadeamento, o nó que já ficou desencadeado da árvore deve ser deletado.

Na Figura abaixo, por exemplo, seria a deleção do 87:



Na terceira e última situação temos um nó a ser deletado que possui dois filhos. Naturalmente não daria para fazer o mesmo procedimento que fizemos nas situações acima.

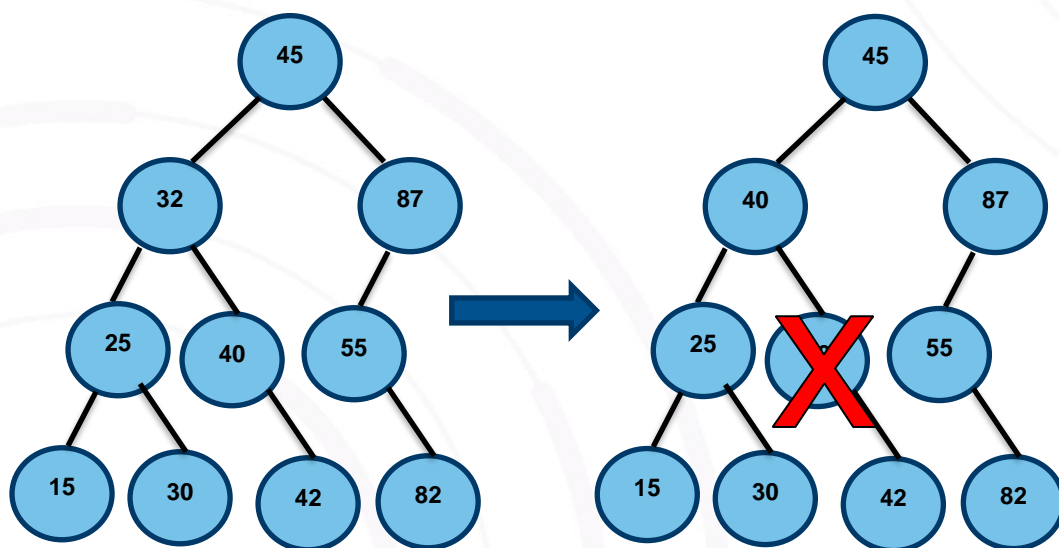
Considere na Figura abaixo, a deleção do 32. Não daria para colocar a sub-árvore do 25 e também a sub-árvore do 40 ligadas no Pai do 32 que é o 45. Mesmo que o 45 não tivesse filho à direita, como o 25 e o 40 são menores do que o 45, não poderiam ser colocados um de cada lado.

A solução é a troca do INFO que será excluído pelo valor imediatamente maior, na sub-árvore da direita. Nesse exemplo seria o 40, que tem um filho. O valor do nó é trocado de 32 para 40 e o nó que é retirado da árvore é o que continha o 40. Deletar um nó que tem apenas um filho já foi visto na situação anterior.

Caso o nó que contém o 40 fosse uma folha, seria só seguir o procedimento da primeira situação vista.

Aí você pode estar pensando... E se o nó 40 tivesse dois filhos? Adivinhei?

Pois fique tranquilo. Isso não vai acontecer. O FESQ do nó 40 será Null, pois se ele tivesse um Filho à Esquerda não seria o imediatamente maior que o 32 e nesse caso a troca não poderia ser pelo 40 e sim pelo seu Filho da Esquerda.

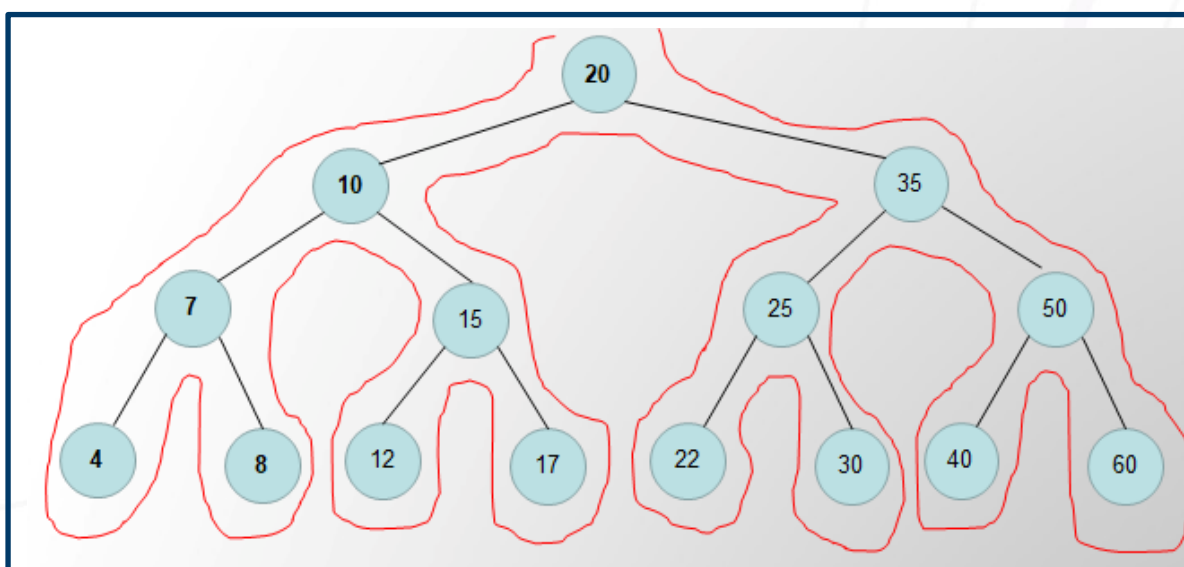


### 3.2.4 Percursos em Árvore Binária

Para listar o conteúdo de uma árvore binária podemos percorrê-la de 3 formas diferentes:

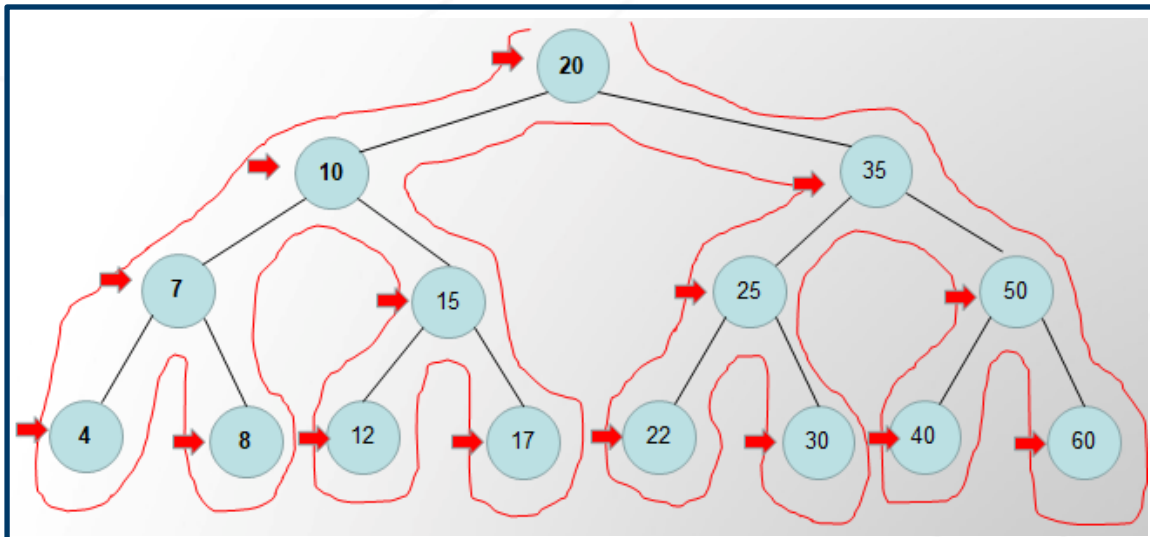
- **Pré-ordem:** Você deve visitar primeiro a raiz, depois a sub-árvore esquerda e por último a sub-árvore direita.
- **Em-ordem:** Você deve visitar primeiro a sub-árvore esquerda, depois a raiz e por último a sub-árvore direita.
- **Pós-ordem:** Você deve visitar primeiro a sub-árvore esquerda, depois a sub-árvore direita e por último a raiz.

Uma dica que facilita a visualização dos percursos citados é traçar uma linha contornando todos os nós da árvore, partindo do nó raiz e ao final retornando ao nó da partida, como no exemplo da Figura abaixo:



Para a PRÉ-ORDEM, os nós são listados sempre que a linha passar pela sua esquerda.

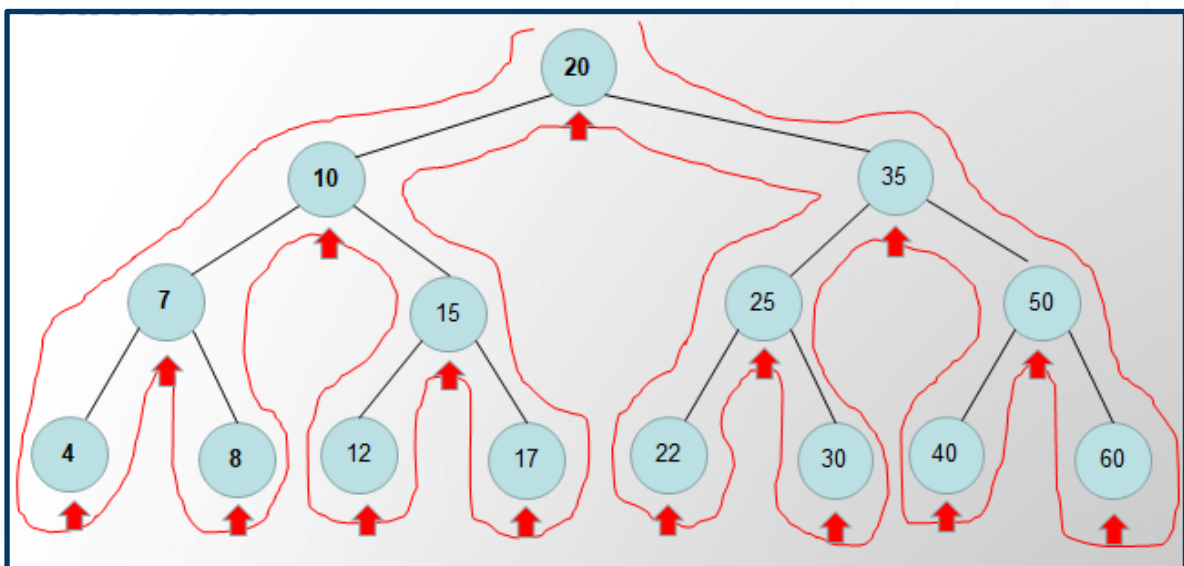




Assim, o Percurso PRÉ-ORDEM resulta em:

20 / 10 / 7 / 4 / 8 / 15 / 12 / 17 / 35 / 25 / 22 / 30 / 50 / 40 / 60

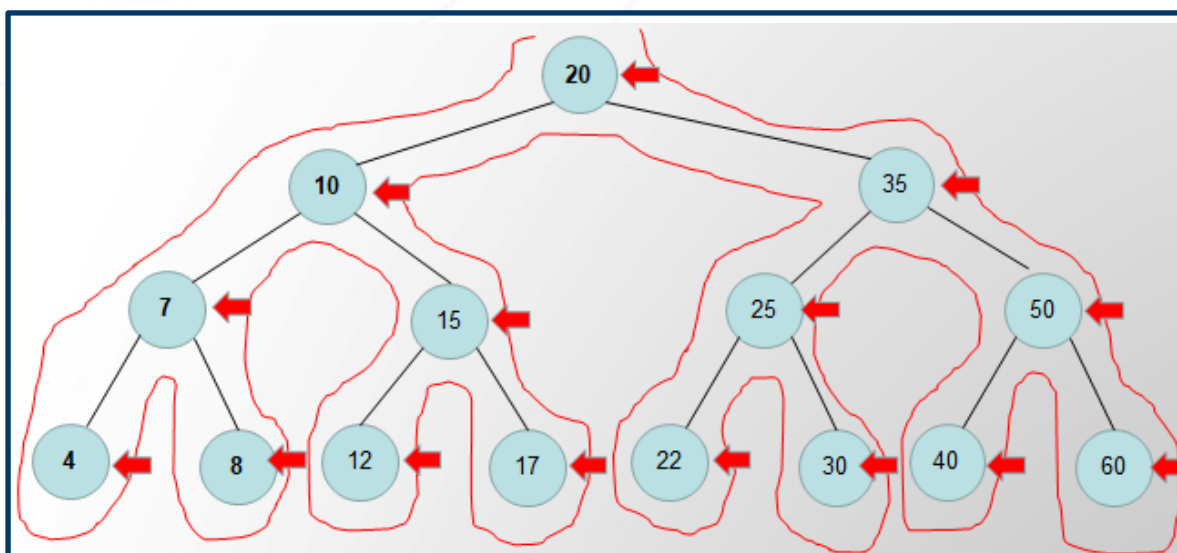
Para listar a árvore pelo Percurso EM ORDEM, o nó será listado toda vez que a linha passar por baixo do nó.



O Percurso EM ORDEM resulta em:

4 / 7 / 8 / 10 / 12 / 15 / 17 / 20 / 22 / 25 / 30 / 35 / 40 / 50 / 60

E para listar a árvore pelo Percurso PÓS-ORDEM, o nó será listado toda vez que a linha passar pela direita do nó.



Já o Percurso PÓS-ORDEM resulta em:

4 / 8 / 7 / 12 / 17 / 15 / 10 / 22 / 30 / 25 / 40 / 60 / 50 / 35 / 20

### 3.2.5 Mãos à Obra

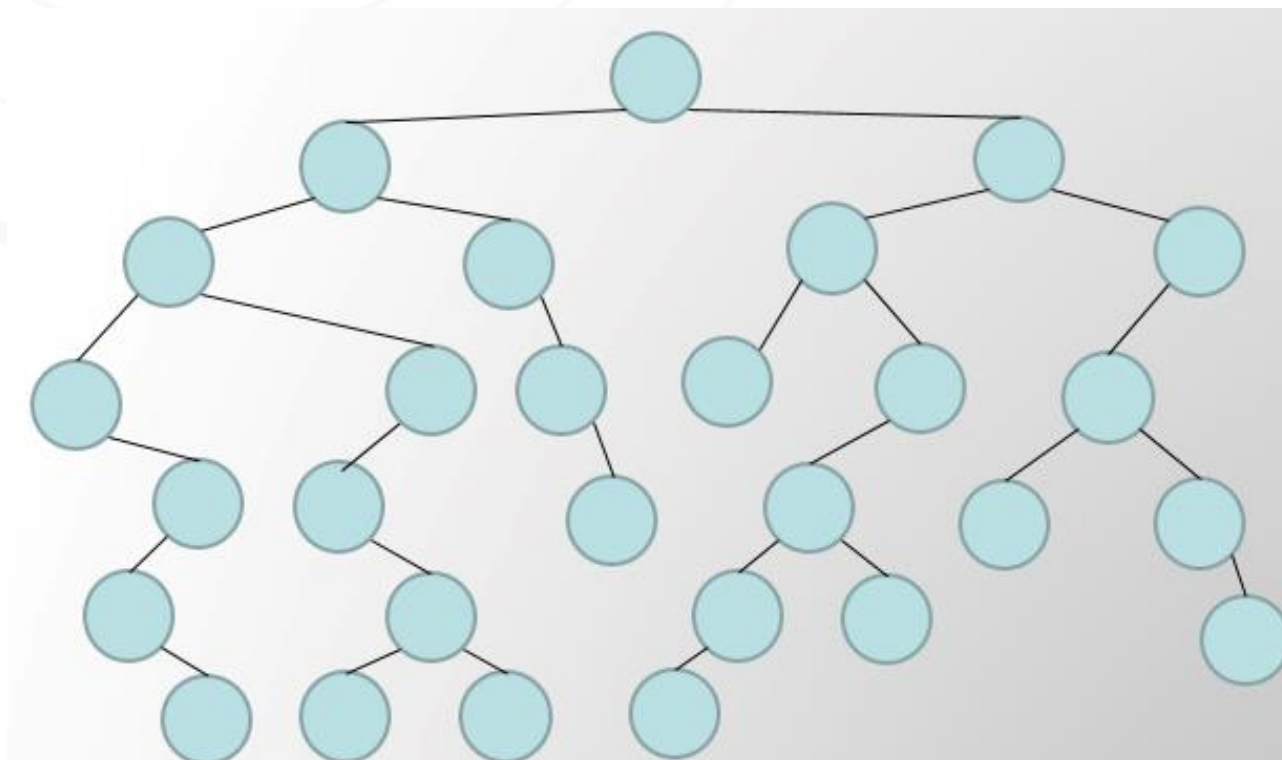
#### EXERCÍCIO 3.2.5-A

Inserir a sequência de números abaixo em uma Árvore Binária, deixando-a em ordem crescente.

32 / 98 / 90 / 75 / 30 / 24 / 10 / 15 / 88 / 17 / 95 / 12 / 72 / 47 / 38 / 48 / 40 / 45 / 42 / 77 / 73 / 46 / 80 / 78 / 27 / 51 / 69 / 60 / 55 / 28.

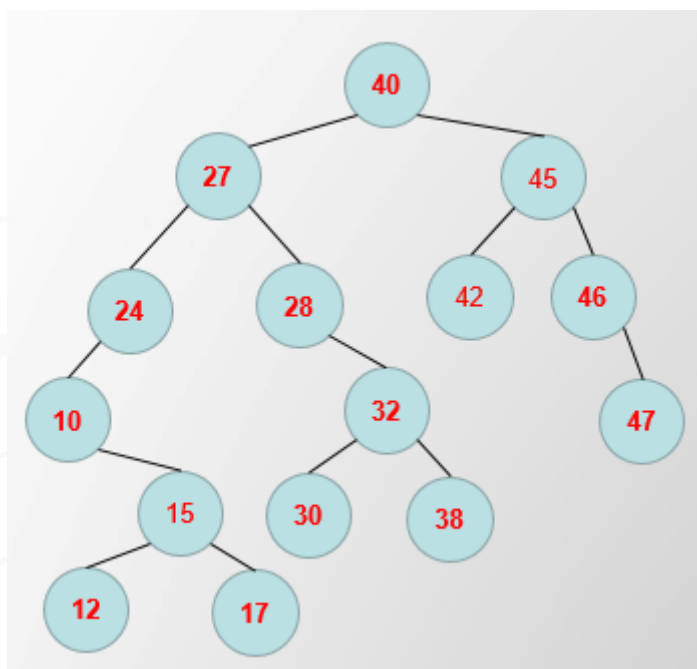
#### EXERCÍCIO 3.2.5-B

Preencher os nós da árvore abaixo com os números de 1 a 28 mantendo a ordem crescente.



### EXERCÍCIO 3.2.5-C

Redesenhe a árvore após deletar os valores: 47, 24 e 27.



### 3.3 Estrutura ÁRVORE AVL

#### 3.3.1 Definição

Árvores AVL são árvores Binárias Balanceadas através de uma técnica desenvolvida em 1962 por Adelson-Velskii e Landis.

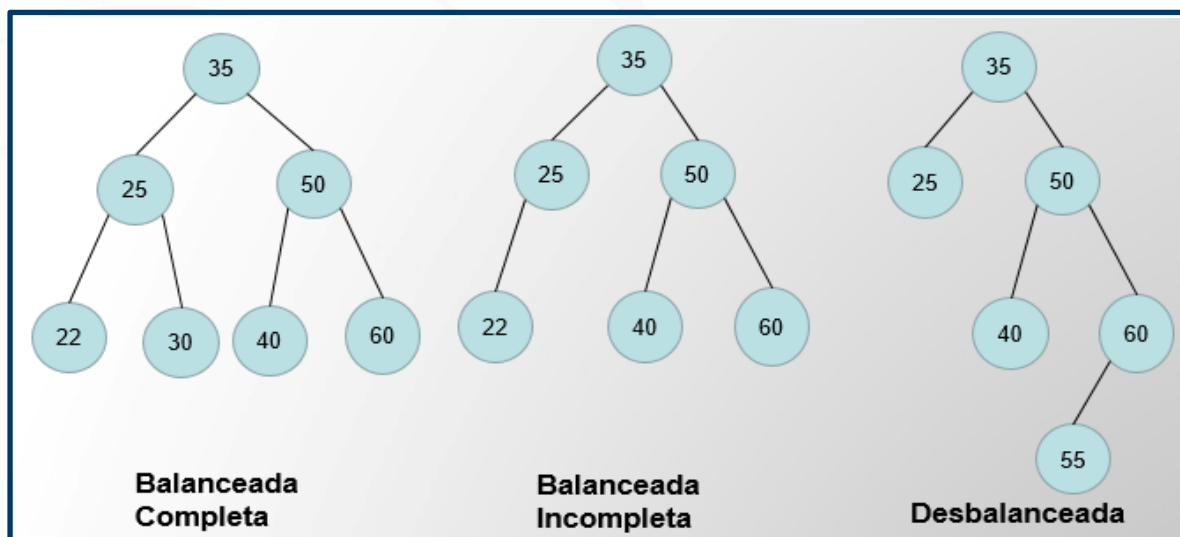
Na árvore AVL a diferença entre as alturas de suas sub-árvores da esquerda e da direita não pode ser maior do que 1 para qualquer nó.

#### 3.3.2 Balanceamento de uma Árvore Binária

A eficiência de uma estrutura em árvore em relação às estruturas lineares é bem maior, pois em média o acesso a um nó procurado exige um percurso menor. Porém vimos que dependendo da ordem dos valores no momento da inserção, a árvore pode ficar com uma altura maior ou menor. Por exemplo, se forem incluídos 100 valores em ordem crescente, cada nó será inserido como Filho da Direita do anterior, gerando uma árvore similar à Lista Duplamente encadeada e o 100º valor estará no nível 99, perdendo totalmente a eficiência.

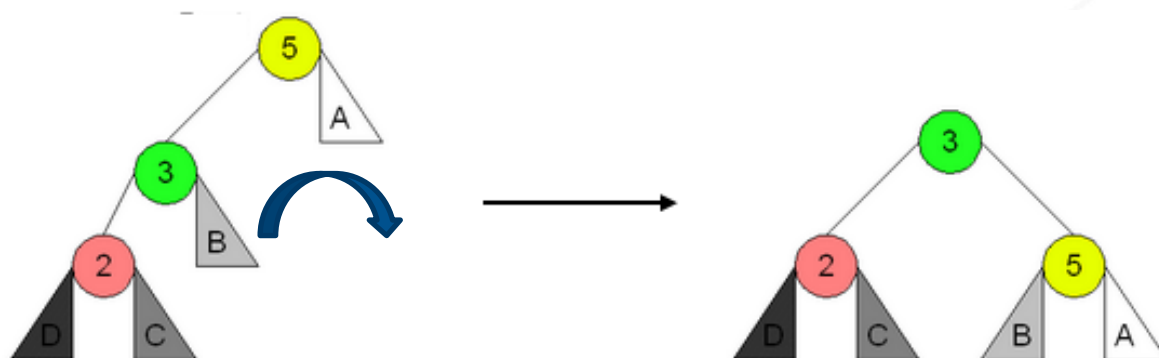
Uma árvore com Altura = 6 poderia armazenar os 100 números, o que melhoraria em muito a performance, pois na pior situação seriam necessários 6 passos para encontrar um número ao invés de 99 passos na situação anterior. A capacidade máxima de nós no nível  $n$  é de  $2^n$  nós.

Uma **árvore binária balanceada** é uma árvore binária na qual as alturas das duas sub-árvores de todos os nós nunca diferem em mais de 1. O balanceamento de um nó é definido como a altura de sua sub-árvore esquerda menos a altura de sua sub-árvore direita.



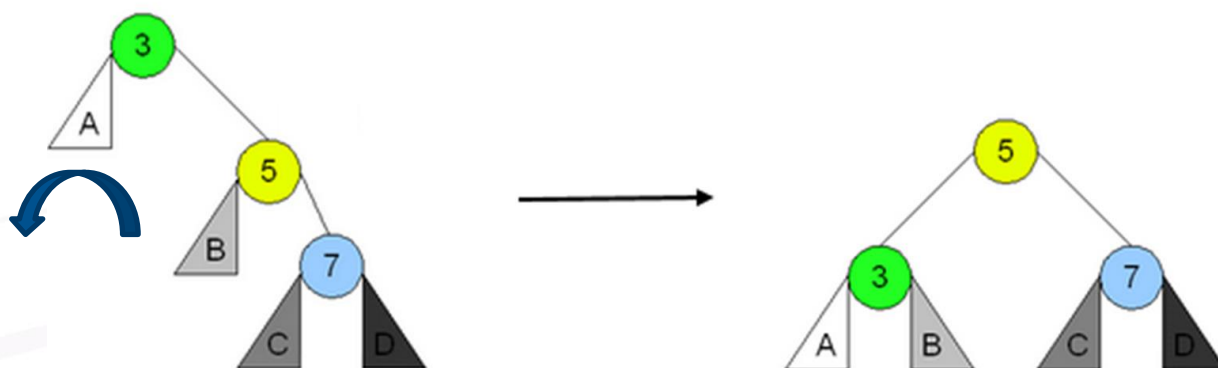
Através de atributos de controle, cada nó identifica a altura de suas sub-árvores e ao ser efetuada uma inserção ou remoção, se for identificado o desbalanceamento, providencia a alteração correspondente, de acordo com a situação identificada, como mostram os exemplos abaixo:

### Situação 1: Rotação Simples à Direita



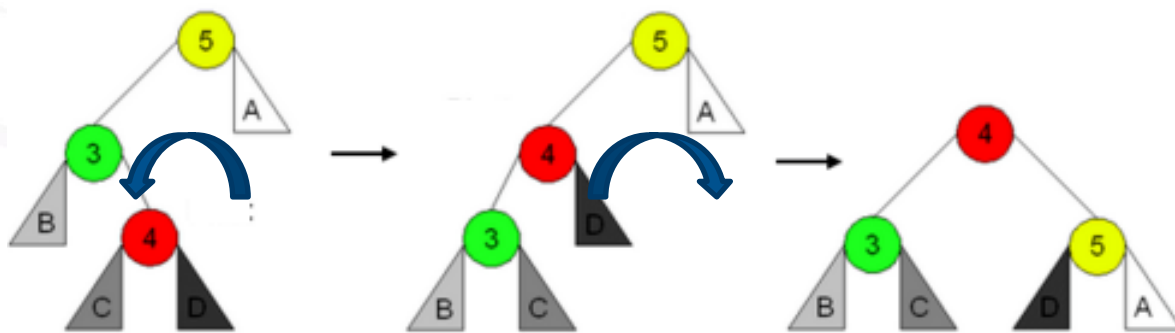
O nó Pai torna-se o filho da direita de Fesq e o nó FEsq anterior torna-se o novo Pai.

### Situação 2: Rotação Simples à Esquerda



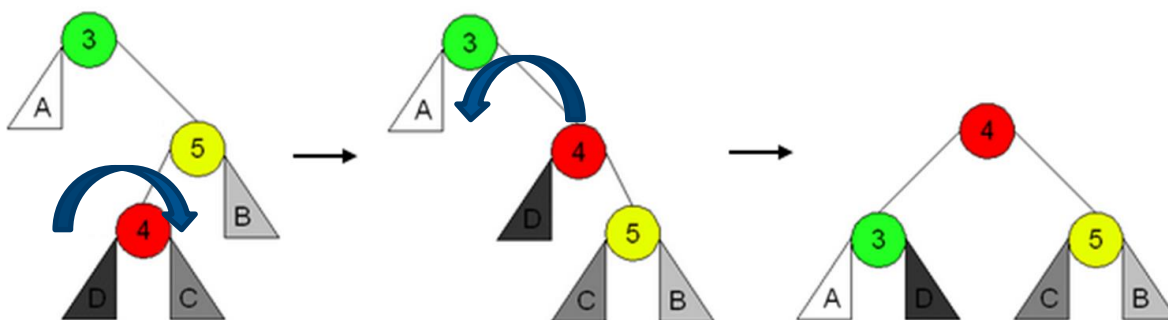
O nó Pai torna-se o filho da esquerda de FDir e o nó FDir anterior torna-se o novo Pai.

### Situação 3: Rotação Dupla à Direita



Aplicar uma rotação à esquerda no nó FEsq e uma rotação à direita no nó Pai.

### Situação 4: Rotação Dupla à Esquerda

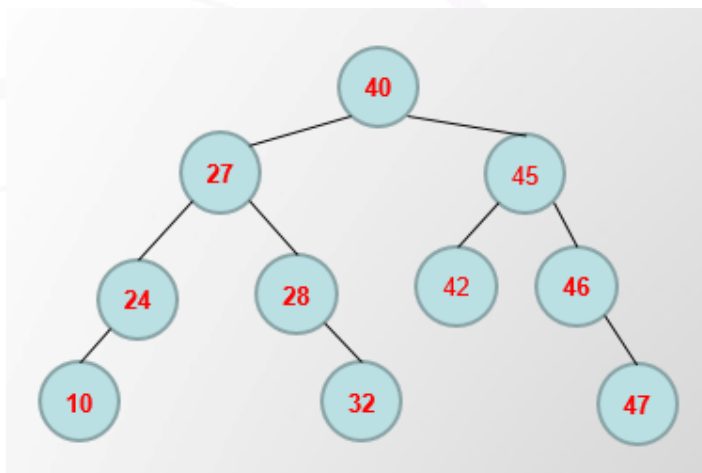


Aplicar uma rotação à direita no nó FDir e uma rotação à esquerda no nó Pai.

#### 3.3.3 Mãos à Obra

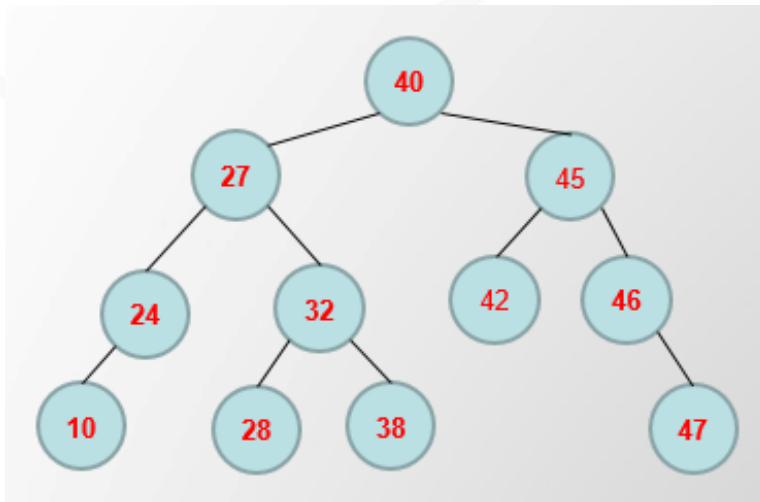
#### EXERCÍCIO 3.3.3-A

Redesenhe a árvore AVL abaixo, após a inclusão de um nó com o valor 38.

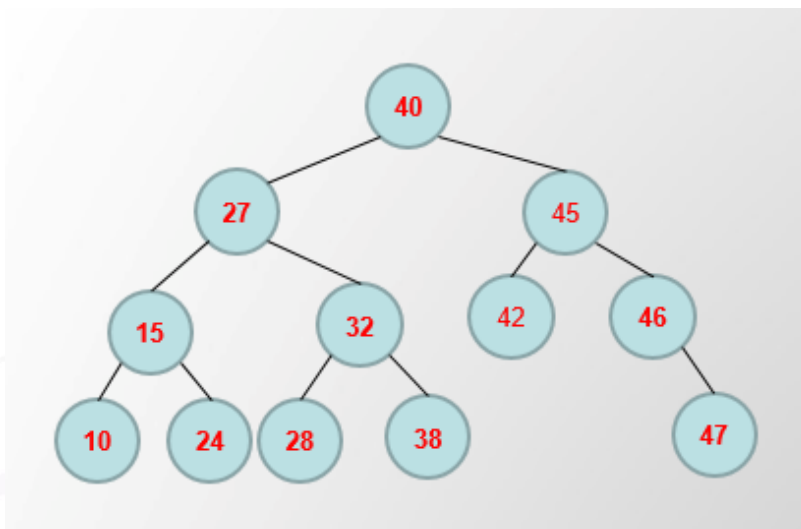


**EXERCÍCIO 3.3.3-B**

Redesenhe a árvore AVL abaixo, após a inclusão de um nó com o valor 15.

**EXERCÍCIO 3.3.3-C**

Redesenhe a árvore AVL abaixo, após a remoção de um nó com o valor 42.

**EXERCÍCIO 3.3.3-D**

Desenhe a árvore AVL resultante da inclusão na sequência dos valores a seguir:

40 / 27 / 47 / 15 / 46 / 25 / 45



## 3.4 Estrutura ÁRVORE B

### 3.4.1 Definição

Uma árvore B, também denominada B-TREE, é uma estrutura de dados em árvore balanceada, que armazena diversas chaves (INFO) classificadas em uma mesma página (nó). A árvore B é uma generalização de uma árvore binária em que um nó pode ter mais que dois filhos, muito importante quando o volume de dados é grande, para otimizar a transferência de dados do armazenamento secundário para a memória, diminuindo o tempo de acesso para operações de busca, inserção e remoção.

Foi criada por Bayes e McCreight em 1972 enquanto trabalhavam no Boeing Scientific Research Labs. Sua denominação não é consenso se o B é devido à sua característica de sempre ser Balanceada ou se é devido a inicial do criador Bayes ou da empresa Boeing. Mas, a origem do nome é o que menos importa.

O número de **ordem** da Árvore B, definida na sua criação, é que indica a quantidade de chaves de cada página e consequentemente a quantidade de filhos de cada página.

### 3.4.2 Conceitos Gerais

Uma árvore B de ordem  $t$  é uma árvore ordenada que, se não estiver vazia, satisfaz as seguintes condições:

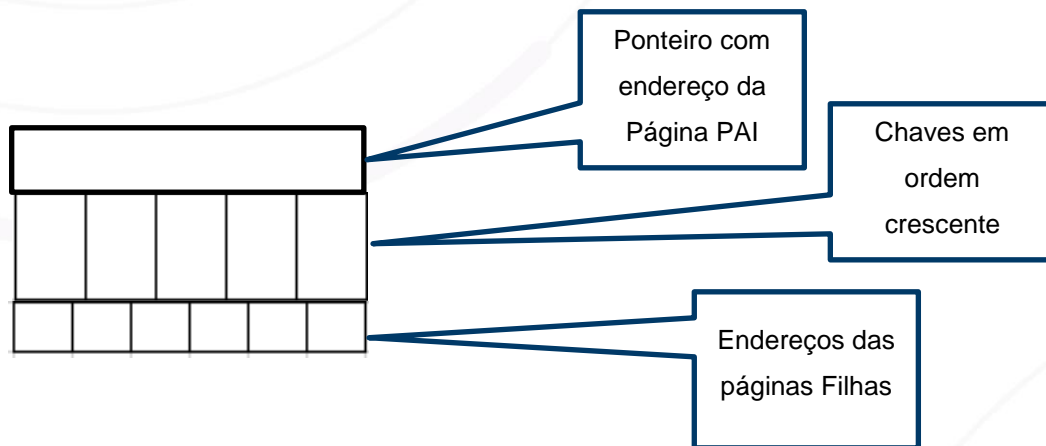
1. A raiz, se não for uma folha, tem no mínimo dois filhos;
2. Cada página, exceto a raiz e as folhas, possui no mínimo  $t$  filhos;
3. Cada página, inclusive a raiz tem no máximo  $2t$  filhos;
4. A quantidade de filhos de cada página é igual a quantidade de chaves + 1;
5. Todas as folhas estão no mesmo nível.

Portanto: Cada página possui entre  $t-1$  e  $2t-1$  chaves, exceto a raiz que possui entre 1 e  $2t-1$  chaves.

Exemplo:  $t = 3$

- cada página, exceto a raiz e as folhas, possui:
  - 2 a 5 chaves;
  - 3 a 6 filhos.





A página do Filho 1 conterá chaves com valores menores do que a Chave 1;

A página do Filho 2 conterá chaves com valores entre a chave 1 e a chave 2;

A página do penúltimo filho preenchido conterá valores entre a penúltima chave preenchida e a última chave preenchida;

A página do último filho preenchido conterá valores maiores que a última chave preenchida.

### 3.4.3 Inclusão em Árvore B

Embora na prática o número de ordem seja muito maior, para efeito de aprendizado vamos trabalhar com número de ordem = 3, que como vimos permite um máximo de 5 chaves por página e portanto, 6 filhos.

Na inclusão em uma árvore B de ordem  $t = 3$ , os 5 primeiros valores incluídos ficarão em ordem crescente na página Raiz.

Vamos ver no exemplo abaixo a inclusão dos valores, na ordem apresentada:

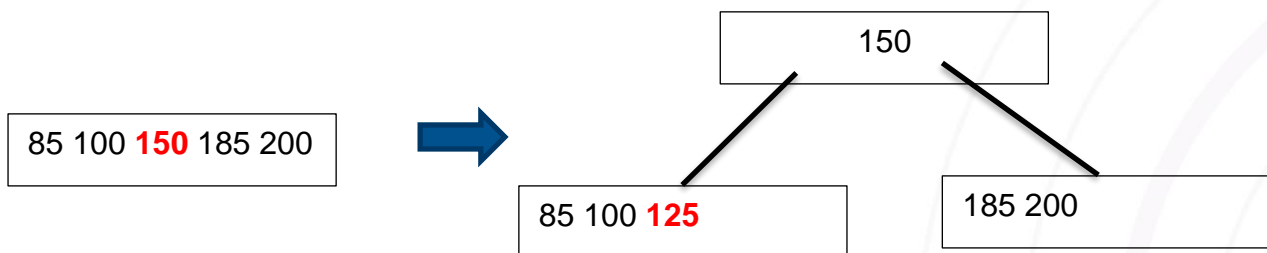
150 85 100 185 200 125 42 160 90 50 130 155 20 35 87 300 250 220 180 270 110  
380 95 140

Após incluir os 5 primeiros valores na Árvore B vazia, teremos a raiz lotada:

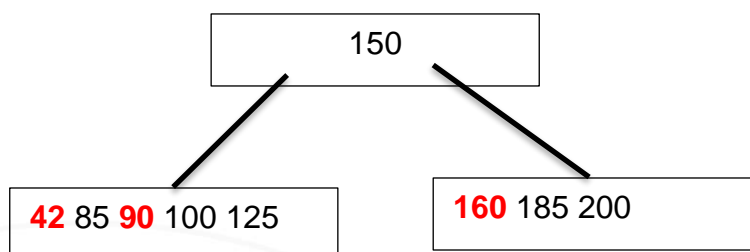
**85 100 150 185 200**

Ao incluir o próximo valor, o 125, a árvore sofrerá uma primeira quebra. Note que neste caso a página lotada é a raiz e também é uma folha. A quebra na árvore B não ocorre para baixo, mas sim a página se rompe em duas, com a chave do meio subindo para o nível de cima. As chaves anteriores ficam numa página e as chaves posteriores em outra. Neste caso, como a quebra está ocorrendo na Raiz, a chave intermediária inaugura uma nova raiz e as páginas quebradas passam a ser filhas da Raiz.

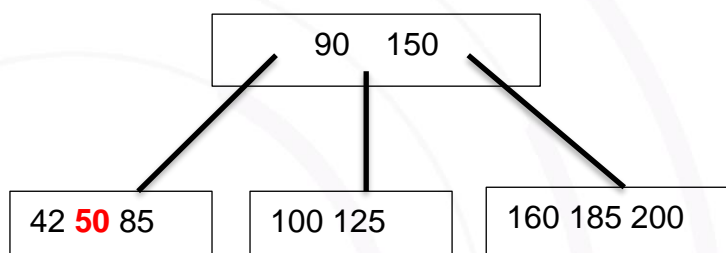
Como o valor a ser incluído é menor do que a chave intermediária ( $125 < 150$ ), a inclusão se dá na página Filho da Esquerda.



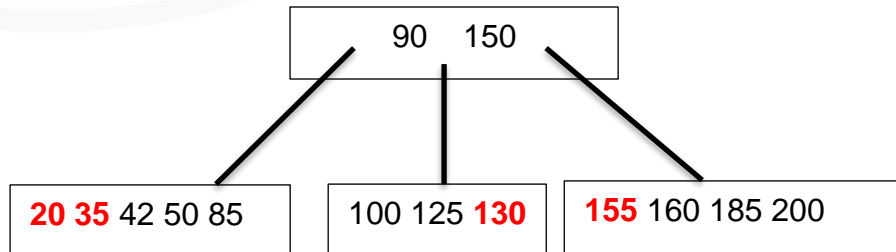
Continuamos a inclusão um a um. O 42 entra no Filho da Esquerda, o 160 entra no Filho da Direita e o 90 entra no Filho Da Esquerda.



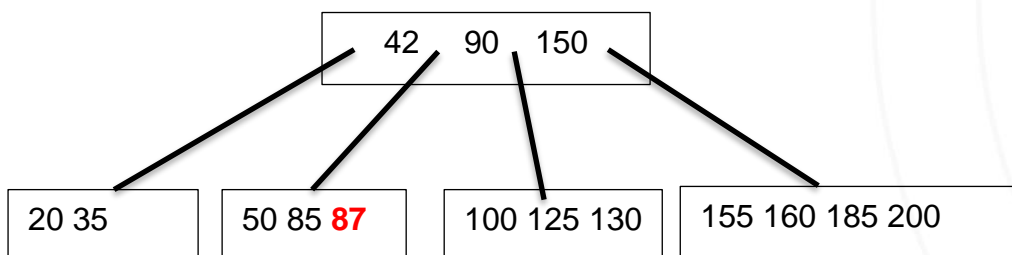
Ao ser incluído o 50, como é menor do que o único valor da raiz ( $50 < 150$ ), a inclusão deve ser feita no Filho da Esquerda que está lotado, provocando nova quebra, subindo a chave intermediária para o Pai:



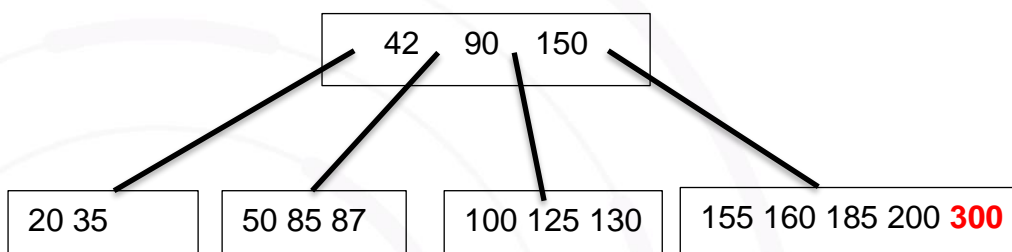
Continuando as inclusões o próximo valor é o 130 que entrará no Filho do meio, por ser maior do que a primeira chave da raiz e menor do que a segunda. O 155 entrará no Filho da Direita e os valores seguintes 20 e 35 completarão a capacidade do Filho da Esquerda.



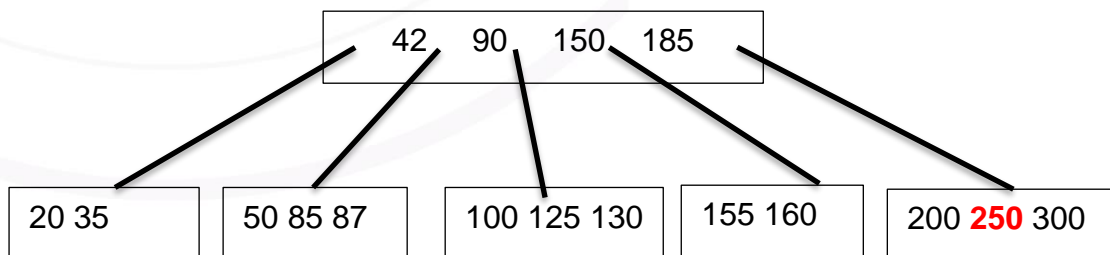
A próxima inclusão é o 87 que é encaminhado para o Filho da Esquerda, mas encontra a página lotada, provocando a quebra para poder incluir. Com isso o 42 por ser a chave intermediária sobe para o Pai e a página se divide em duas. O 87 fica na segunda parte da divisão por ser maior do que o 42.



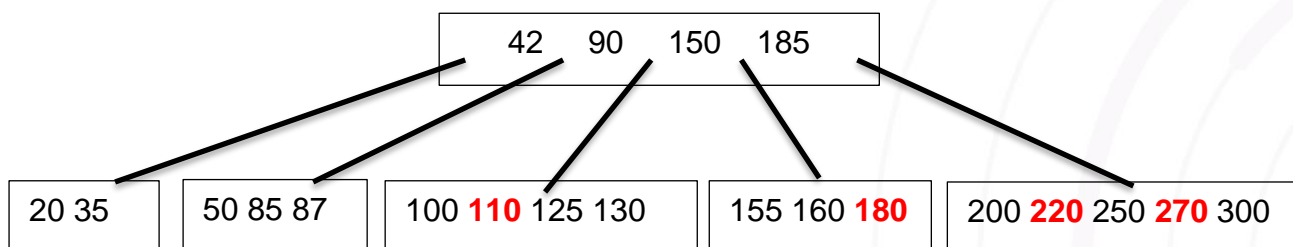
A próxima inclusão é o 300 que é encaminhado para o Filho da Direita, completando a sua capacidade:



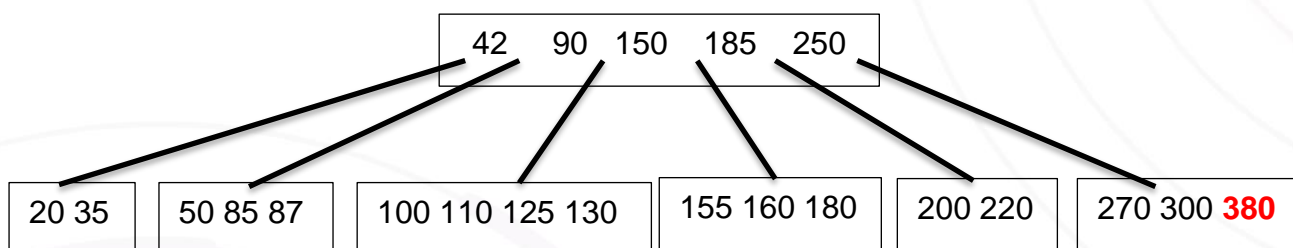
Como a próxima inclusão, o 250, também deveria integrar a página dos maiores valores, que está lotada, é provocada nova quebra. Como você já sabe, sobe o do meio e fica metade em cada página após a quebra.



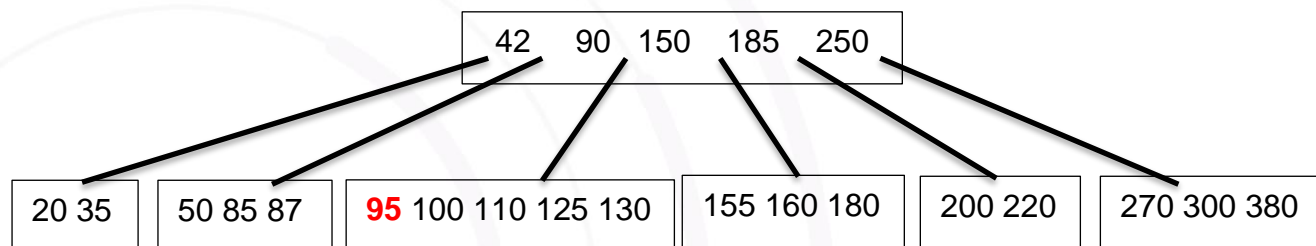
Continuando as inclusões, o 220 é direcionado para o Filho mais à direita, o 180 para o Filho intermediário entre o 150 e o 185 o 270 vai completar a capacidade da Página dos maiores valores, o 110 vai para a página intermediária entre o 90 e 150.



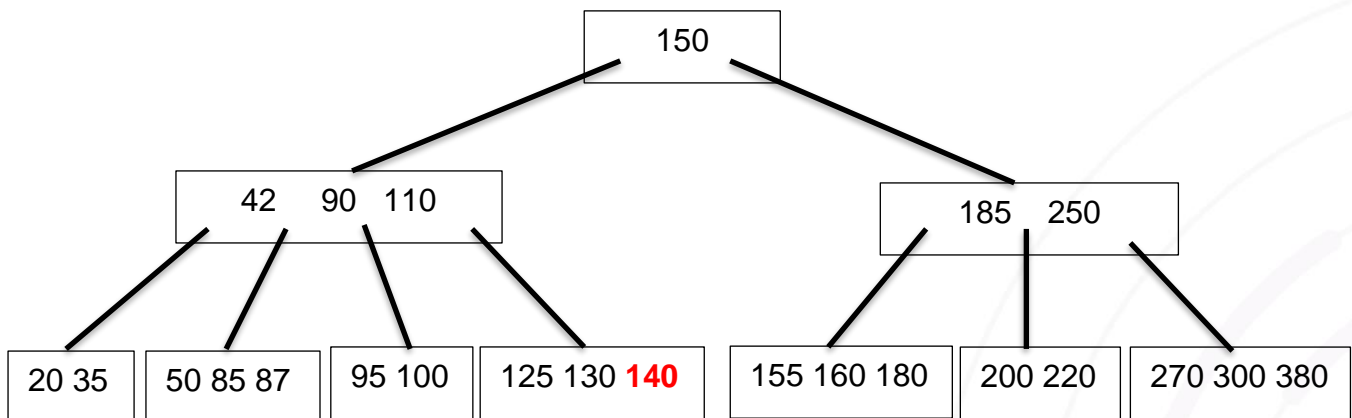
Mas, quando chega a vez de incluir o 380, naturalmente na página mais à direita, é provocada uma nova quebra, subindo o 250.



Ao incluir o 95 a página Filho intermediário entre o 90 e o 150 fica repleta.



Finalmente chegou a vez de incluir o 140, que será direcionado para a mesma página que recebeu o 95. Já sabemos que como a página está lotada sofrerá uma quebra, subindo o 110 que é a chave intermediária. Mas... a Raiz está lotada. Não vai caber o 110. Então o mesmo procedimento ocorrido nas folhas será feito na raiz, aumentando a altura da árvore.



Concluído.

Bem, acredito que com esse passo-a-passo deu para demonstrar a dinâmica da inclusão numa Árvore B e entendido melhor os conceitos inicialmente apresentados sobre essa estrutura de dados. Mas, que tal rever os conceitos agora que ficou mais claro?

### 1. A raiz é uma folha ou tem no mínimo dois filhos;

Note que na inclusão das 5 primeiras chaves a raiz era uma folha, pois não tinha filhos. Ao ser incluído o 125 e posteriormente ao incluir o 110 foram provocadas quebras na raiz ficando a nova raiz com apenas uma chave e dois filhos.

### 2. Cada página, exceto a raiz e as folhas, possui no mínimo $t$ filhos;

No caso do exemplo apresentado o  $t$  é igual a 3. As folhas não têm filhos e a raiz quando a árvore estava vazia e quando ocorre quebra, fica com apenas 1 chave e, portanto, 2 filhos, como ocorre com a árvore binária.

### 3. Cada página, inclusive a raiz tem no máximo $2t$ filhos;

Cada página contém além do atributo ponteiro Pai, dois arrays sendo um de chaves e outro de ponteiros para indicar o endereço das páginas Filhos. Como sabemos,

um array tem um tamanho pré-definido e para  $t = 3$ , esse tamanho é 5 chaves e 6 filhos.

**4. A quantidade de filhos de cada página é igual a quantidade de chaves + 1;**

Como vimos no exemplo, tem uma página Filho que recebe as chaves menores que a primeira chave do Pai, uma página Filho para receber as chaves intermediárias entre cada chave e uma página Filho para as chaves maiores que a última chave do Pai. Mesmo quando algumas chaves não estão preenchidas, essa relação é mantida.

**5. Todas as folhas estão no mesmo nível.**

Como visto, as quebras provocam a subida de uma chave mantendo as folhas sempre no mesmo nível, fazendo com que a árvore nunca fique desbalanceada.

### 3.4.4 Exclusão em Árvore B

A remoção é análoga à inserção, devendo ser mantidas as propriedades da árvore B. Uma chave pode ser eliminada de qualquer página e não apenas de páginas folha. A remoção de um nó interno exige que os filhos do nó sejam reorganizados.

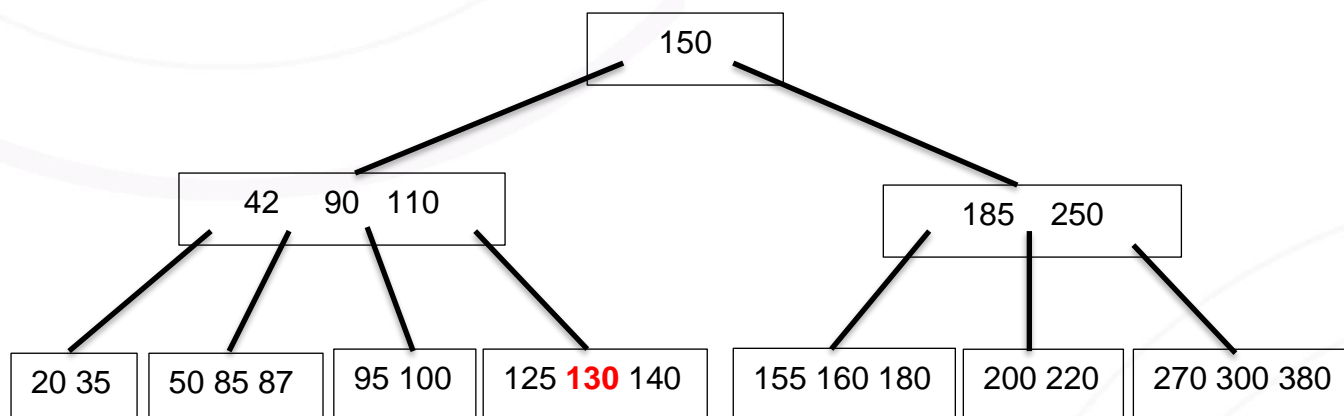
Dependendo do posicionamento da chave a ser removida, teremos soluções específicas para que as propriedades da árvore B sejam respeitadas:

**Situação 1** - Remoção de uma chave em um nó folha, que tem quantidade de chaves maior que o mínimo exigido numa página ( $2t-1$ ).

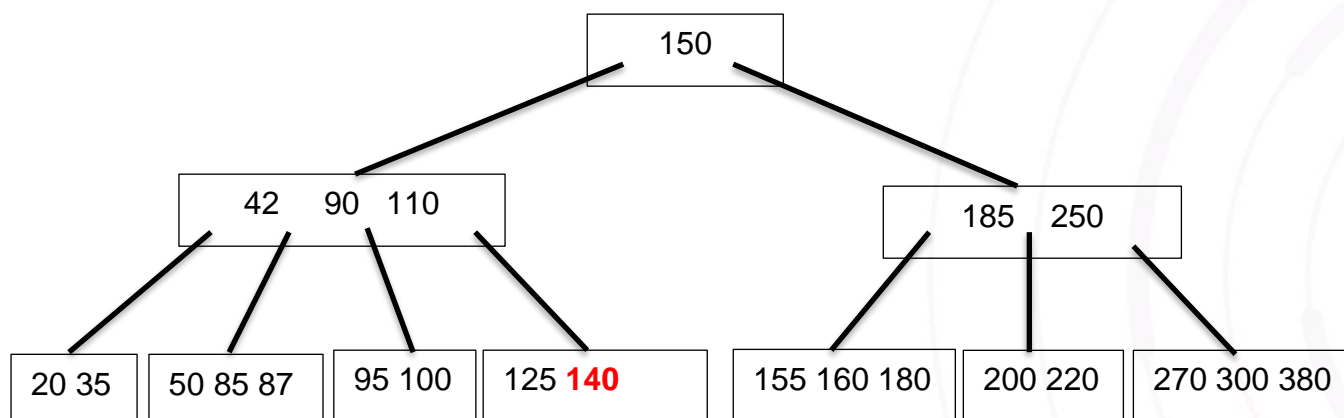
Essa é a solução mais simples. A chave será excluída da página e caso não seja a última preenchida, as chaves posteriores dentro da página devem ser remanejadas para não ficar espaço vazio entre chaves preenchidas.

Exemplo: Remover o 130

Antes:



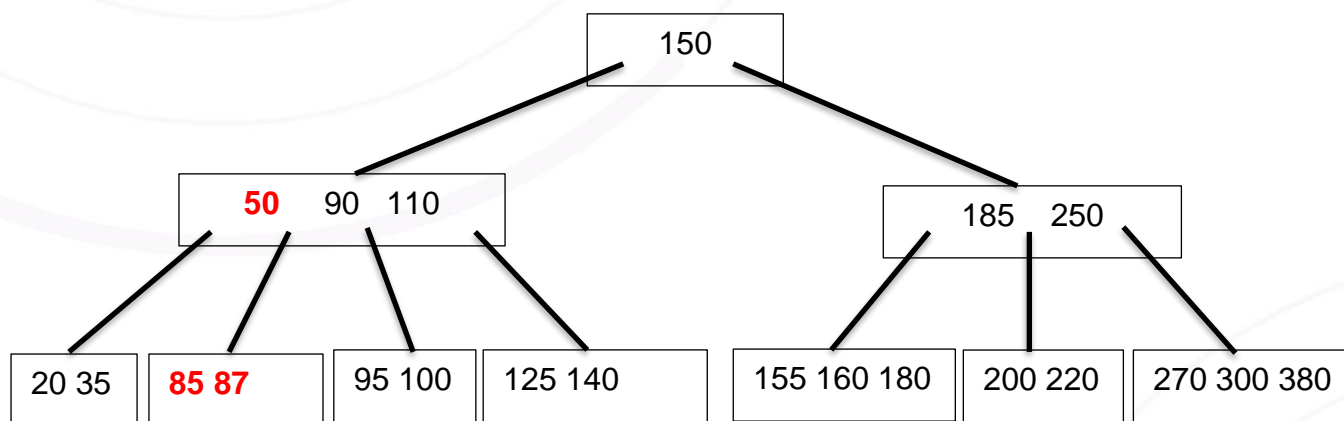
Depois:



### Situação 2 - Remoção de uma chave em uma página não folha.

Não podemos simplesmente remover um valor de páginas intermediárias ou da raiz, pois elas têm um papel importante no direcionamento para as páginas Folhas. Para que a árvore não fique deficiente, o seu lugar na página deve ser ocupado pelo imediatamente maior, que estará numa Folha. A Folha é que perderá uma chave, pois cederá para o Pai a menor chave da página e realocará as demais para não ficar espaço vazio no seu array de chaves. Caso fique com quantidade de chaves abaixo da mínima deve haver nova correção.

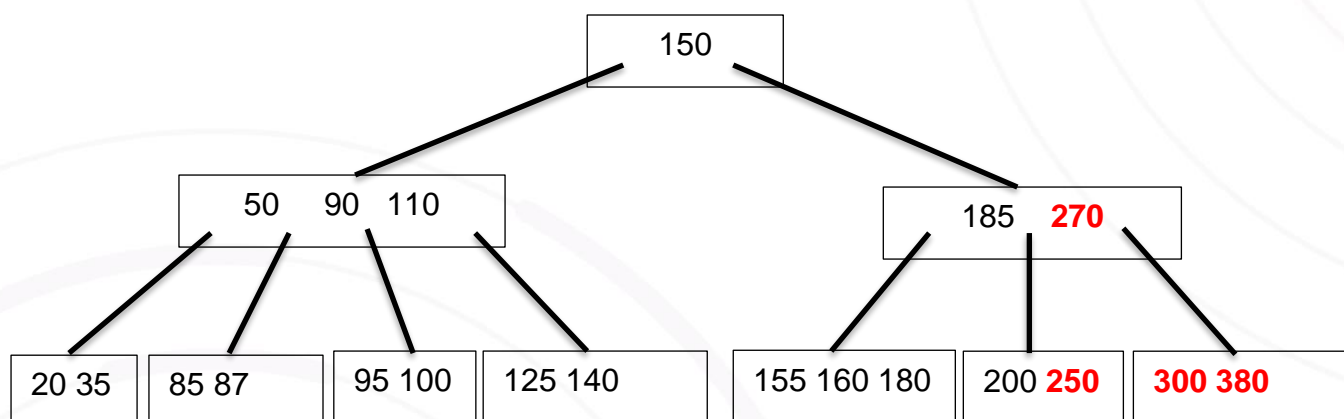
Exemplo: Remover o 42 da árvore resultante da situação anterior:



**Situação 3** - Remoção de uma chave em uma página que possuía a quantidade mínima de chaves, mas tenha uma página irmã adjacente que contenha mais chaves do que o mínimo.

Redistribuir as chaves entre as páginas irmãs. A chave separadora entre as irmãs desce para o Filho que terá a chave removida permanecendo com a quantidade mínima de chaves. A irmã cede para o Pai a chave com valor mais próximo do valor da chave cedida para a irmã.

Exemplo: Remover o 220 da árvore resultante da situação anterior:

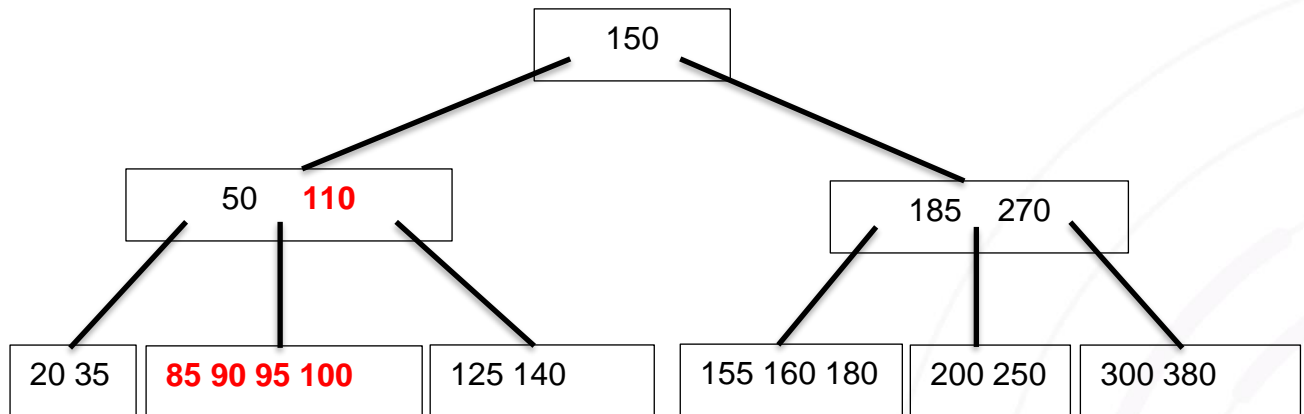


**Situação 4** - Remoção de uma chave em uma página que possuía a quantidade mínima de chaves, mas suas páginas irmãs adjacentes possuem a quantidade mínima de chaves.



Concatenar a página que teve a chave removida com uma irmã, recebendo também a chave separadora que estava no Pai, formando. Caso essa alteração provoque underflow (quantidade abaixo do mínimo) no Pai, corrigir também o nó Pai.

Exemplo: Remover o 87 da árvore resultante da situação anterior:

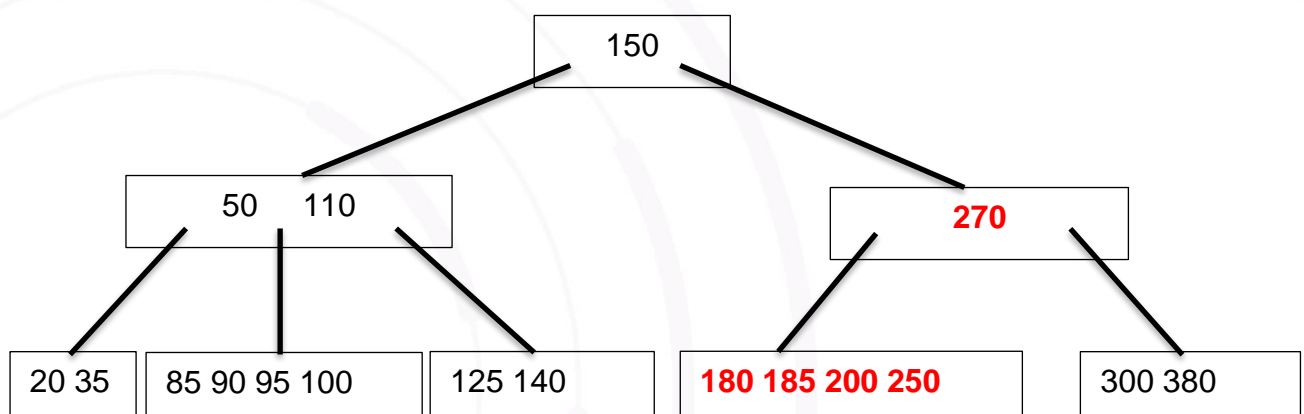


**Situação 5** - Underflow no nó pai causado pela remoção de uma chave em um nó filho.

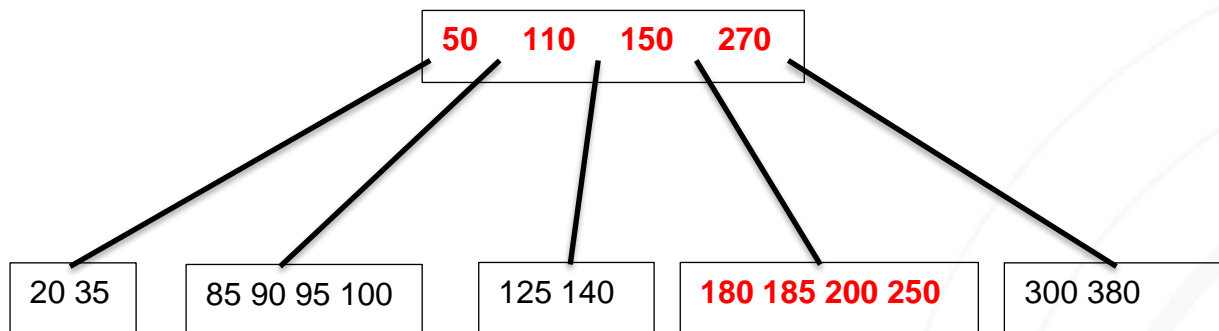
Dependendo da quantidade de chaves que a página irmã adjacente possui, utilizar redistribuição ou concatenação,

Exemplo: Vamos utilizar a árvore resultante da situação anterior para, depois de ter removido o 155 efetuar a remoção do 160 (poderia ser qualquer outra chave dessa página ou de suas irmãs):

Solução intermediária:



Naturalmente essa solução não está concluída, pois a página intermediária não pode conter apenas uma chave. Como sua irmã está com a quantidade mínima para sua ordem  $t = 3$  será realizada a concatenação entre as duas páginas intermediárias e como a raiz só possui uma chave que será absorvida pela concatenação, a nova página assumirá a função de raiz da árvore B.

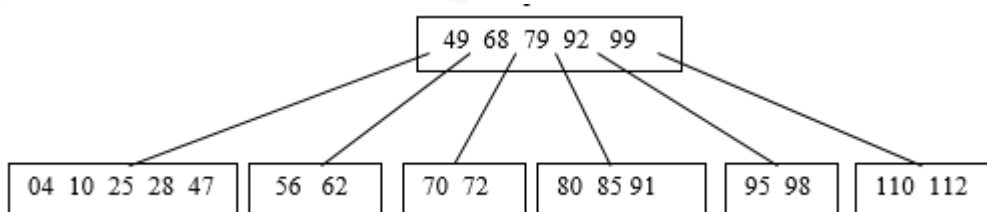


Caso a página intermediária irmã tivesse pelo menos mais uma chave além do 50 e 110, seria realizada uma redistribuição passando o 150 da Raiz para a página que sofreu o underflow e a irmã cederia a chave 110 para a raiz. Com isso, a folha que contém as chaves 125 e 140 seriam apontadas pela folha intermediária da direita à frente do 150.

### 3.4.5 Mãos à Obra

#### EXERCÍCIO 3.4.5-A

Como ficará a árvore B abaixo, com fator = 3, após serem incluídos os valores 82; 88; 23 e 90 na sequência dada?



## 3.5 Estrutura ÁRVORE B\*

### 3.5.1 Definição

A árvore B\* (B asterisco) foi criada em 1973 por Donald E. Knuth, sendo uma variação da árvore B. Apresenta mecanismos de inserção, remoção e busca muito semelhantes aos realizados em árvores B, mas com a diferença em que a técnica de

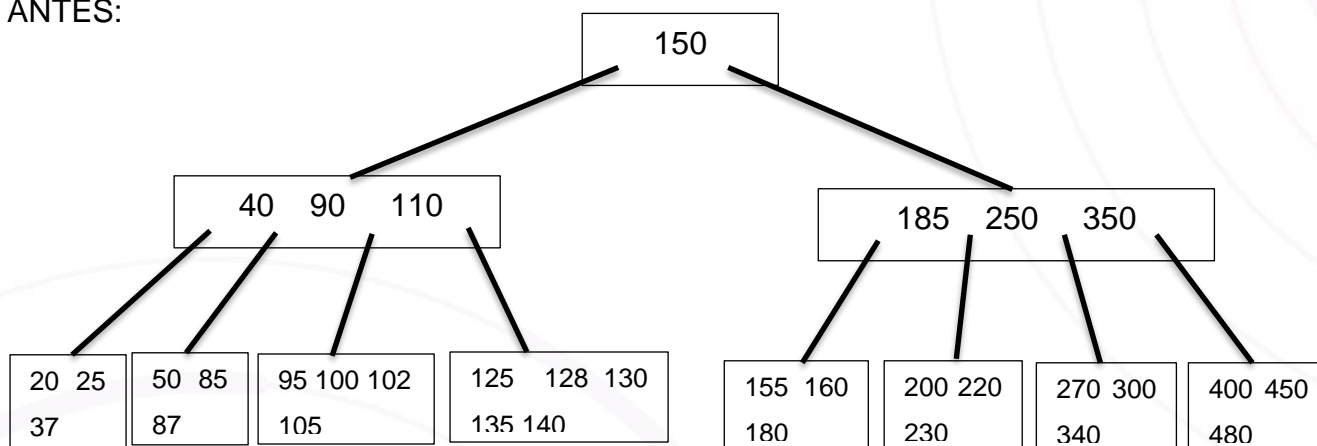
redistribuição de chaves também é empregada durante as operações de inserção. Dessa maneira a operação de split pode ser adiada até que duas páginas irmãs estejam completamente cheias e, a partir daí o conteúdo dessas páginas irmãs é redistribuído entre três páginas.

Tal técnica é conhecida como divisão two-to-three split, ou no português divisão de dois para três, que proporciona propriedades diferentes das árvores B na qual utiliza a divisão usual conhecida como one-to-two split. A grande melhoria direta proporcionada por essa abordagem é o melhor aproveitamento de espaço do arquivo, pois, no pior caso, cada página apresenta no mínimo  $2/3$  do número máximo de chaves que a página pode armazenar, enquanto na divisão usual cada página apresenta, no pior caso, metade do número máximo de chaves.

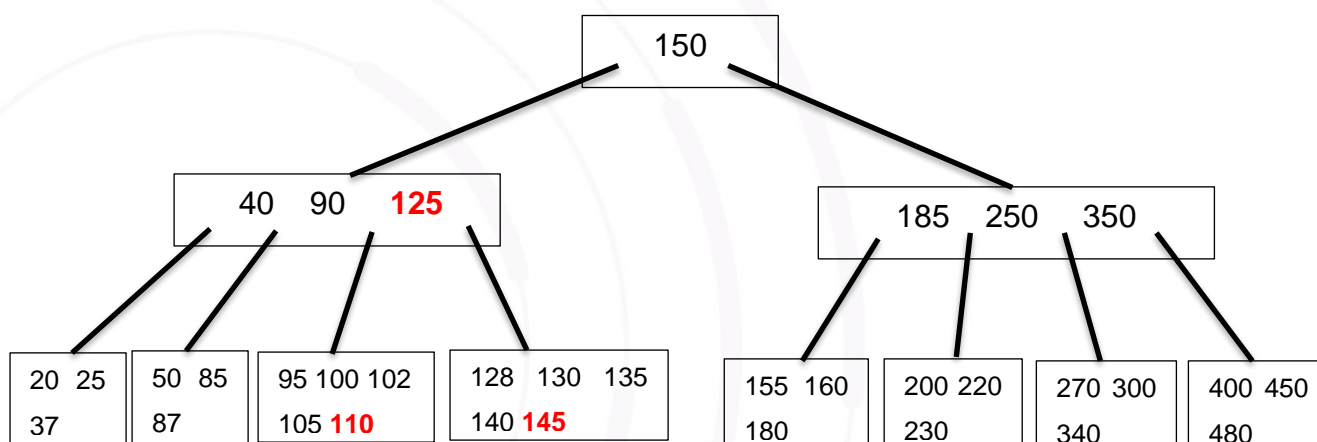
### 3.5.2 Comparação com a Árvore B

Vamos dar um exemplo para entender a principal diferença entre a Árvore B e a Árvore B\*. Vamos inserir o valor 145 na árvore B\* abaixo:

ANTES:



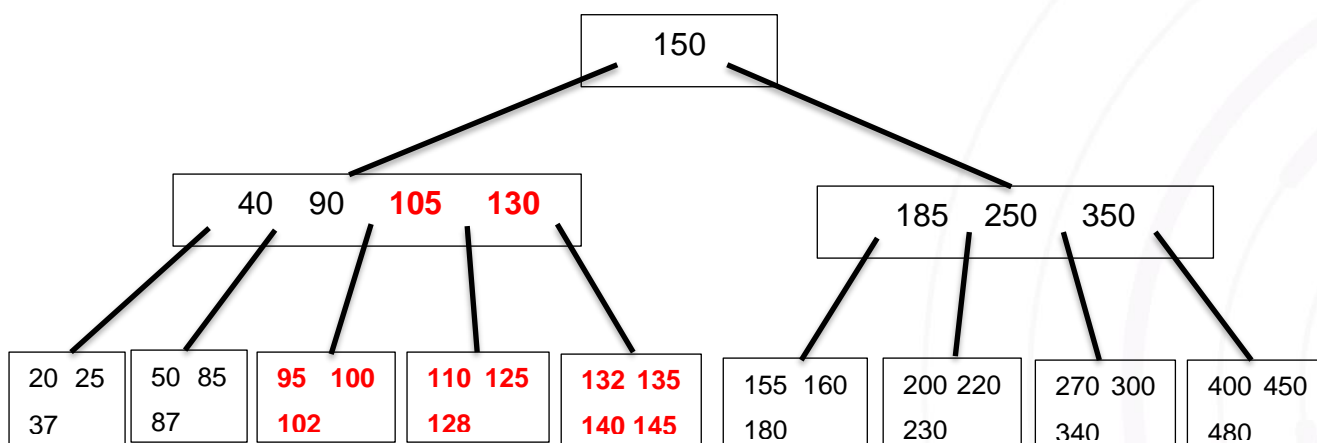
APÓS INCLUIR O 145:



A página que deve receber o 145 estava repleta, mas ao invés de provocar o split em duas páginas, a Árvore B\* faz a redistribuição com a página irmã, que ainda tinha espaço livre, descendo para ela a chave separadora constante no Pai e cedendo para o Pai a sua chave de valor mais baixo. Posteriormente incluiu o 145 mantendo suas chaves em ordem crescente.

Agora vamos incluir o valor 132 na mesma árvore:

APÓS INCLUIR O 132:



Ao incluir o 132 e encontrar a página repleta e tentar fazer a redistribuição, encontrou a página irmã também repleta. Nessa situação a árvore B\* provoca o Split de 2 para 3 páginas, como demonstrado.

Apenas a Raiz, como não tem irmãos, ao estar repleta e receber novo integrante, faz o Split de 1 para 2 páginas de forma semelhante à Árvore B.

## 3.6 Estrutura ÁRVORE B+

### 3.6.1 Definição

Árvores B+ são derivadas das árvores B com Split de 1 para 2 páginas, mas com uma forma diferente de armazenamento de suas chaves. São muito empregadas em banco de dados e sistemas de arquivos como o NTFS para o Microsoft Windows, o sistema de arquivos ReiserFS para Unix, o XFS para IRIX e Linux, e o JFS2 para AIX, OS/2 e Linux, usam este tipo de árvore.

Assim como as árvores B, as árvores B+ visam reduzir as operações de leitura e escrita em memória secundária, uma vez que, essas operações são demoradas para um sistema computacional e devem ser minimizadas sempre que possível.

Árvores B são muito eficientes, mas não são muito apropriadas para acesso sequencial, pois quando termina o percurso dentro de uma página, precisa alternar de páginas para buscar o próximo integrante e nele ter acesso ao endereço da página seguinte. Ao terminar a primeira sub-árvore da página raiz, esse percurso deve ser de retorno até a raiz para descer até a próxima página e acessar o valor seguinte. A árvore B+ permite o acesso sequencial de forma bem mais eficiente.

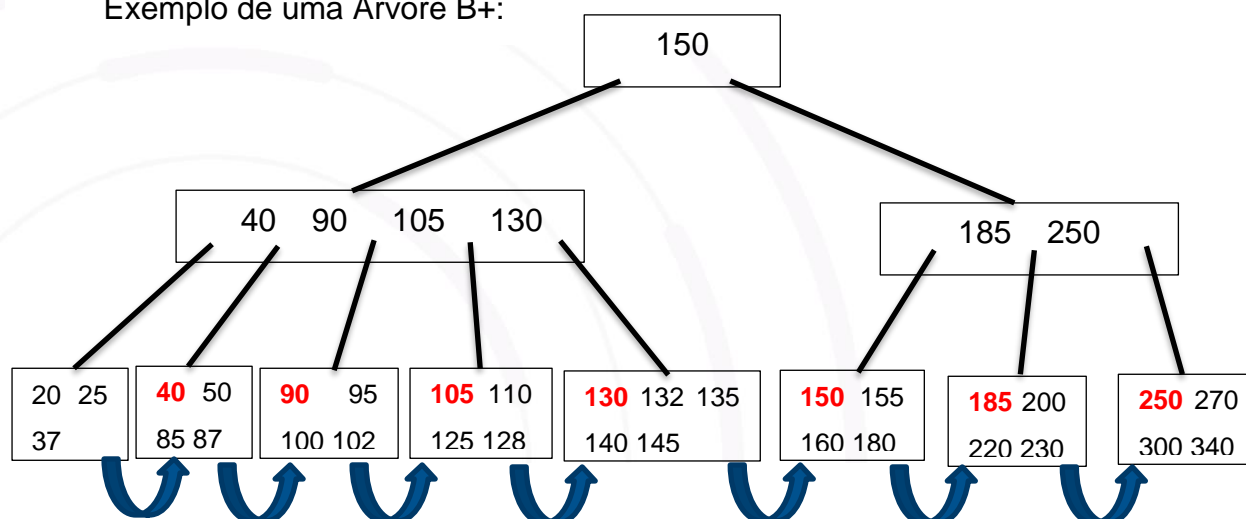
A ideia inicial desta variação da árvore B é manter todas as chaves de busca em seus nós folha, replicando nelas os valores constantes na raiz e nas páginas intermediárias.

Para manter o acesso sequencial, cada nó folha contém um ponteiro adicional que indica o endereço da próxima folha, similar ao ponteiro NEXT, formando o nível das folhas uma sequência similar à uma Lista Encadeada.

### 3.6.2 Comparação com a Árvore B

O comportamento referente à inserção e remoção de chaves na Árvore B+ é semelhante às movimentações na árvore B, respeitando-se as características que a diferenciam como a replicação de todos os valores nas folhas e o ponteiro para a folha seguinte.

Exemplo de uma Árvore B+:



Pode parecer um desperdício tal replicação, fazendo com que a página fique repleta mais cedo e provoque eventualmente uma altura maior do que a árvore B com os mesmos dados. Lembre-se que nos nossos exemplos adotamos utilizar a ordem = 3, mas na prática esse número é muito maior e essa chave a mais faz menos diferença do que no exemplo com poucas chaves por página.

Esse tipo de árvore é especialmente interessante quando sua utilização é grande para acessos sequenciais e nesse caso a eficiência é muito maior e independe da altura da árvore.

### 3.7 Simulado

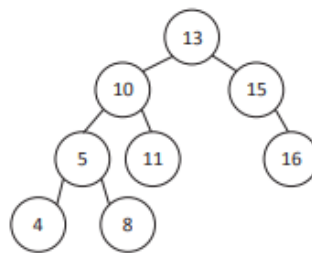
#### EXERCÍCIO 3.7-A

(Questão retirada de prova do Enade)

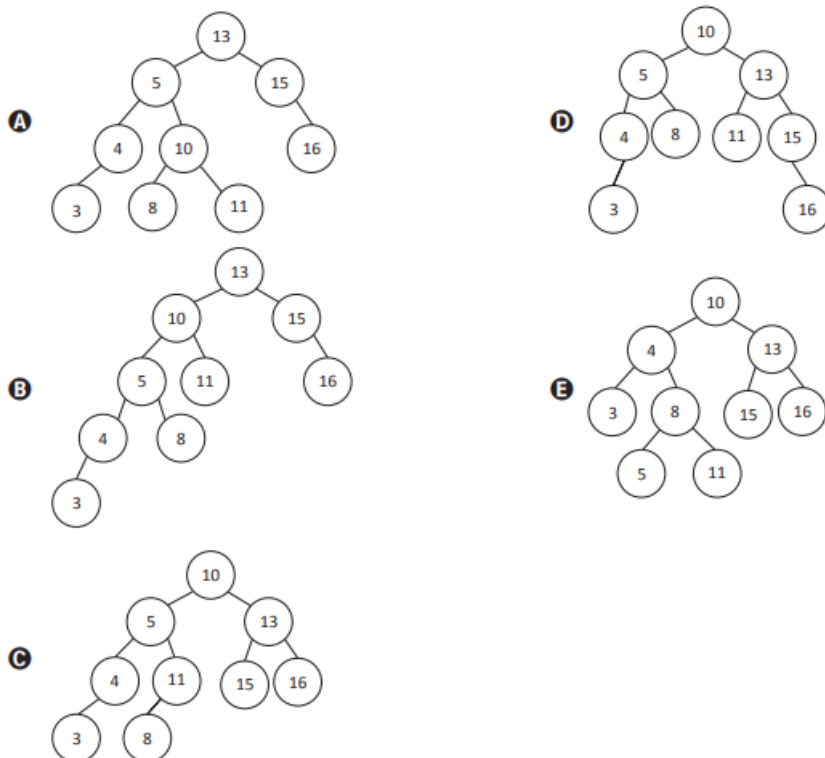
Uma árvore AVL é um tipo de árvore binária balanceada na qual a diferença entre as alturas de suas subárvores da esquerda e da direita não pode ser maior do que 1 para qualquer nó. Após a inserção de um nó em uma AVL, a raiz da subárvore de nível mais baixo no qual o novo nó foi inserido é marcada. Se a altura de seus filhos diferir em mais de uma unidade, é realizada uma rotação simples ou uma rotação dupla para igualar suas alturas.

LAFORE, R. *Data structures & algorithms in Java*. Indianópolis: Sams Publishing, 2003 (adaptado).

A seguir, é apresentado um exemplo de árvore AVL.



Pelo exposto no texto acima, após a inserção de um nó com valor 3 na árvore AVL exemplificada, é correto afirmar que ela ficará com a seguinte configuração:



### EXERCÍCIO 3.7-B

(Questão retirada de prova do Enade) Uma árvore binária de busca é uma árvore ordenada que pode apresentar prejuízos no desempenho de determinados algoritmos em função do desbalanceamento causado pela ordem de inserção dos elementos na estrutura. Uma árvore AVL é uma árvore binária de busca balanceada em que a diferença em módulo entre a altura da sub-árvore esquerda e a altura da sub-árvore direita de cada nó é, no máximo, de uma unidade.

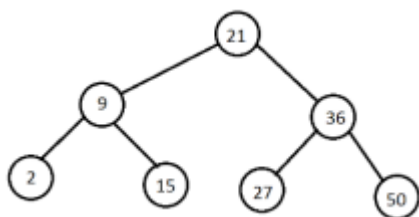
Nesse contexto, faça o que se pede nos itens a seguir.

- Apresente uma árvore binária de busca balanceada com os elementos 2, 9, 15, 21, 27, 36 e 50 em que o nó raiz principal contém o elemento 21 e o balanceamento de cada nó seja no máximo uma unidade.
- Considerando as inserções dos elementos 9, 27 e 50, nesta ordem, em uma árvore AVL inicialmente vazia, apresente a árvore resultante.

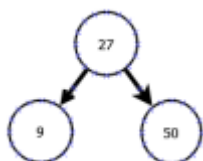


## RESPOSTAS:

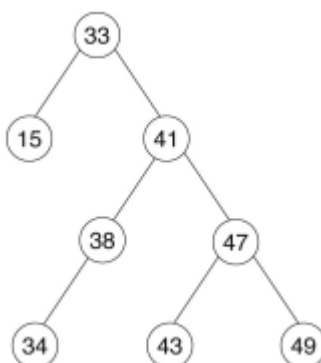
a)



b)

**EXERCÍCIO 3.7-C**

(Questão retirada de Prova Enade) Tendo como base a árvore a seguir, faça o que se pede nos itens a seguir:



- Considerando que o nó de valor 33 seja a raiz da árvore, descreva a ordem de visita para uma varredura em pré-ordem na árvore.
- Considerando que a árvore cuja raiz é o nó de valor 33 represente uma árvore de busca binária, desenhe a nova árvore que será obtida após a realização das seguintes operações: inserir um nó de valor 21; remover o nó de valor 47; inserir um nó de valor 48.



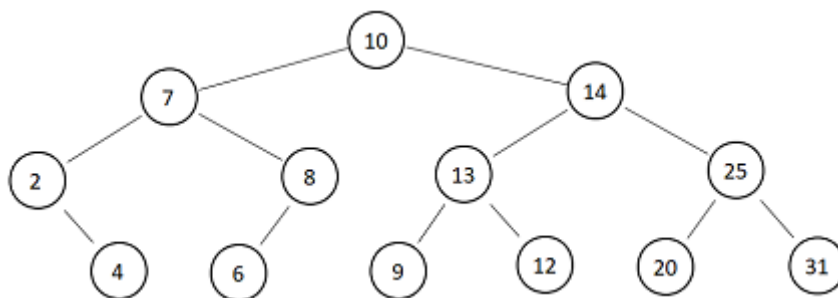
**EXERCÍCIO 3.7-D**

(2021 - Câmara de Marabá - PA - Técnico em Processamento de Dados) Seja T uma árvore balanceada do tipo AVL (Adelson-Velski e Landis) vazia. Supondo que os elementos 5, 10, 12, 8, 7, 11 e 13 sejam inseridos nessa ordem em T, a sequência que corresponde a um percurso de T em pré-ordem é

- a) 10, 8, 5, 7, 12, 11 e 13.
- b) 10, 7, 5, 8, 12, 11 e 13.
- c) 5, 7, 8, 10, 11, 12 e 13.
- d) 5, 8, 7, 11, 13, 12 e 10.
- e) 5, 10, 12, 8, 7, 11 e 13.

**EXERCÍCIO 3.7-E**

(2017 - IGP-SC - Perito Criminal em Informática) Considere a figura abaixo e assinale a alternativa que contém todas as afirmações corretas.



I. A figura representa uma Árvore Binária de Busca (BST – *Binary Search Tree*). Se ela for percorrida em ordem (*inorder*), a sequência de nodos visitados será: 2, 4, 7, 6, 8, 10, 9, 13, 12, 14, 20, 25, 31.

II. A figura representa uma Árvore Binária de Busca (BST – *Binary Search Tree*). Se ela for percorrida em pós-ordem (*posorder*), a sequência de nodos visitados será: 10, 7, 2, 4, 8, 6, 14, 13, 9, 12, 25, 20, 31.

III. A figura representa uma Árvore Binária de Busca (BST – *Binary Search Tree*). Se ela for percorrida em pré-ordem (*preorder*), a sequência de nodos visitados será: 4, 2, 6, 8, 7, 9, 12, 13, 20, 31, 25, 14, 10.

IV. A figura não representa uma Árvore Binária de Busca (BST – *Binary Search Tree*).

- a) Somente a I.
- b) I, II e III.
- c) Somente a IV.
- d) I e II.

## 4 SOLUÇÕES DOS EXERCÍCIOS E SIMULADOS

### SOLUÇÃO 1.1.1.2-A

Acrescentaremos ao final do Algoritmo, uma estrutura de repetição onde identificaremos qual o maior valor constante em VETVALOR. A seguir faremos uma PESQUISA SEQUENCIAL nesse mesmo vetor e listaremos os números das lojas que somaram esse maior valor, pois pode ser uma única ou mais do que uma com o mesmo valor. Para tal, acrescentaremos também uma nova variável, para guardar o maior valor.

**MAIOR-VAL: real**

<rotina a ser acrescentada ao final do Algoritmo>

```

MAIOR-VAL = VETVALOR[0]
para N de 1 até 50 faça
    se VETVALOR[N] > MAIOR-VAL
        MAIOR-VAL = VETVALOR[N]
    fim_se
fim_para

escreva ("LOJAS COM MAIOR FATURAMENTO - Valor = R$" MAIOR-VAL) // cabeçalho

para N de 0 até 50 faça // Pesquisa Sequencial
    se VETVALOR[N] = MAIOR-VAL
        escreva ("LOJA " N)
    fim_se
fim_para

```

### SOLUÇÃO 1.1.1.2-B

**Algoritmo vetorContaAniversariantesMes**

**variáveis**

**I: inteiro**

**NIVER: vetor[0..11] inteiro**

**CADASTRO: registro**

**início**

**NOME: caractere**

**DIA: inteiro**

**MÊS: inteiro**

**ANO: inteiro**

**fim\_registro**

**início**

```

para I de 0 até 11 faça
    NIVER[I] = 0
fim_para

abrir CADASTRO

enquanto .não. EOF faça
    leia (MÊS)
    NIVER[MÊS - 1] = NIVER[MÊS - 1] + 1
fim_enquanto

fechar CADASTRO

escreva ("Quantidade de Aniversariantes por mês")
para I de 1 até 12 faça
    escreva (I, " = " NIVER[I - 1])
fim_para

```

### SOLUÇÃO 1.1.1.5-A

Algoritmo vetorContaLetrasAlfabeto

```

variáveis
    CONT, L, LET: inteiro
    TEXTO: vetor[0..99] caractere // Já preenchido com um texto
    ALFABETO: vetor[0..25] caractere // preenchido com "ABCDE...XYZ"

início

para L de 0 até 25 faça
    CONT = 0
    para LET de 0 até 99 faça
        se TEXTO[LET] = ALFABETO[L]
            CONT = CONT + 1
        fim_se
    fim_para
    escreva ("A letra " ALFABETO[L] " aparece " CONT " vezes.")
fim_para

fim

```



## Entendendo

Repetimos a contagem no texto para cada letra do alfabeto, utilizando um único contador, com o cuidado de zerar dentro do looping que controla a letra que está sendo contada. A apresentação do total também tem que estar dentro desse looping externo, após a totalização da letra.

### SOLUÇÃO 1.1.2.2-A

Algoritmo matrizContaLetraA

variáveis

CONT, COL, LIN: inteiro

TEXTO: matriz[0..99, 0..39] caractere // Já preenchido com um texto

início

CONT = 0

para COL de 0 até 99 faça

para LIN de 0 até 39 faça

se TEXTO[COL, LIN] = "A"

CONT = CONT + 1

fim\_se

fim\_para

fim\_para

escreva ("Quantidade de A que aparece no texto:" CONT)

fim

### SOLUÇÃO 1.1.2.2-B

Algoritmo matrizContaLetrasAlfabeto

variáveis

CONT, L, COL, LIN: inteiro

TEXTO: matriz[0..99, 0..39] caractere // Já preenchido com um texto

ALFABETO: vetor[0..25] caractere // preenchido com "ABCDE...XYZ"

início

para L de 0 até 25 faça

CONT = 0

para COL de 0 até 99 faça

para LIN de 0 até 39 faça

se TEXTO[COL, LIN] = ALFABETO[L]

CONT = CONT + 1

fim\_se

fim\_para

fim\_para

escreva ("A letra " ALFABETO(L) " aparece " CONT " vezes.")

fim\_para

fim

## SOLUÇÃO 1.1.5-A

Ao final da execução da linha 12:										
Vetor A										
Posição	1	2	3	4	5	6	7	8	9	10
Valor	2	2	6	4	10	6	14	8	18	10
Vetor B										
Posição	1	2	3	4	5	6	7	8	9	10
Valor	0	0	0	0	0	0	0	0	0	0
Ao final da execução da linha 19:										
Vetor A										
Posição	1	2	3	4	5	6	7	8	9	10
Valor	1	2	3	4	5	6	7	8	9	10
Vetor B										
Posição	1	2	3	4	5	6	7	8	9	10
Valor	2	0	4	0	6	0	8	0	10	0

## SOLUÇÃO 1.1.5-B

RESPOSTA: b) 6, 10, 9

## SOLUÇÃO 1.1.5-C

RESPOSTA: a) Um índice para cada uma de suas dimensões.

## SOLUÇÃO 2.2.7-A

### Algoritmo ChecaExpressãoDinâmica

variáveis

I: inteiro

RESULTADO, AUX: caractere

EXPRESSAO: vetor [0..49] inteiro // Já preenchido com uma expressão matemática

Início

Cria Pilha

TOPO = Null

RESULTADO = "OK"

I = 0

Enquanto i < 49 .e. RESULTADO = "OK" faça

se EXPRESSAO[i] = '{'

Cria novo objeto

INFO[THIS] = '}'

```

        NEXT[THIS] = TOPO
        TOPO = THIS
    fim_se
    se EXPRESSAO[i] = '['
        Cria novo objeto
        INFO[THIS] = ']'
        NEXT[THIS] = TOPO
        TOPO = THIS
    fim_se
    se EXPRESSAO[i] = '('
        Cria novo objeto
        INFO[THIS] = ')'
        NEXT[THIS] = TOPO
        TOPO = THIS
    fim_se

    se EXPRESSAO[i] = '}' .ou. EXPRESSAO[i] = ']' .ou. EXPRESSAO[i] = ')'
        se TOPO = Null
            RESULTADO = "INCORRETA"
        senão
            se EXPRESSAO[i] = INFO[TOPO]
                AUX = TOPO
                TOPO = NEXT[TOPO]
                deleta[AUX]
            senão
                RESULTADO = "INCORRETA"
            fim_se
        fim_se
    fim_se

    I = I + 1

fim_enquanto
se IND > 0
    RESULTADO = " INCORRETA"
fim_se
escreva (RESULTADO)
fim

```

### SOLUÇÃO 2.3.6-A

#### Algoritmo FilaDinâmicaCompleta

variáveis

OPÇÃO: inteiro

Início

faça

```

    escreva ("OPÇÕES:" _
    escreva (" 1 – INCLUIR CLIENTE")
    escreva (" 2 – EXCLUIR CLIENTE")
    escreva (" 3 – LISTAR CLIENTES")
    escreva (" 9 – SAIR")

```

```

escreva ("O que deseja fazer : X ")
leia (OPÇÃO)
se OPÇÃO = 1
    chama <INCLUIR-FILA>
senão
    se OPÇÃO = 2
        chama <EXCLUIR-FILA>
    senão
        se OPÇÃO = 3
            chama <LISTAR-FILA>
        senão
            se OPÇÃO <> 9
                escreva ("OPÇÃO INVÁLIDA")
            fim_se
        fim_se
    fim_se
fim_se
enquanto OPÇÃO <> 9
fim

```

### INCLUIR-FILA

```

Cria objeto
escreva("Informe o valor a ser inserido na Fila ")
leia (INFO[THIS])
NEXT[THIS] = Null
se START = Null      // indica que a fila estava vazia
    START = THIS
senão
    NEXT[END] = THIS
fim_se
END = THIS

```

### EXCLUIR-FILA

```

se START = NULL
    escreva ("ARQUIVO VAZIO")
senão
    AUX = START
    START = NEXT[START]
    se START = Null      // indica que a fila ficará vazia
        END = Null
    fim_se
    deleta[AUX]
fim_se

```

### LISTAR-FILA

```

se START = NULL
    escreva ("ARQUIVO VAZIO")
senão
    AUX = START
    enquanto AUX <> Null faça
        escreva (INFO[AUX])
        AUX = NEXT[AUX]

```



```

        fim_enquanto
    fim_se

```

## SOLUÇÃO 2.4.5-A

### Algoritmo ListaDinâmicaCompleta

variáveis

OPÇÃO: inteiro  
EXCL: caractere  
AUX, AUX2: ponteiro

Início

faça

```

    escreva ("OPÇÕES:")
    escreva (" 1 – INCLUIR UM ALUNO")
    escreva (" 2 – EXCLUIR UM ALUNO")
    escreva (" 3 – EMITIR LISTAGEM DE ALUNOS")
    escreva (" 9 – SAIR")
    escreva ("O que deseja fazer : X ")
    leia (OPÇÃO)
    se OPÇÃO = 1
        chama INCLUIR-LISTA
    senão
        se OPÇÃO = 2
            chama EXCLUIR- LISTA
        senão
            se OPÇÃO =3
                chama LISTAR- LISTA
            senão
                se OPÇÃO <> 9
                    escreva ("OPÇÃO INVÁLIDA")
                fim_se
            fim_se
        fim_se
    fim_se
    enquanto OPÇÃO <> 9
    fim

```

### INCLUIR-LISTA

```

    escreva ("Informe o nome a ser inserido na Lista")
    Cria objeto
    leia (INFO[THIS])
    se START = Null // indica que a fila está vazia
        chama <INCLUIR-PRIMEIRO-LISTA>
    senão
        se INFO[THIS] < INFO[START]
            chama <INCLUIR-NO-INÍCIO>
        senão
            se INFO[THIS] > INFO[END]
                chama <INCLUIR-NO-FINAL>
            senão
                chama <INCLUIR-NO-MEIO>
        fim_se
    fim_se

```

```

    fim_se
fim_se

```

#### INCLUIR-PRIMEIRO-LISTA

```

    NEXT[THIS] = Null
    START = THIS
    END = THIS

```

#### INCLUIR-NO-INÍCIO

```

    NEXT[THIS] = START
    START = THIS

```

#### INCLUIR-NO-FINAL

```

    NEXT[THIS] = Null
    NEXT[END] = THIS
    END = THIS

```

#### INCLUIR-NO-MEIO

```

    AUX = START
    enquanto INFO[THIS] > INFO[NEXT[AUX]] faça
    AUX = NEXT[AUX]
    fim_enquanto
    NEXT[THIS] = NEXT[AUX]
    NEXT[AUX] = THIS

```

#### EXCLUIR-LISTA

```

se START = Null
    escreva ("ARQUIVO VAZIO")
senão
    escreva ("Informe Nome a ser excluído")
    leia (EXCL)
    se EXCL < INFO[START] .ou. EXCL > INFO[END]
        escreva ("NOME NÃO CONSTA NA LISTA")
    senão
        se EXCL = INFO[START]
            chama <EXCLUIR-PRIMEIRO-LISTA>
        senão
            se EXCL = INFO[END]
                chama <EXCLUIR-FINAL-LISTA>
            senão
                chama <EXCLUIR-MEIO-LISTA>
        fim_se
    fim_se
fim_se

```

#### EXCLUIR-PRIMEIRO-LISTA

```

    AUX = START
    START = NEXT[START]
    se START = Null
        END = Null
    fim_se

```

// igual ao EXCLUIR-FILA

// indica que a fila ficará vazia

deleta[AUX]

### EXCLUIR-FINAL-LISTA

```
AUX = START
enquanto NEXT[AUX] <> END faça    // localiza o penúltimo
    AUX = NEXT[AUX]
fim_enquanto
NEXT[AUX] = Null
Deleta[END]
END = AUX
```

### EXCLUIR-MEIO-LISTA

```
AUX = START
enquanto EXCL > INFO[NEXT[AUX]] faça
    AUX = NEXT[AUX]
fim_enquanto
se EXCL = INFO[NEXT[AUX]]
    AUX2 = NEXT[AUX]
    NEXT[AUX] = NEXT[NEXT[AUX]]
    deleta[AUX2]
senão
    escreva ("NOME NÃO CONSTA NA LISTA")
fim_se
```

### LISTAR-LISTA

```
se START = Null
    escreva ("ARQUIVO VAZIO")
senão
    AUX = START
    enquanto AUX <> Null faça
        escreva (INFO[AUX])
        AUX = NEXT[AUX]
    fim_enquanto
fim_se
```

## SOLUÇÃO 2.5.5-A

### Algoritmo ListaDuplaDinâmicaCompleta

variáveis

OPÇÃO: inteiro  
EXCL: caractere  
AUX, AUX2: ponteiro

Início

faça

```
    escreva ("OPÇÕES:")
    escreva (" 1 – INCLUIR UM ALUNO")
    escreva (" 2 – EXCLUIR UM ALUNO")
```

```

escreva (" 3 – EMITIR LISTAGEM DE ALUNOS (ordem crescente)")
escreva (" 4 – EMITIR LISTAGEM DE ALUNOS (ordem decrescente)")
escreva (" 9 – SAIR")
escreva ("O que deseja fazer : X ")
leia (OPÇÃO)
se OPÇÃO = 1
    chama <INCLUIR-LISTA-DUPLA>
senão
    se OPÇÃO = 2
        chama <EXCLUIR- LISTA-DUPLA>
    senão
        se OPÇÃO =3
            chama <LISTAR-ORDEM-CRESCENTE>
        senão
            se OPÇÃO =4
                chama <LISTAR-ORDEM-DECRESCENTE>
            senão
                se OPÇÃO <> 9
                    escreva ("OPÇÃO INVÁLIDA")
                fim_se
            fim_se
        fim_se
    fim_se
    fim_se
    enquanto OPÇÃO <> 9
fim

```

### INCLUIR-LISTA-DUPLA

```

escreva ("Informe o nome a ser inserido na Lista")
cria objeto
leia (INFO[THIS])
se START = Null // indica que a fila está vazia
    chama <INCLUIR-PRIMEIRO-LISTA-DUPLA>
senão
    se INFO[THIS] < INFO[START]
        chama <INCLUIR-NO-INÍCIO-LISTA-DUPLA>
    senão
        se INFO[THIS] > INFO[END]
            chama <INCLUIR-NO-FINAL-LISTA-DUPLA>
        senão
            chama <INCLUIR-NO-MEIO-LISTA-DUPLA>
        fim_se
    fim_se
fim_se

```

### INCLUIR-PRIMEIRO-LISTA-DUPLA

```

NEXT[THIS] = Null
PREV[THIS] = Null
START = THIS
END = THIS

```

**INCLUIR-NO-INÍCIO-LISTA-DUPLA**

```

PREV(THIS) = Null
PREV(START) = THIS
NEXT[THIS] = START
START = THIS

```

**INCLUIR-NO-FINAL-LISTA-DUPLA**

```

PREV[THIS] = END
NEXT[THIS] = Null
NEXT[END] = THIS
END = THIS

```

**INCLUIR-NO-MEIO-LISTA-DUPLA**

```

AUX = START
enquanto INFO[THIS] > INFO[NEXT[AUX]] faça
AUX = NEXT[AUX]
fim_enquanto
NEXT[THIS] = NEXT[AUX]
PREV[THIS] = AUX
NEXT[AUX] = THIS
PREV[NEXT[THIS]] = THIS

```

## &lt;OUTRA OPÇÃO&gt;

```

AUX = START
enquanto INFO[THIS] > INFO[AUX] faça
AUX = NEXT[AUX]
fim_enquanto
NEXT[THIS] = AUX
PREV[THIS] = PREV[AUX]
NEXT[PREV[THIS]] = THIS
PREV[AUX] = THIS

```

**EXCLUIR-LISTA-DUPLA**

```

se START = Null
    escreva ("ARQUIVO VAZIO")
senão
    escreva ("Informe Nome a ser excluído")
    leia (EXCL)
    se EXCL < INFO[START] .ou. EXCL > INFO[END]
        escreva ("NOME NÃO CONSTA NA LISTA")
    senão
        se EXCL = INFO[START]
            chama <EXCLUIR-PRIMEIRO-LISTA-DUPLA>
        senão
            se EXCL = INFO[END]
                chama <EXCLUIR-FINAL-LISTA-DUPLA>
            senão
                chama <EXCLUIR-MEIO-LISTA-DUPLA>
        fim_se
    fim_se
fim_se

```

**EXCLUIR-PRIMEIRO-LISTA-DUPLA**

// igual ao EXCLUIR-FILA

```

AUX = START
START = NEXT[START]
se START = Null
END = Null
fim_se
deleta[AUX]

```

// indica que a fila ficará vazia

**EXCLUIR-FINAL-LISTA-DUPLA**

```

AUX = START
enquanto NEXT[AUX] <> END faça // localiza o penúltimo
AUX = NEXT[AUX]
fim_enquanto
NEXT[AUX] = Null
Deleta[END]
END = AUX

```

**EXCLUIR-MEIO-LISTA-DUPLA**

```

AUX = START
enquanto EXCL > INFO[NEXT[AUX]] faça
AUX = NEXT[AUX]
fim_enquanto
se EXCL = INFO[NEXT[AUX]]
    AUX2 = NEXT[AUX]
    NEXT[AUX] = NEXT[NEXT[AUX]]
    deleta[AUX2]
senão
    escreva ("NOME NÃO CONSTA NA LISTA")
fim_se

```

**LISTAR-ORDEM-CRESCENTE**

// igual à Lista Simples e à Fila

```

se START = Null
    escreva ("ARQUIVO VAZIO")
senão
    AUX = START
    enquanto AUX <> Null faça
        escreva (INFO[AUX])
        AUX = NEXT[AUX]
    fim_enquanto
fim_se

```

**LISTAR-ORDEM-DECRESCENTE**

```

se START = Null
    escreva ("ARQUIVO VAZIO")
senão
    AUX = END
    enquanto AUX <> Null faça
        escreva (INFO[AUX])
        AUX = PREV[AUX]

```

```

fim_se
fim_enquanto

```

### SOLUÇÃO 2.7-A

Resposta: a) P – I – S – 2019 – F

### SOLUÇÃO 2.7-B

RESPOSTA: a) I e II

### SOLUÇÃO 2.7-C

RESPOSTA: c) a pilha: tipo especial de lista encadeada, na qual o último objeto a ser inserido é o primeiro a ser lido; nesse mecanismo, conhecido como estrutura FIFO (*First In – First Out*), a inserção e a remoção são feitos na mesma extremidade e a estrutura deve possuir um nó com a informação e um apontador para o próximo nó.

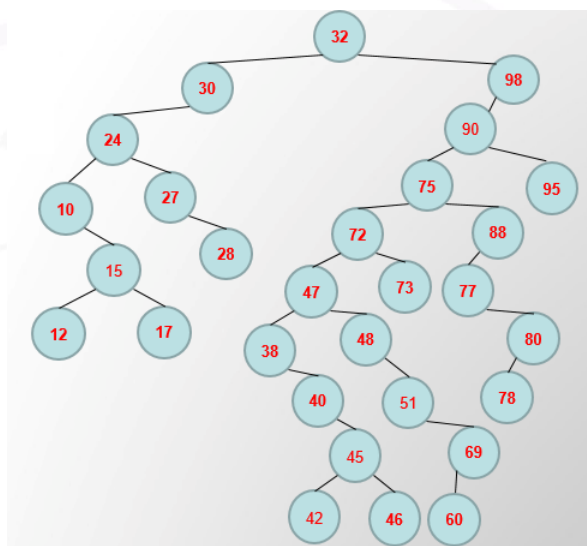
### SOLUÇÃO 2.7-D

RESPOSTA: e) Lista circular

### SOLUÇÃO 2.7-E

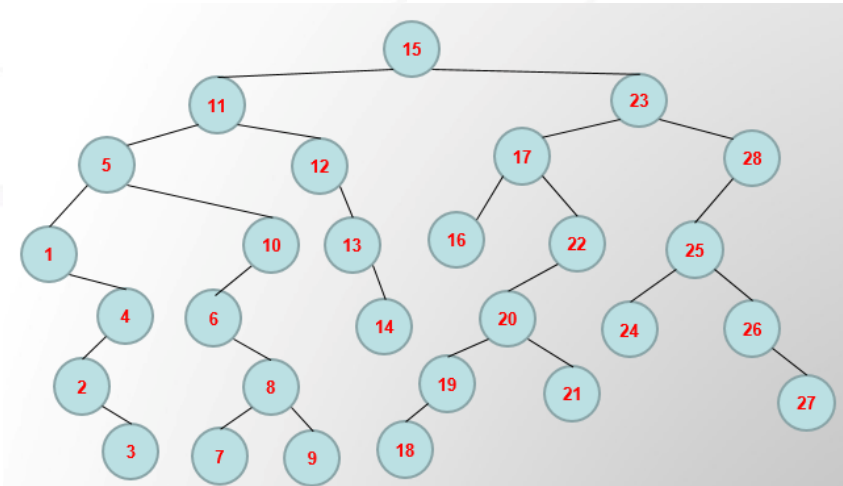
RESPOSTA: e) um vetor ordenado

### SOLUÇÃO 3.2.5-A

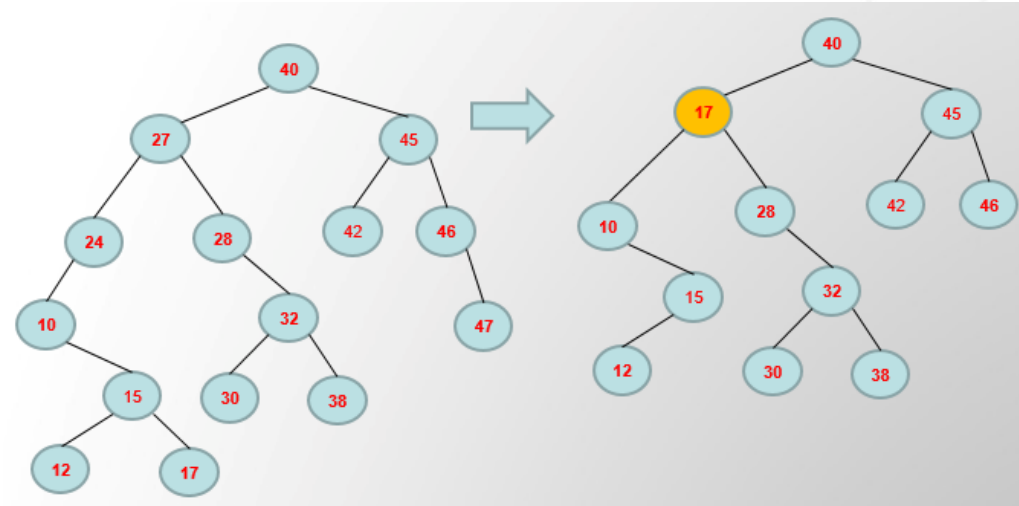




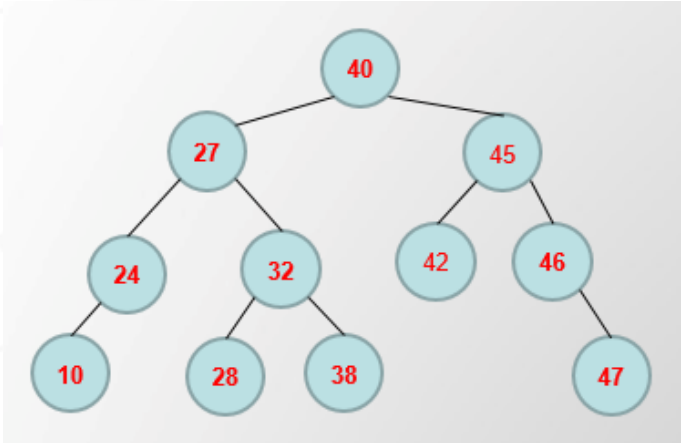
SOLUÇÃO 3.2.5-B



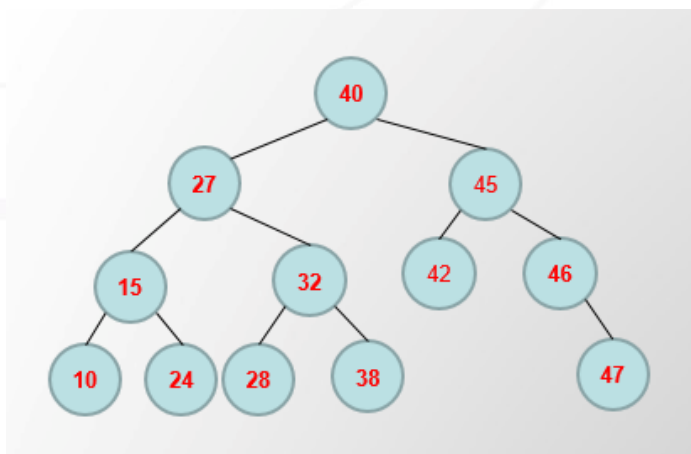
SOLUÇÃO 3.2.5-C



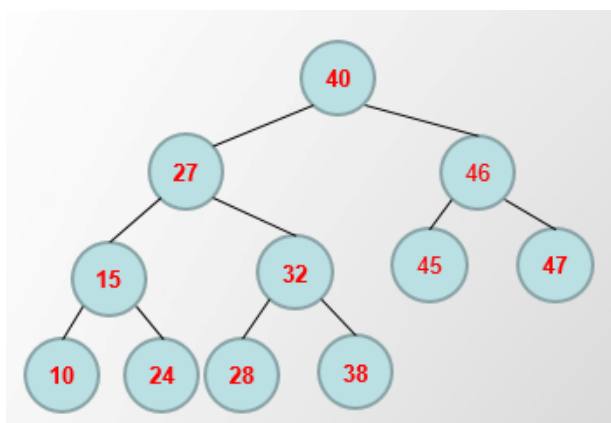
SOLUÇÃO 3.3.3-A



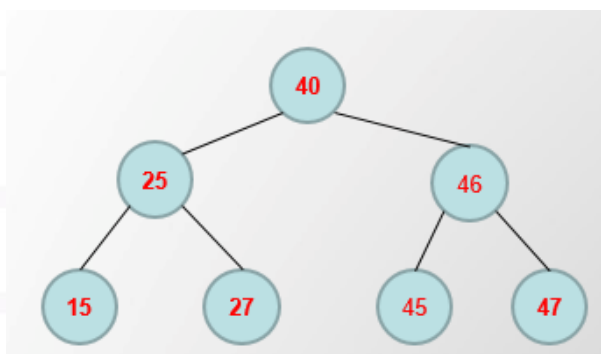
### SOLUÇÃO 3.3.3-B



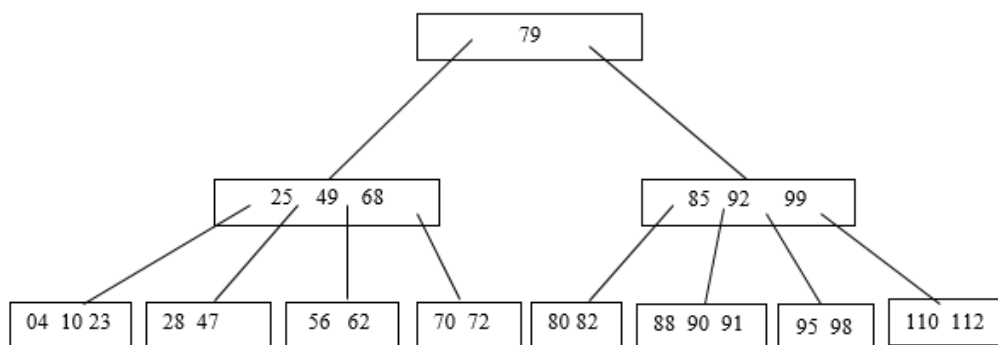
### SOLUÇÃO 3.3.3-C



### SOLUÇÃO 3.3.3-D

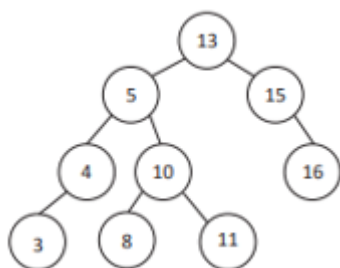


### SOLUÇÃO 3.4.5-A



### SOLUÇÃO 3.7-A

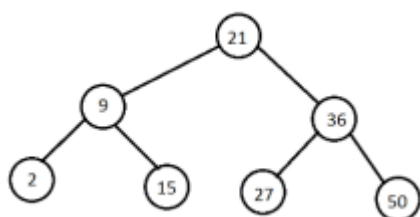
RESPOSTA: a)



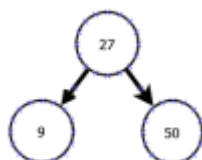
### SOLUÇÃO 3.7-B

RESPOSTAS:

a)

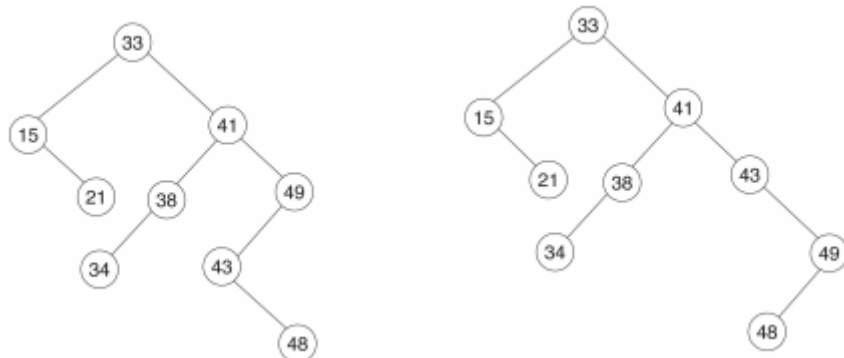


b)



**SOLUÇÃO 3.7-C**

- a) A descrição correta do percurso é: 33, 15, 41, 38, 34, 47, 43, 49.  
b) As duas soluções a seguir são válidas:

**SOLUÇÃO 3.7-D**

RESPOSTA: b) 10, 7, 5, 8, 12, 11 e 13.

**SOLUÇÃO 3.7-E**

RESPOSTA: c)