

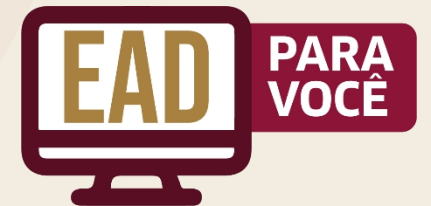


dted

DIRETORIA DE TECNOLOGIAS
NA EDUCAÇÃO

Estrutura de dados: Árvores binárias

Sérgio Souza Costa



Árvores binárias: Definição

Uma árvore binária T é um conjunto finito de elementos, denominados nós ou vértices, tal que:

- $T = \emptyset$, quando a árvore é dita vazia, ou
- $T = \{r\} \cup \{T_e\} \cup \{T_d\}$

Nesta definição:

- r é um nó especial chamado raiz
- Os demais nós são um conjunto vazio ou são conjuntos disjuntos T_e e T_d , chamados subárvore à esquerda de T e subárvore à direita de T , cada qual uma árvore binária.

Árvores binárias: Implementação

Árvore binária pode ser implementada em Python como a seguir:

```
class Arvore:
    def __init__(self, info, se=None, sd=None) :
        self.info = info
        self.se = se
        self.sd = sd

    def __str__(self) :
        return str(self.info)

a = Arvore (10,Arvore (30, Arvore ( 40,None, Arvore (60) )), Arvore (20))
```

Árvores binárias: Implementação

Árvore binária pode ser implementada em Python como a seguir:

```
class Arvore:
    def __init__(self, info, se=None, sd=None) :
        self.info = info
        self.se = se
        self.sd = sd

    def __str__(self) :
        return str(self.info)

a = Arvore (10,Arvore (30, Arvore ( 40,None, Arvore (60) )), Arvore (20))
```

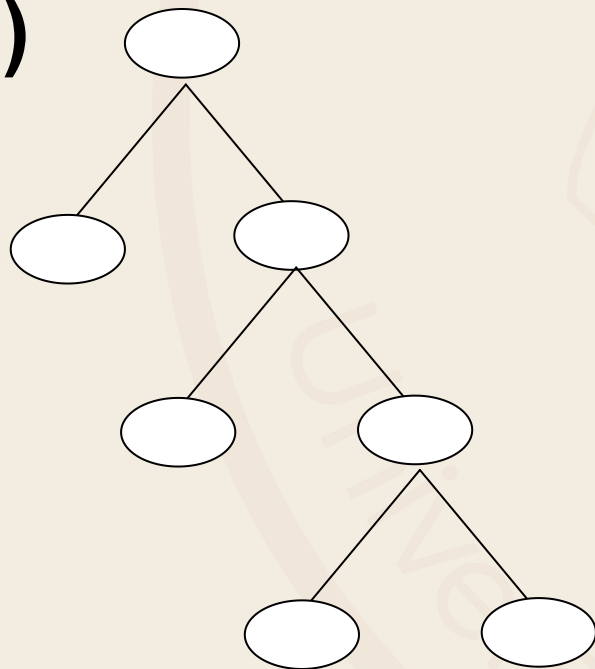
Que ár

Que árvore é essa ?

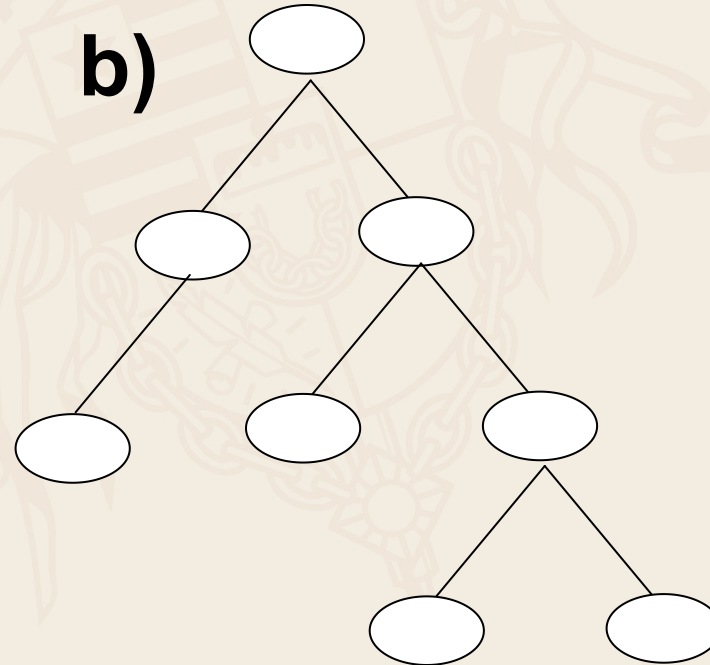
Árvores binárias: Tipos

Árvore estritamente binária: cada nó tem grau 0 ou 2, ou seja, todo nó tem 0 ou 2 filho

a)

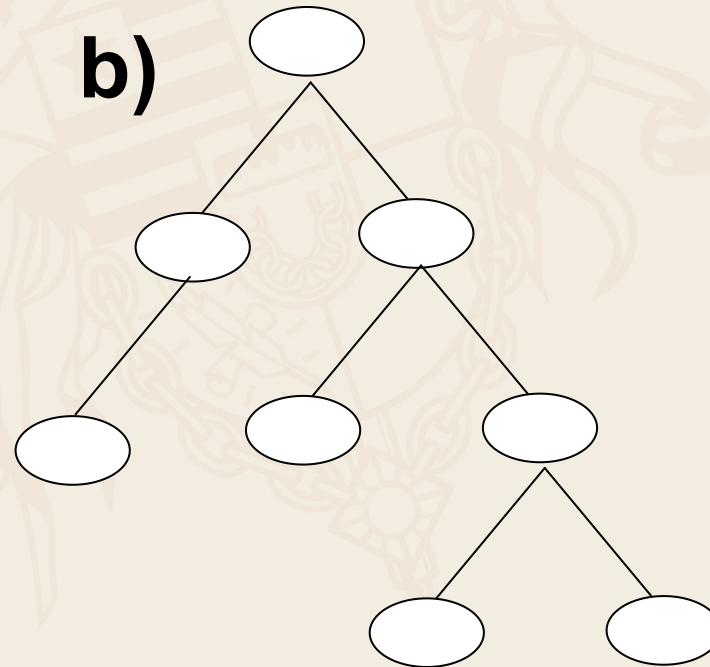
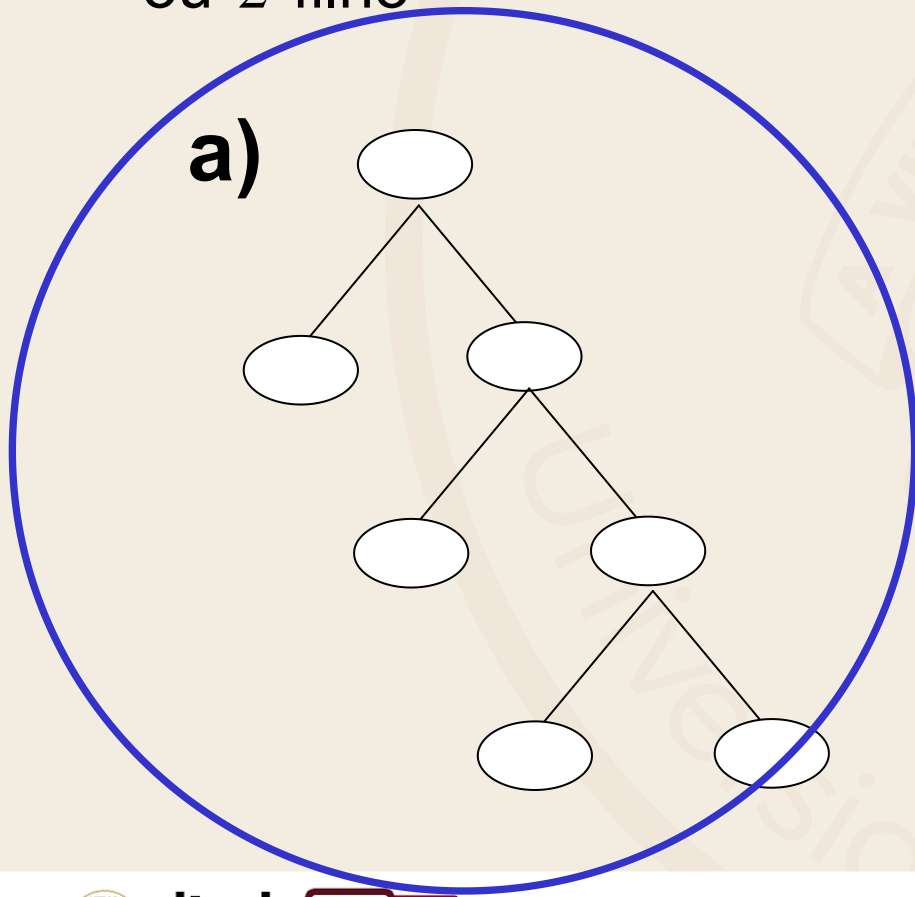


b)



Árvores binárias: Tipos

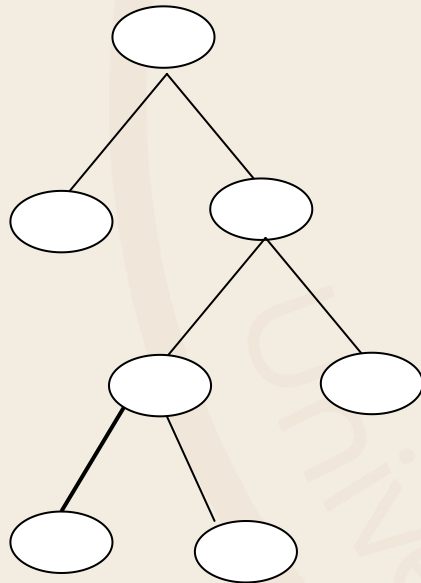
Árvore estritamente binária: cada nó tem grau 0 ou 2, ou seja, todo nó tem 0 ou 2 filho



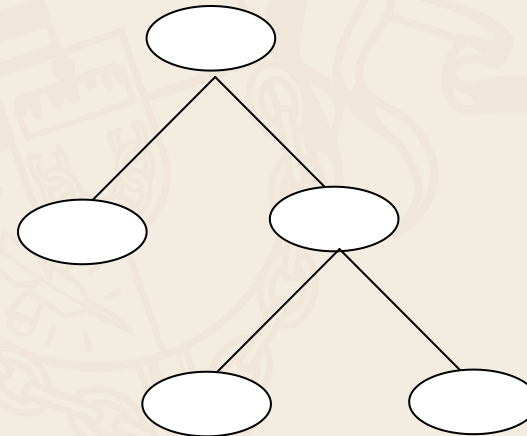
Árvores binárias: Tipos

Árvore binária completa é uma árvore estritamente binária na qual todo nó que apresente alguma sub-árvore vazia está localizado no último ou no penúltimo nível da árvore

a)



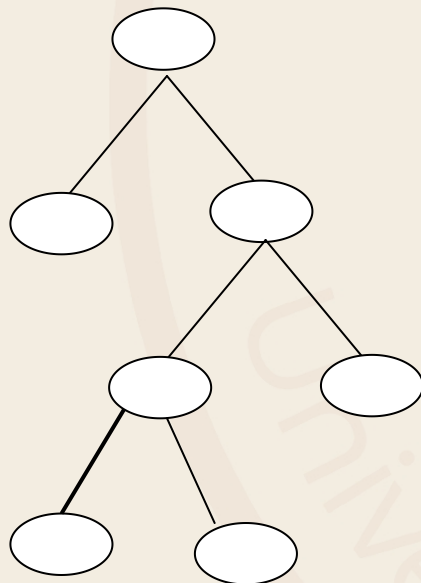
b)



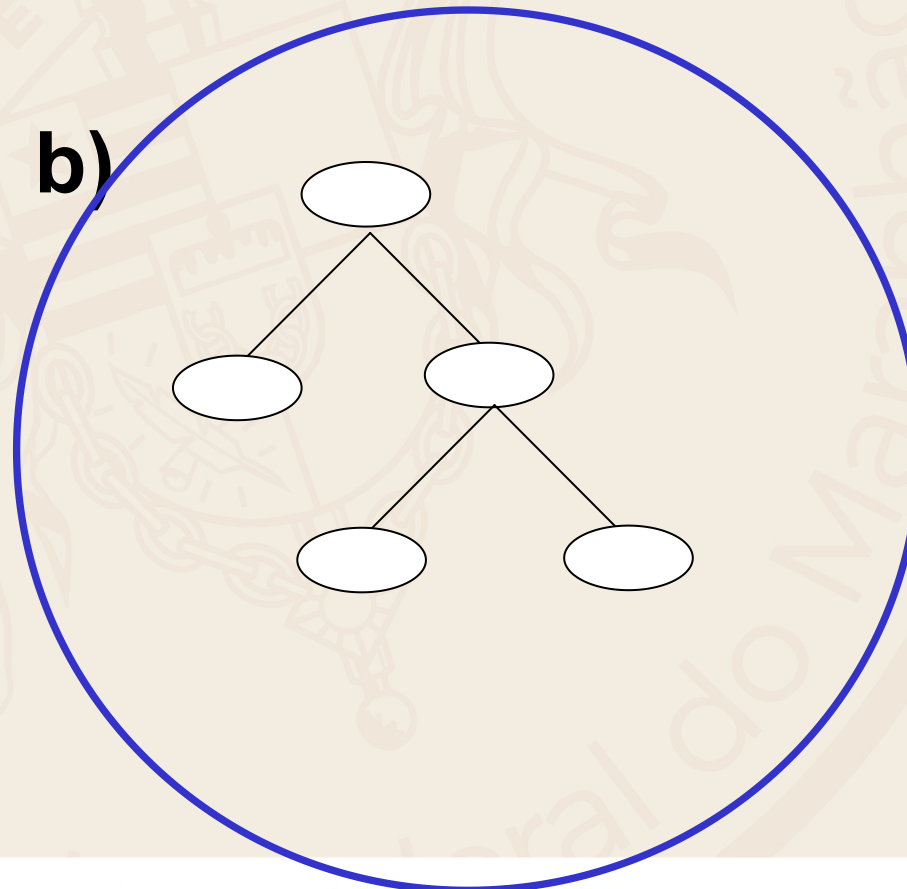
Árvores binárias: Tipos

Árvore binária completa é uma árvore estritamente binária na qual todo nó que apresente alguma sub-árvore vazia está localizado no último ou no penúltimo nível da árvore

a)



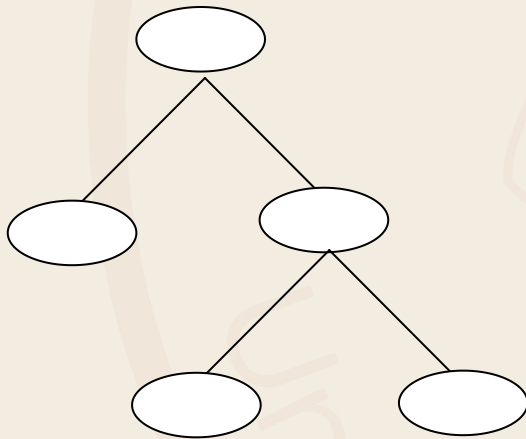
b)



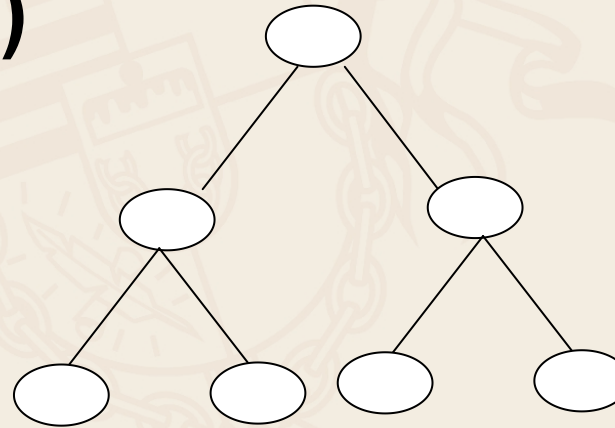
Árvores binárias: Tipos

Árvore binária cheia quando todos os nós internos tem grau 2 e todas as folhas estão no mesmo nível

a)



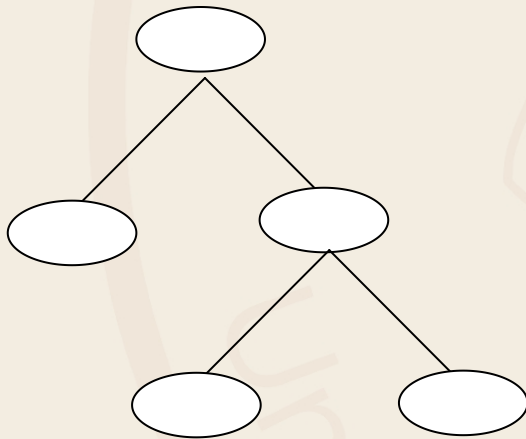
b)



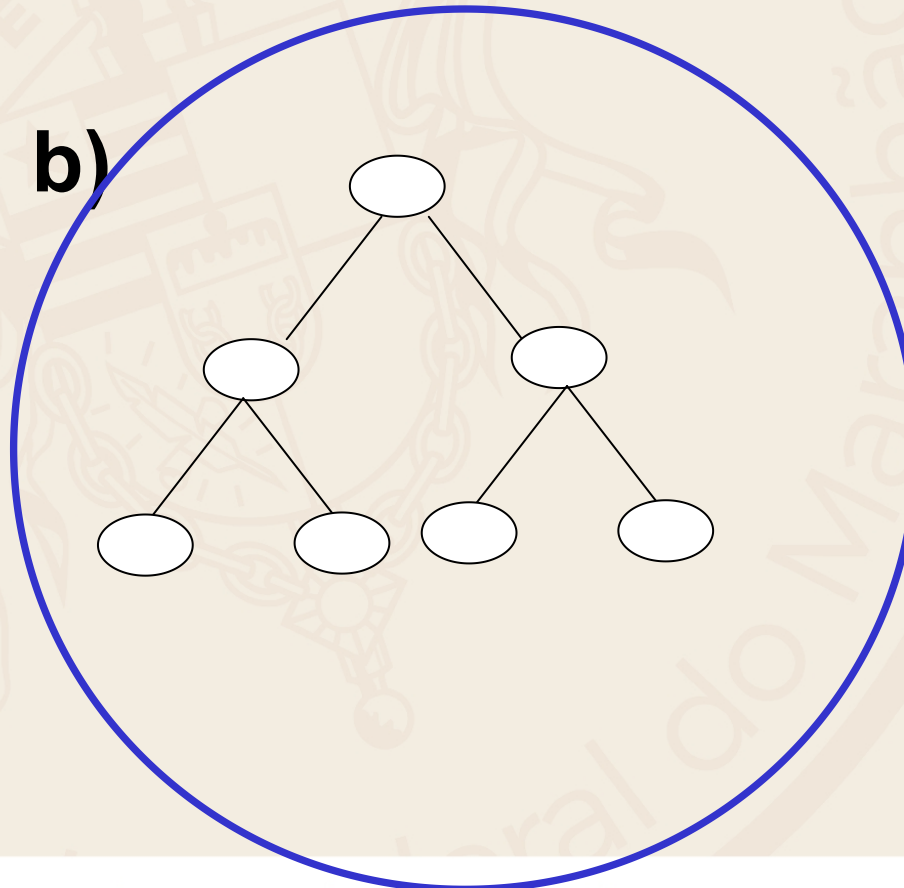
Árvores binárias: Tipos

Árvore binária cheia quando todos os nós internos tem grau 2 e todas as folhas estão no mesmo nível

a)



b)



Árvores binárias: Tipos

Árvore binária cheia quando todos os nós internos tem grau 2 e todas as folhas estão no mesmo nível

Árvore binária cheia: propriedade

- O número de nós no nível i é 2^i
- Sendo k o nível máximo da árvore, então o número total de nós é $2^{k+1}-1$

Pergunta:

Dado que a árvore binária cheia tenha n nós, qual o nível da árvore? $\log_2(n+1)-1$

Árvores binárias: Percurso

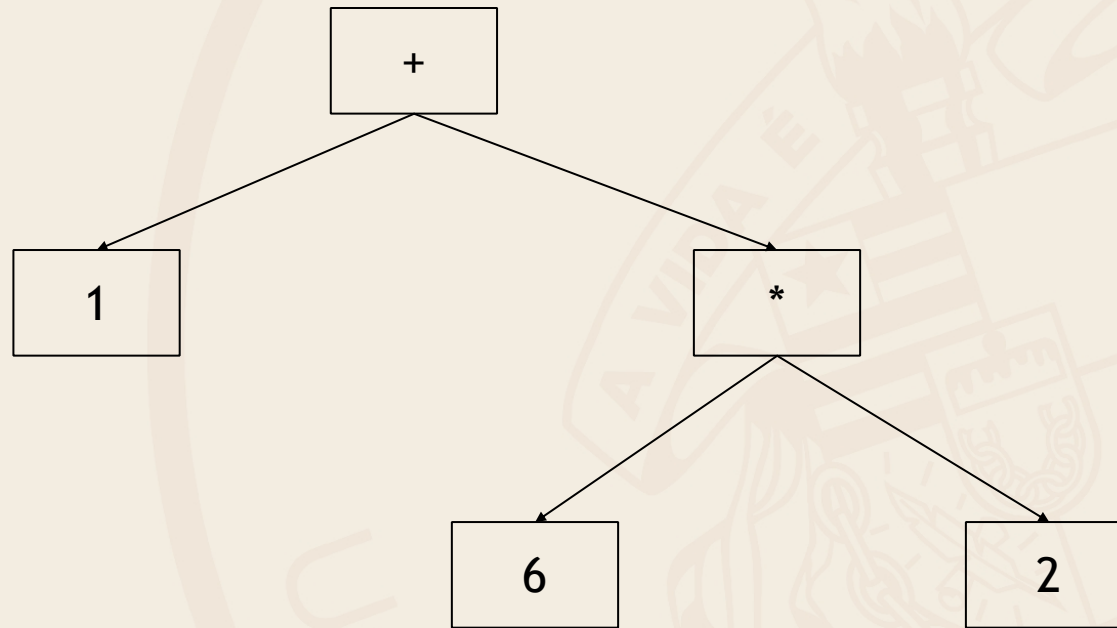
Percorrer uma árvore binária “visitando” cada nó uma única vez. Visitar pode ser:

- Imprimir o valor do nó
- Alterar o valor do nó
- Etc

Não existe um único percurso para árvores (binárias ou não). Diferentes percursos podem ser realizados, dependendo da aplicação.

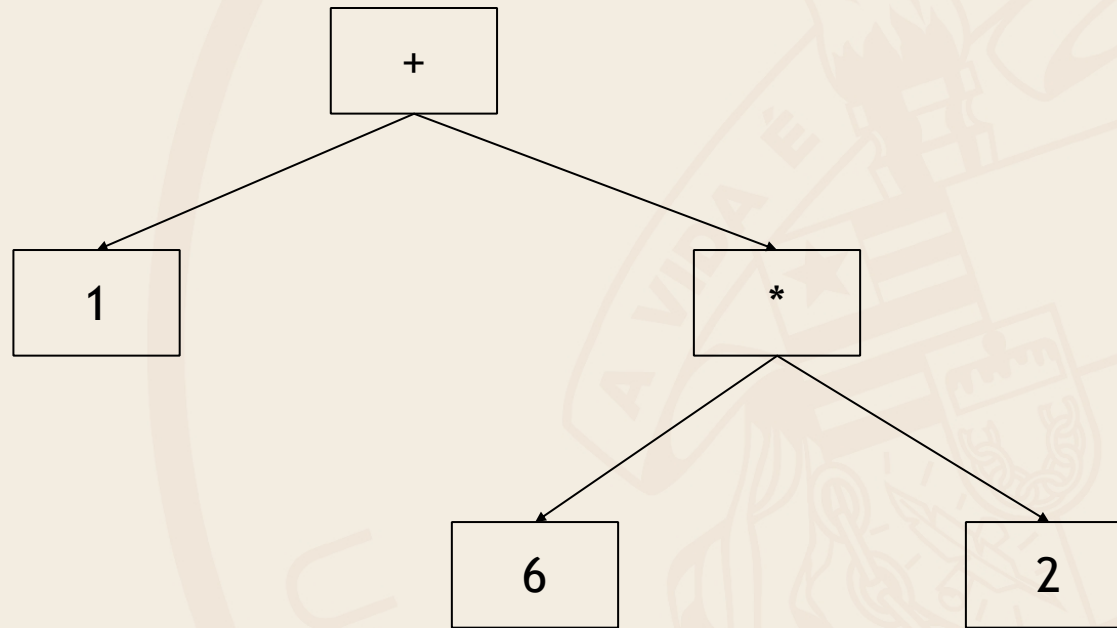
Há três percursos básicos: pré-ordem, em-ordem e pós-ordem.

Árvores binárias: pré-ordem



1. Visitar o nó raiz
2. Percorrer a subárvore à esquerda
3. Percorrer a subárvore à direita

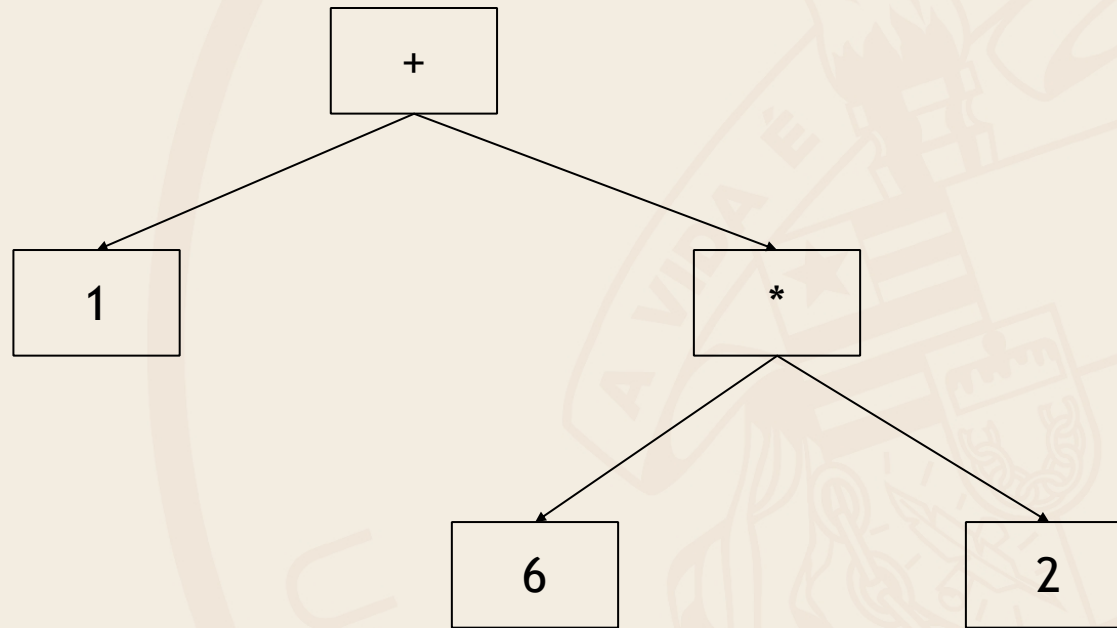
Árvores binárias: pré-ordem



1. Visitar o nó raiz
2. Percorrer a subárvore à esquerda
3. Percorrer a subárvore à direita

+1*62

Árvores binárias: pré-ordem

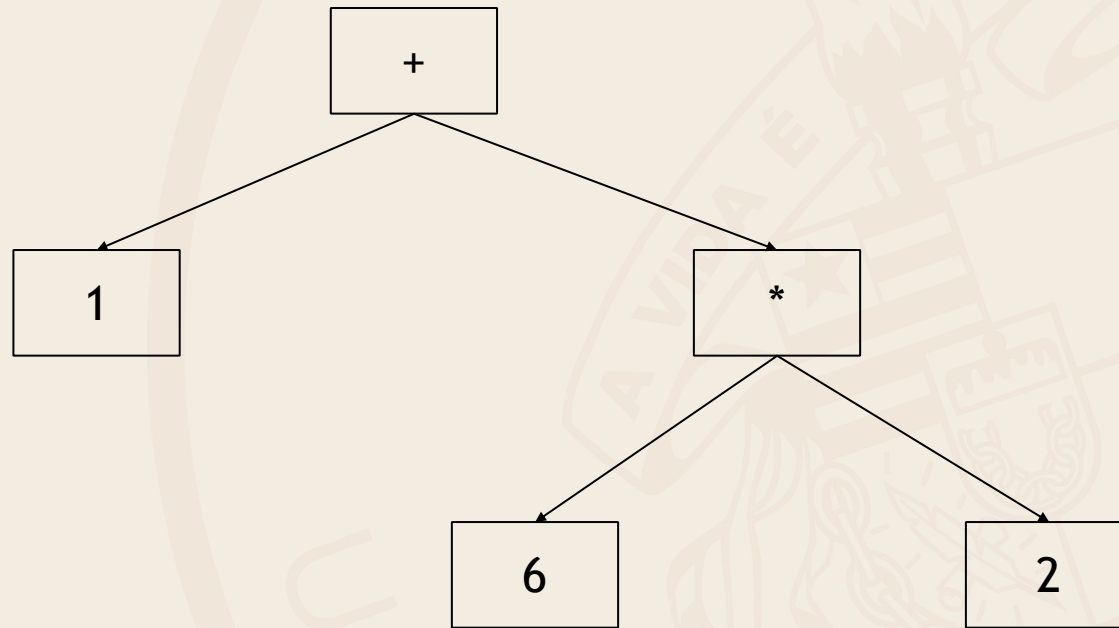


1. Visitar o nó raiz
2. Percorrer a subárvore à esquerda
3. Percorrer a subárvore à direita

```
def preOrdem (arvore):  
    if arvore != None:  
        print (arvore.info)  
        preOrdem (arvore.se)  
        preOrdem (arvore.sd)
```

+1*62

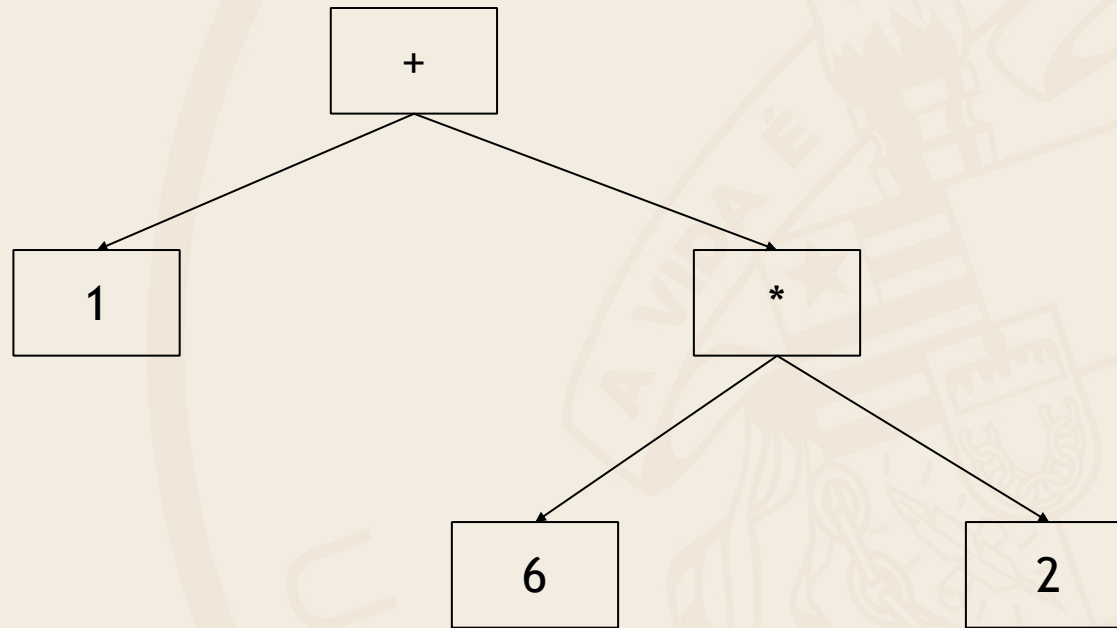
Árvores binárias: em-ordem



1. Percorrer a subárvore à esquerda
2. Visitar o nó raiz
3. Percorrer a subárvore à direita

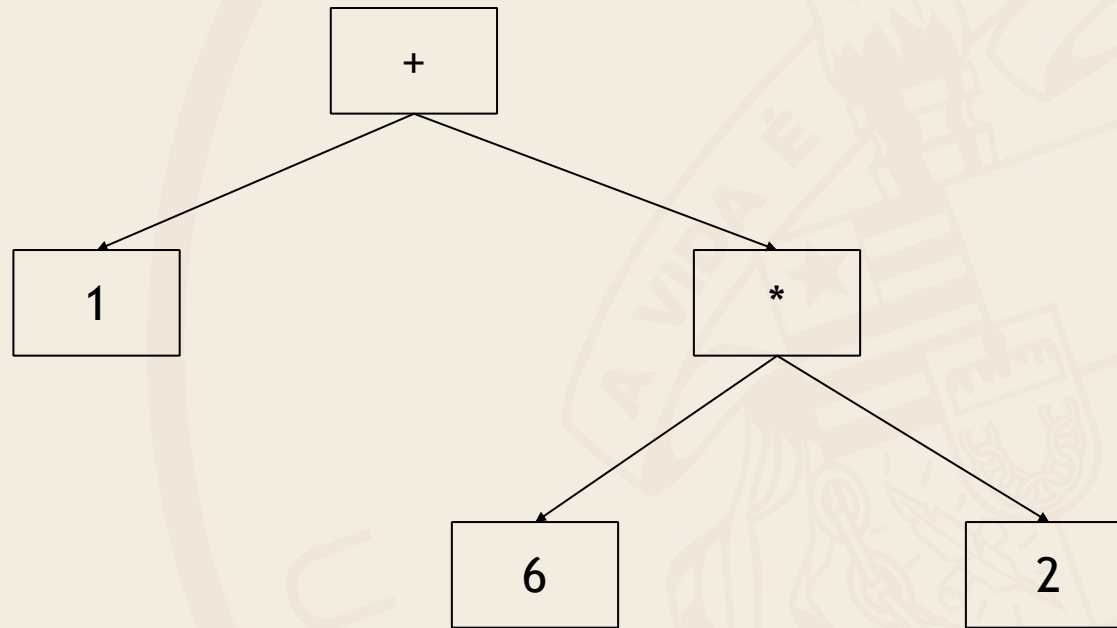
1+6*2

Árvores binárias: em-ordem



1. Percorrer a subárvore à esquerda
2. Visitar o nó raiz
3. Percorrer a subárvore à direita

Árvores binárias: em-ordem

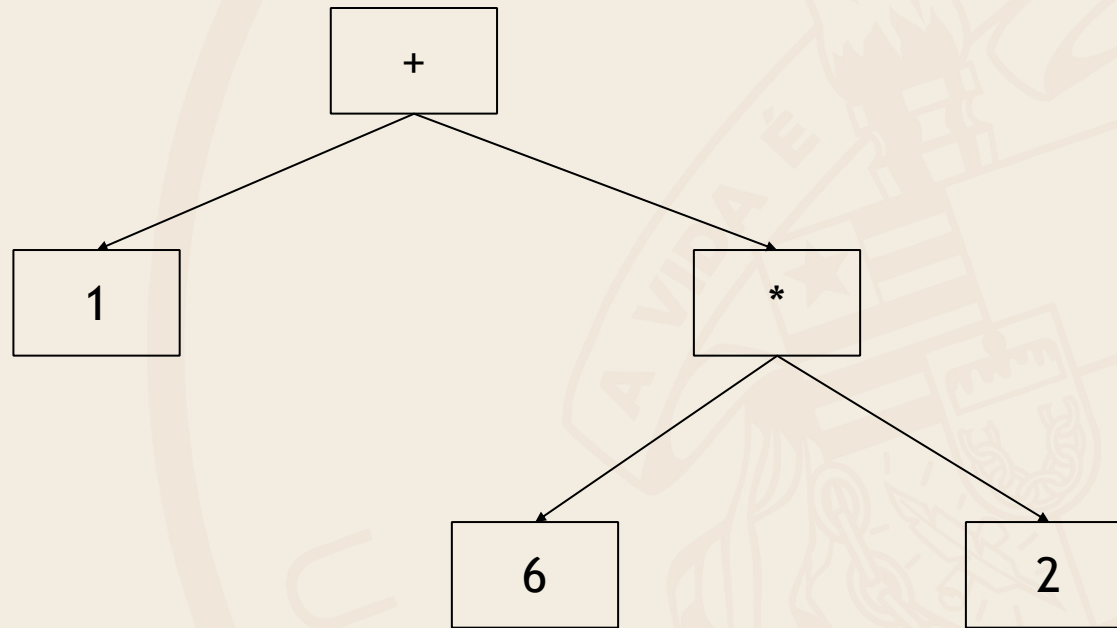


1. Percorrer a subárvore à esquerda
2. Visitar o nó raiz
3. Percorrer a subárvore à direita

```
def inOrdem (arvore):  
    if arvore != None:  
        inOrdem (arvore.se)  
        print (arvore)  
        inOrdem (arvore.sd)
```

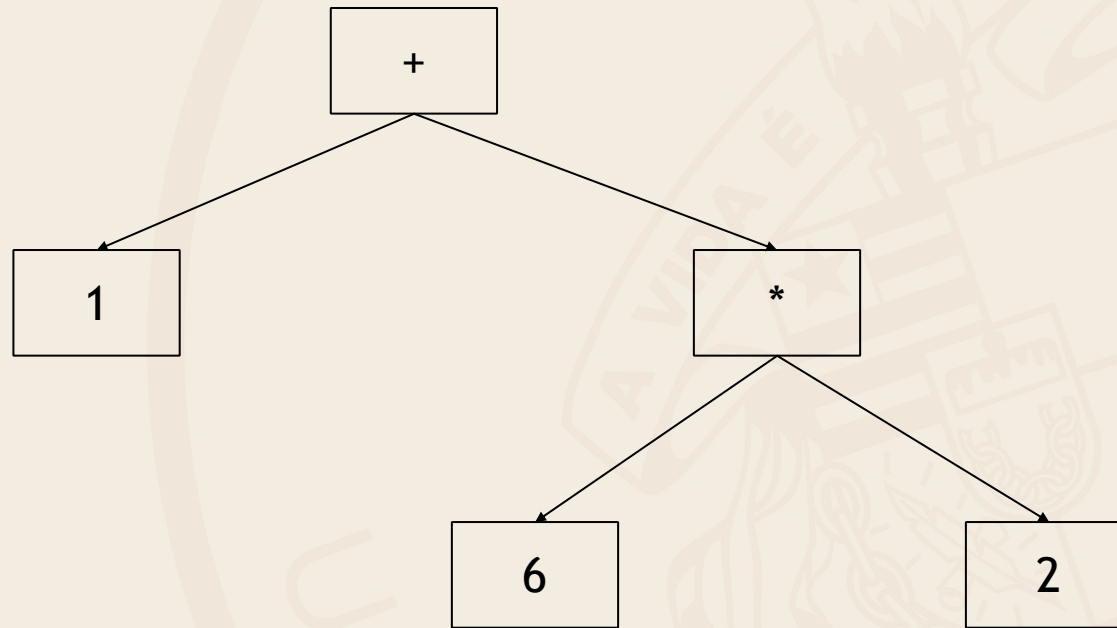
1 + 6 * 2

Árvores binárias: pós-ordem



1. Percorrer a subárvore à esquerda
2. Percorrer a subárvore à direita
3. Visitar o nó raiz

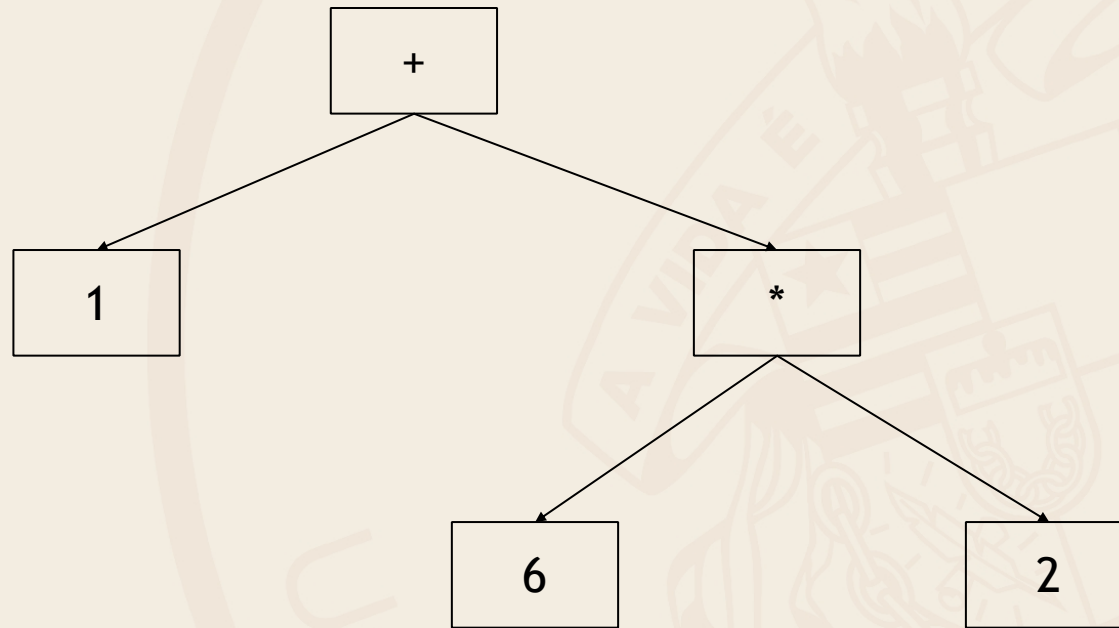
Árvores binárias: pós-ordem



1. Percorrer a subárvore à esquerda
2. Percorrer a subárvore à direita
3. Visitar o nó raiz

162*+

Árvores binárias: pós-ordem



1. Percorrer a subárvore à esquerda
2. Percorrer a subárvore à direita
3. Visitar o nó raiz

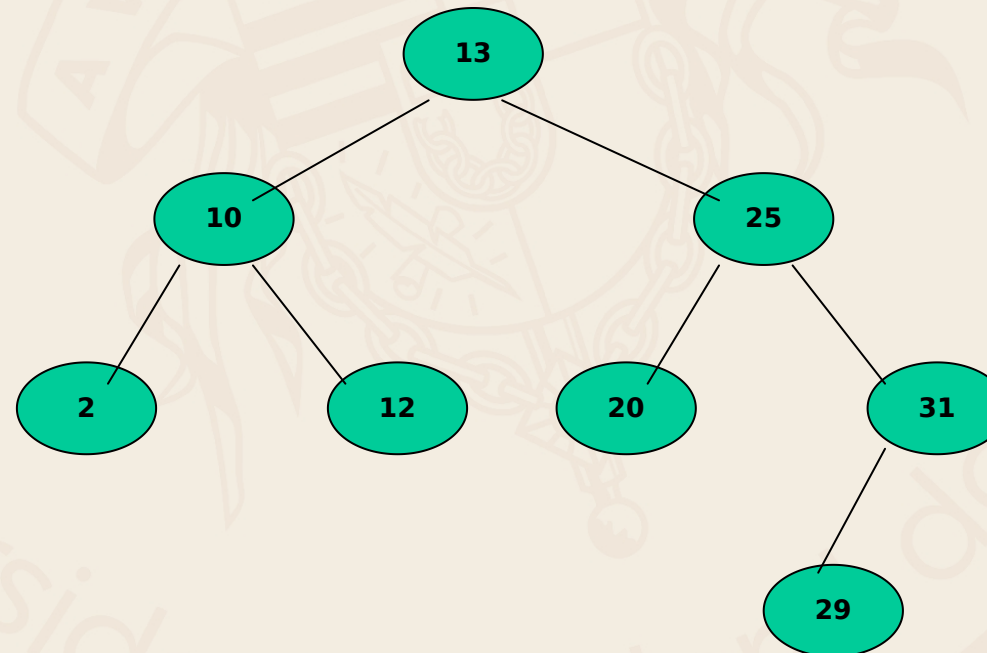
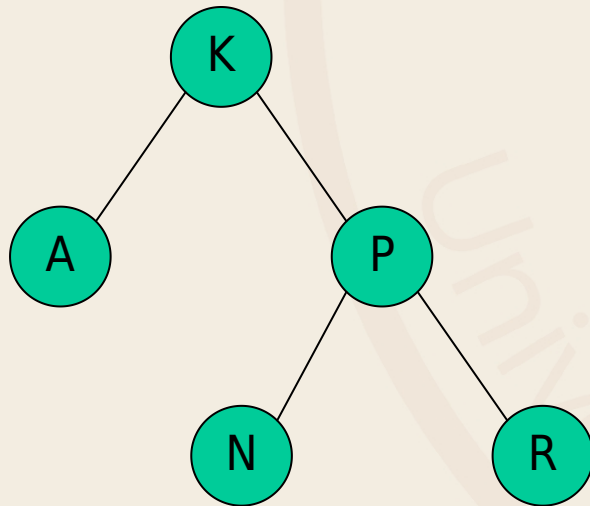
```
def posOrdem (arvore):  
    if arvore != None:  
        posOrdem (arvore.se)  
        posOrdem (arvore.sd)  
        print (arvore)
```

162*+

Árvore Binária de Busca

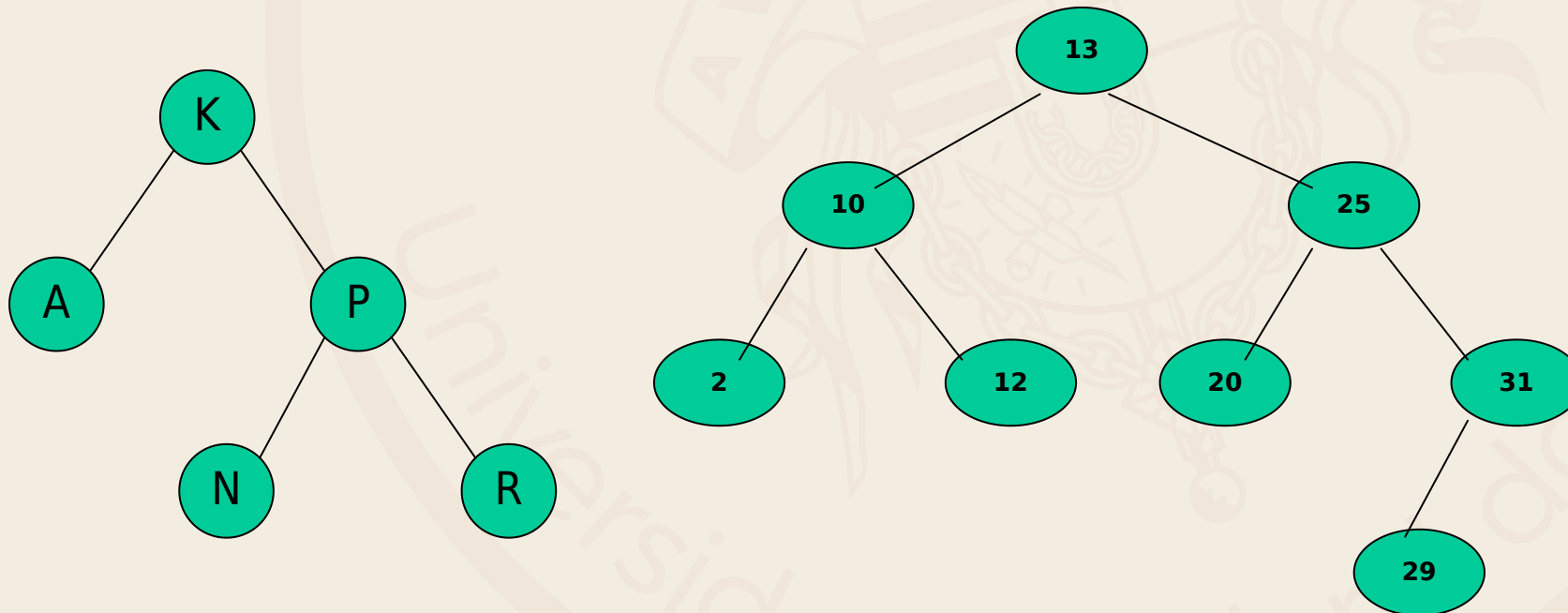
Uma importante aplicação das árvores binárias é para a realização de buscas.

O que as árvores abaixo tem em comum?



Árvore Binária de Busca

Para cada nó da árvore, todos os valores armazenados em sua sub-árvore à esquerda são menores que o valor armazenado no próprio nó, e todos os valores armazenados na sub-árvore à direita são maiores que o nó.



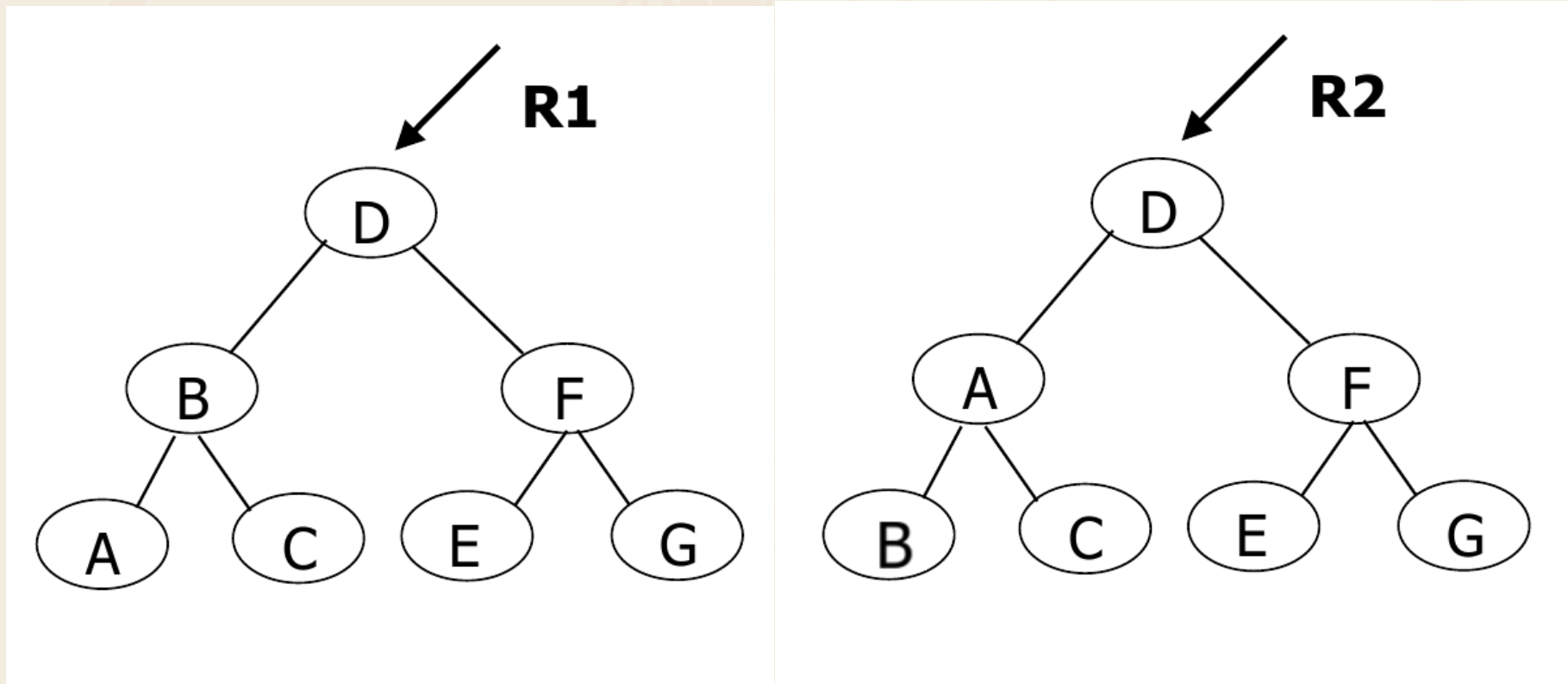
Árvore Binária de Busca

Então, em uma ABB, tem-se que

- É uma árvore binária
- Para qualquer nó v , os nós de sua sub-árvore à esquerda possuem valores menores do que o valor associado a v
- Para qualquer nó v , os nós de sua sub-árvore à direita possuem valores maiores do que o valor associado a v
- As sub-árvores à esquerda e à direita são ABB

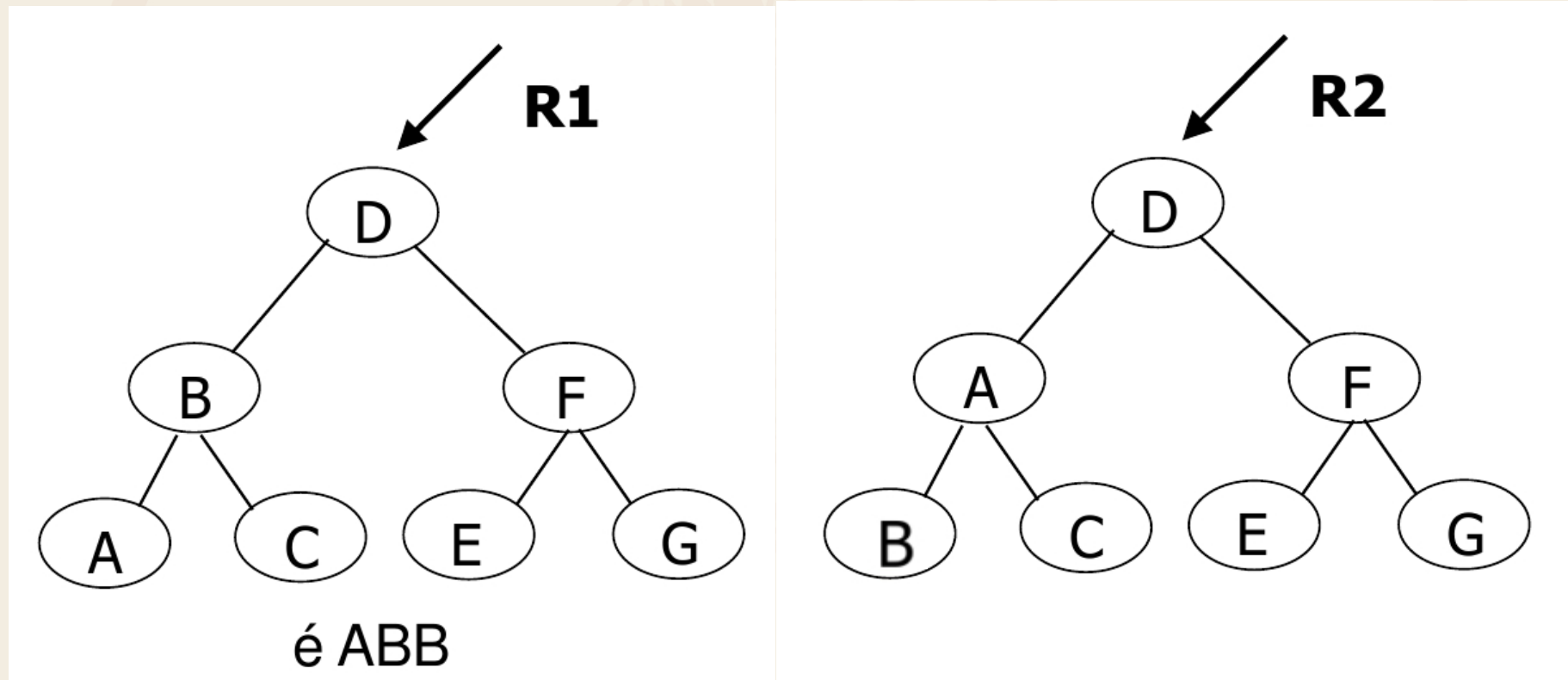
Árvore Binária de Busca

Exemplos



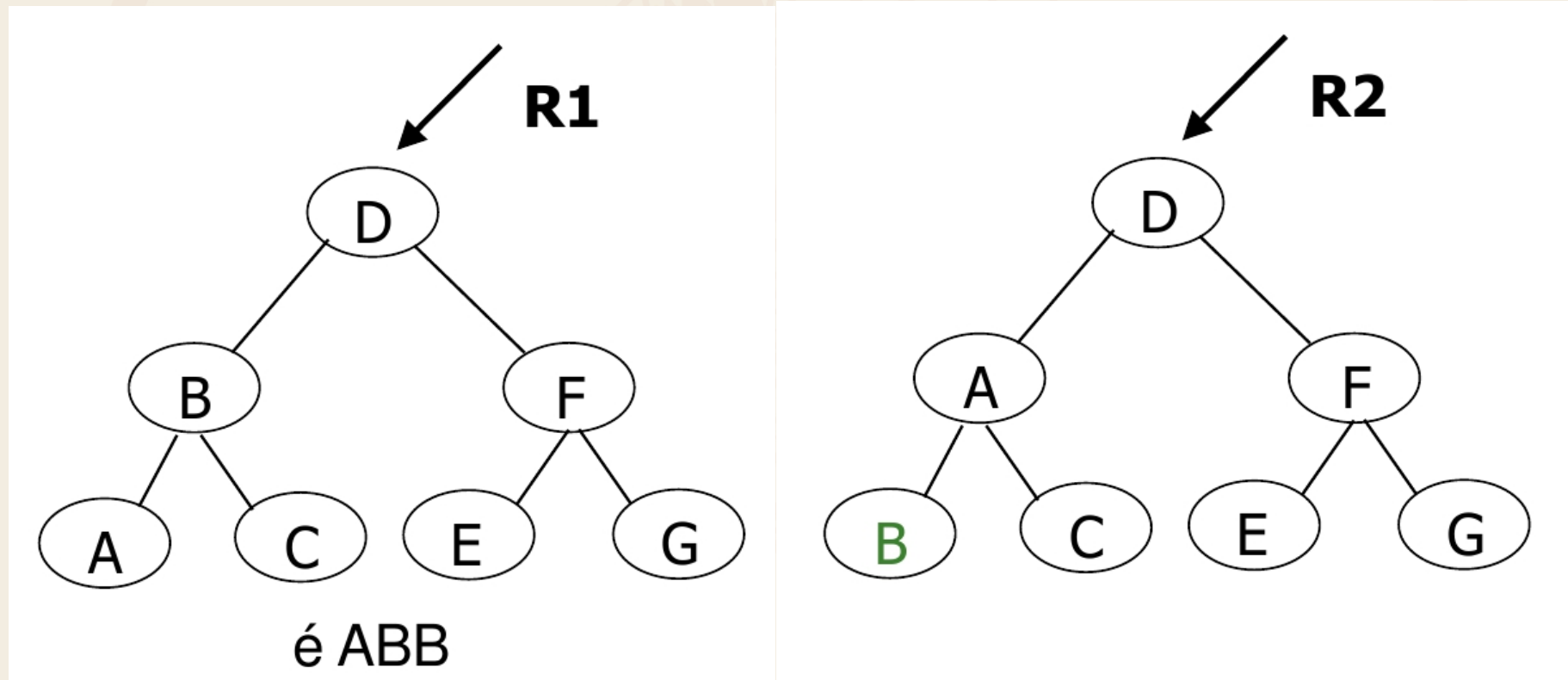
Árvore Binária de Busca

Exemplos



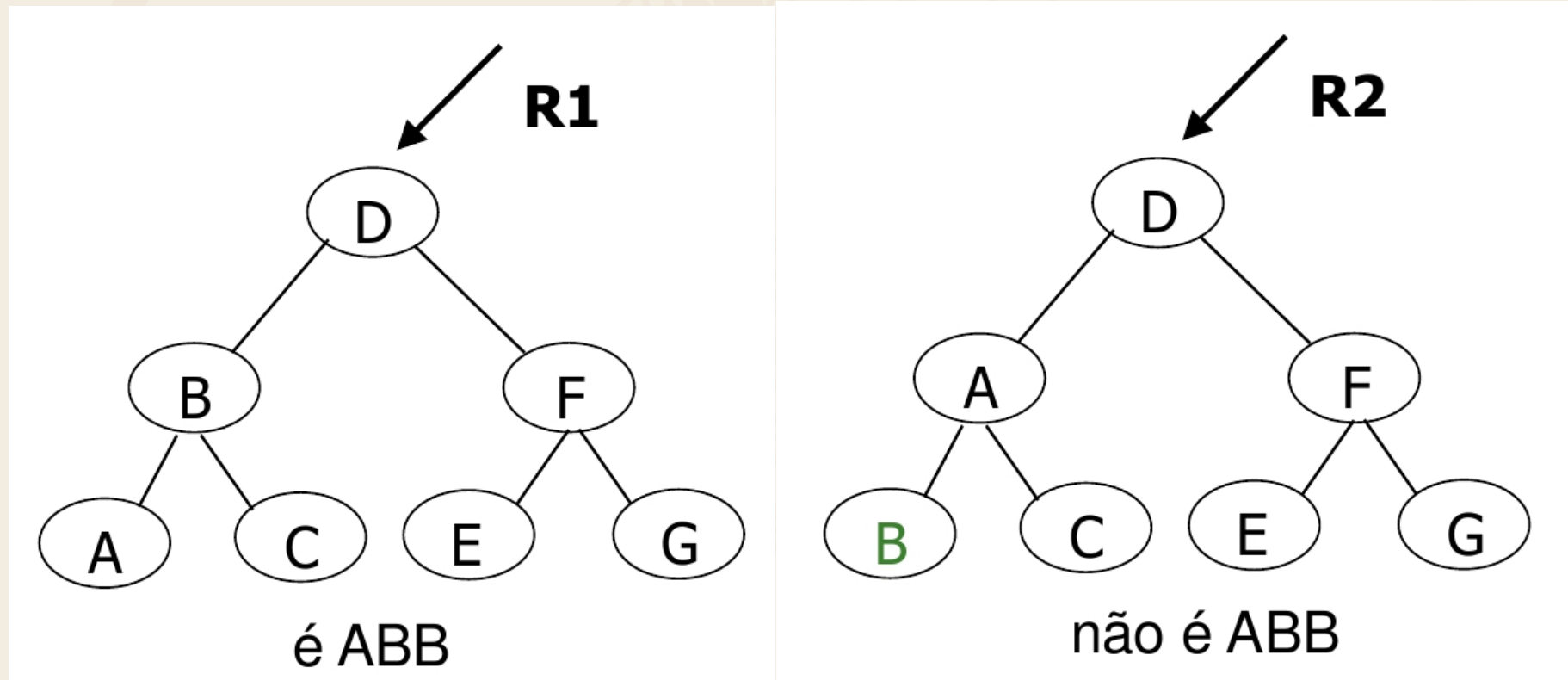
Árvore Binária de Busca

Exemplos



Árvore Binária de Busca

Exemplos



Árvore Binária de Busca

Por que utilizar ABB ?

Imagine um sistema de votação por telefone

- Cada número só pode votar uma vez
- O sistema deve armazenar todos os números que já votaram
- A cada ligação deve-se verificar se o número já votou
- A votação deve ter resultado em tempo real

Árvore Binária de Busca

Por que utilizar ABB ?

Implementação do sistema por ABB

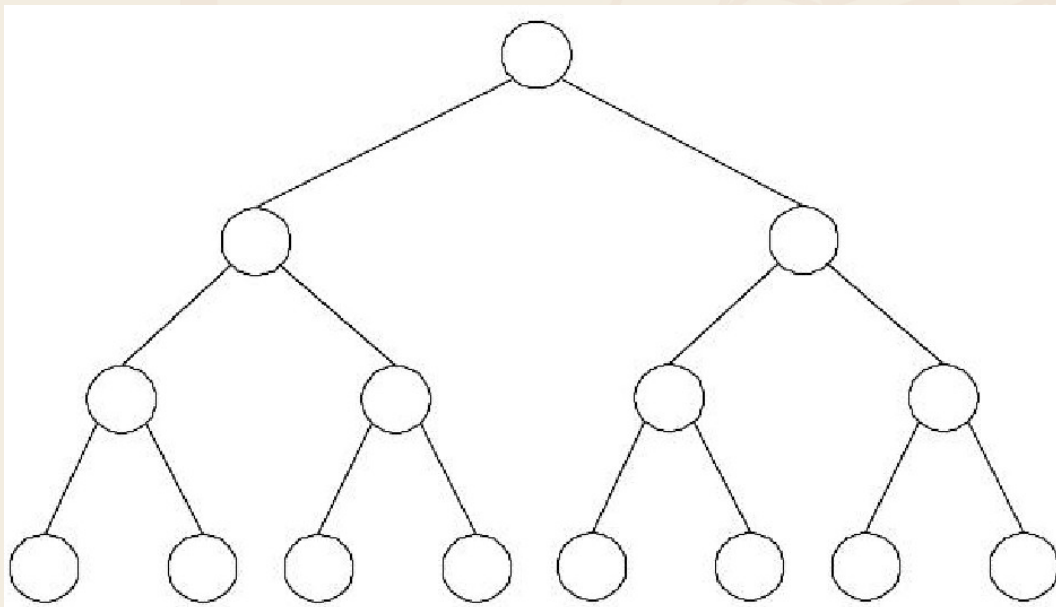
- Cada número é armazenado em um nó da ABB
- Suponha que em um determinado momento, a ABB tenha milhões de telefones armazenados
- Surge uma nova ligação e é preciso saber se o número está ou não na árvore (já votou ou não)

Árvore Binária de Busca

Por que utilizar ABB?

Implementação do sistema por ABB

– Considere, neste caso, que esta ABB é uma Árvore Binária Cheia



0	1
1	3
2	7
...	...
10	2.047
16	131.071
20	2 milhões

Árvore Binária de Busca

Por que utilizar ABB ?

Implementação do sistema por ABB

– Considere, neste caso, que esta ABB é uma Árvore Binária Cheia

» Para buscar um elemento na árvore, percorre-se os nós da raiz até as folhas, sem passar por mais de um nó em um mesmo nível

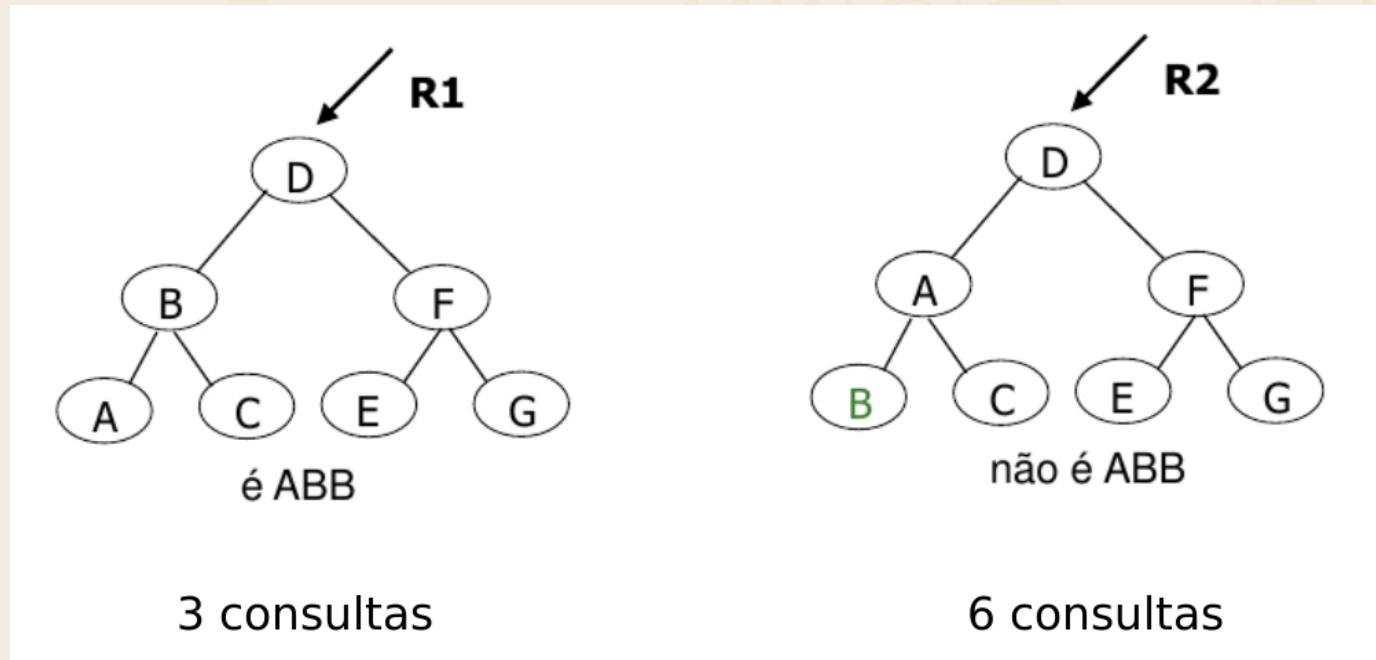
» Portanto, no pior caso, a busca passa por tantos nós quanto for a altura da árvore

Árvore Binária de Busca

Por que utilizar ABB ?

Implementação do sistema por ABB

– Exemplo: buscar pelo elemento E nas árvores abaixo



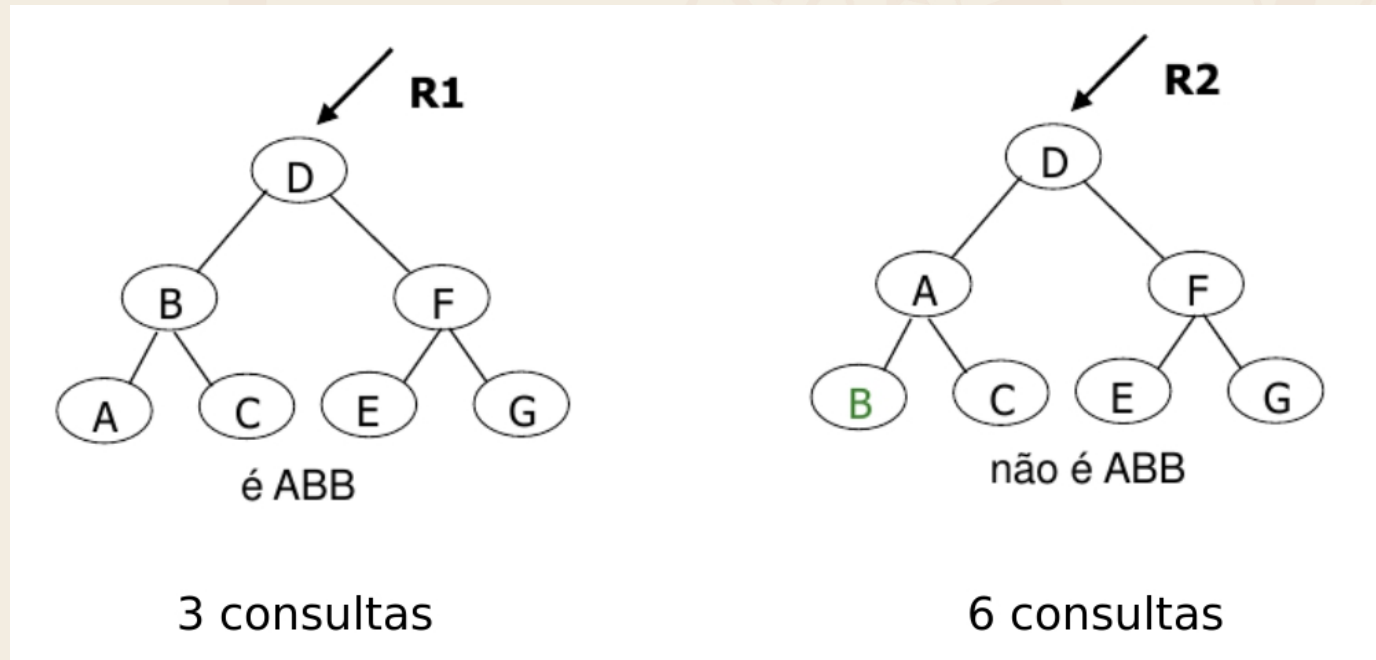
Árvore Binária de Busca

Buscas muito rápidas!

Por que utilizar ABB ?

Implementação do sistema por ABB

– Exemplo: buscar pelo elemento E nas árvores abaixo



Árvore Binária de Busca

Operações básicas em uma ABB

1. Buscar um elemento
2. Inserir um elemento
3. Remover um elemento

OBS: as operações devem considerar a ordenação dos elementos na ABB. Por exemplo, na inserção, deve-se procurar pelo lugar certo para Inserir o elemento.

Árvore Binária de Busca: Busca

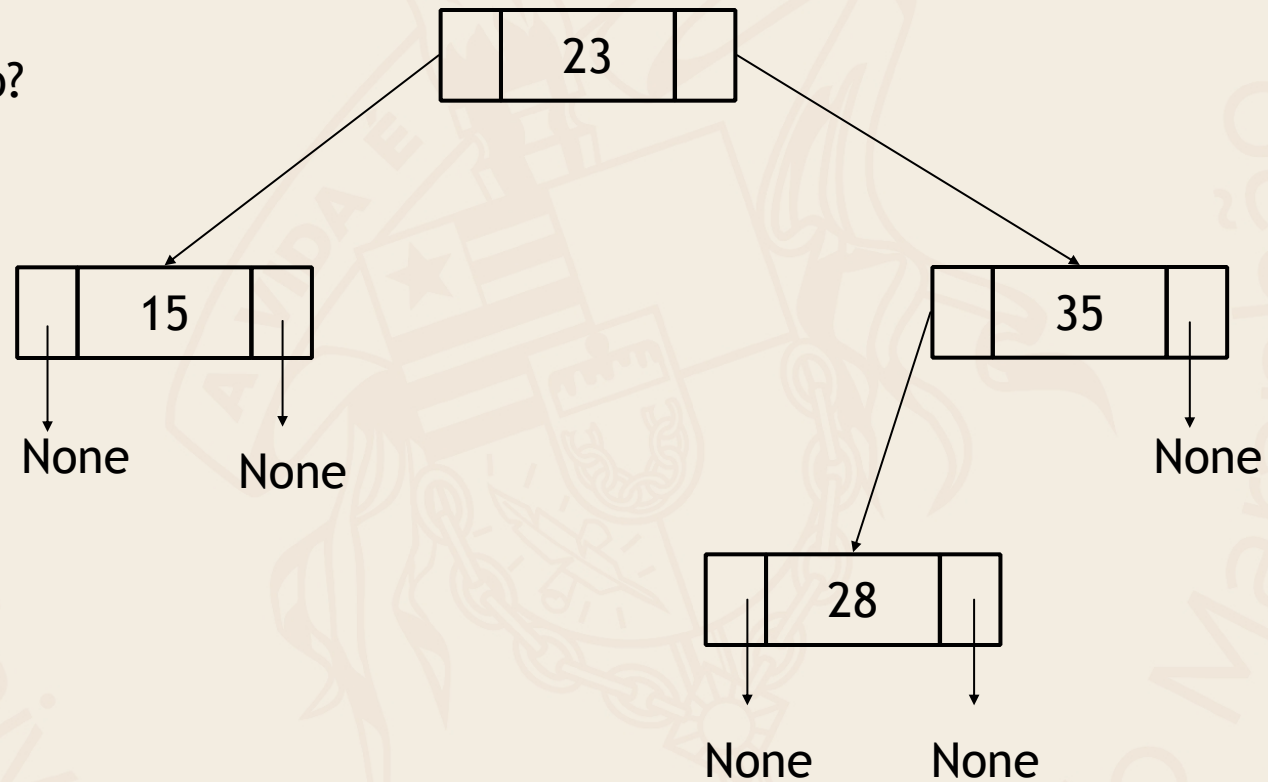
Comparando a “chave” com a informação no nó raiz, quatro casos pode ocorrer:

1. A árvore é vazia → a chave não está na árvore → Fim
2. A chave buscada está na raiz → elemento encontrado → Fim
3. A chave é menor do que a informação da raiz → fazer busca na sub-árvore da esquerda
4. A chave é maior do que a informação da raiz → fazer busca na sub-árvore da direita

```
def busca (r, x):  
    if r == None:  
        return None  
    elif r.info == x:  
        return r  
    elif x > r.info:  
        return busca(r.sd,x)  
    else:  
        return busca(r.se,x)
```

Buscar

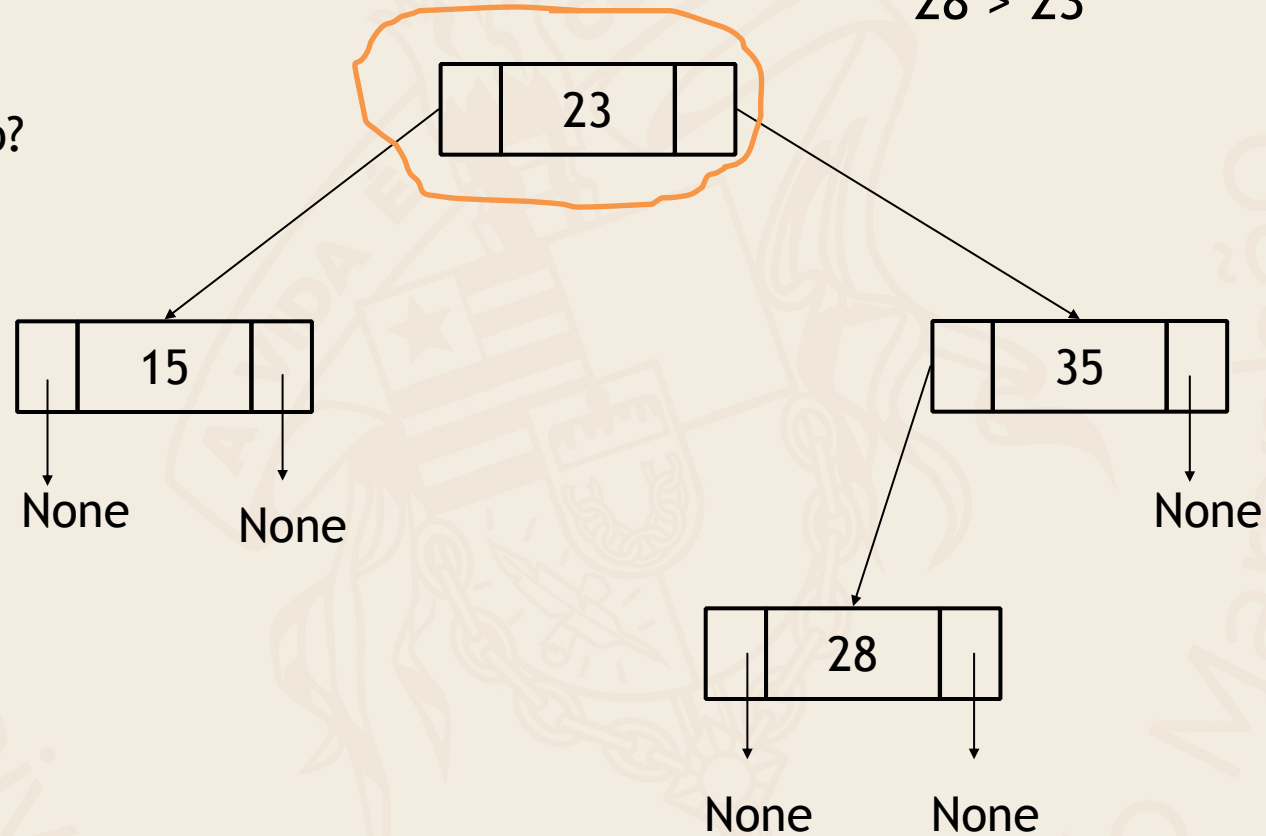
Há um 28
armazenado?



Buscar

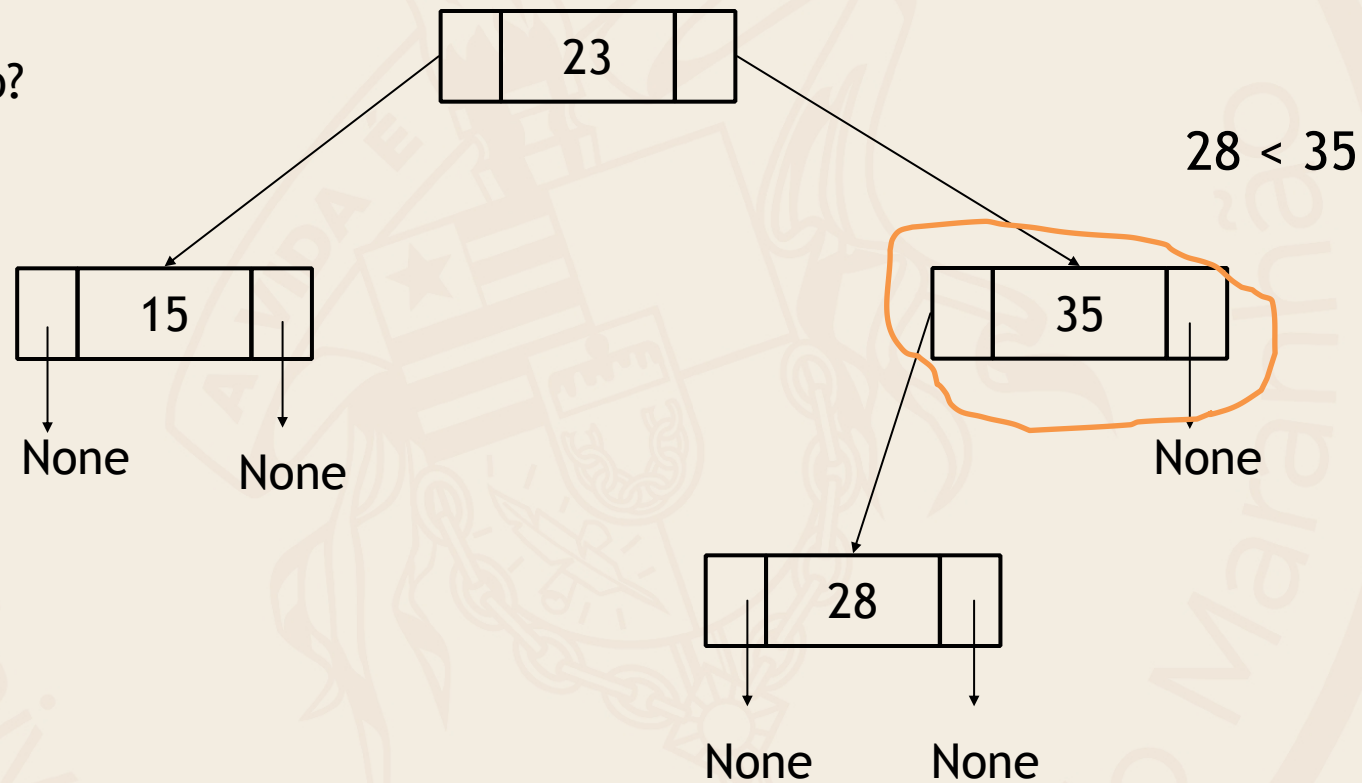
Há um 28
armazenado?

$28 > 23$



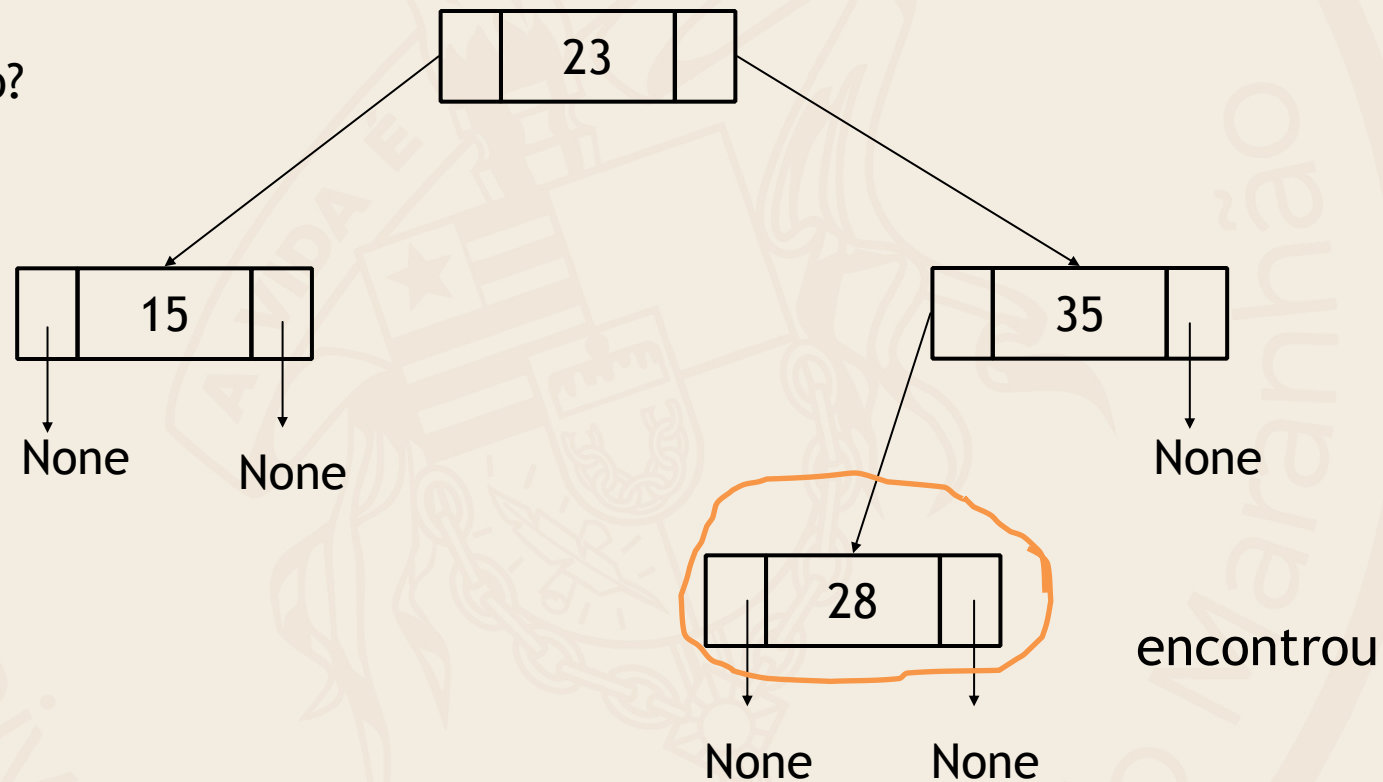
Buscar

Há um 28
armazenado?



Buscar

Há um 28
armazenado?



Árvore Binária de Busca: inserção

Estratégia geral: sempre inserir elementos como nós folhas, procurando a posição correta.

Comparando a “chave” com a informação no nó raiz, quatro casos pode ocorrer:

1. A árvore é vazia → insere o elemento, que passará a ser a raiz → Fim
2. A chave buscada está na raiz → o elemento já está na árvore → Fim
3. A chave é menor do que a informação da raiz → insere na sub- árvore da esquerda
4. A chave é maior do que a informação da raiz → insere na sub- árvore da direita

```
def insere (r, x):  
    if r == None:  
        return Arvore (x)  
    elif x == r.info:  
        return r  
    elif x < r.info:  
        r.sd = insere (r.sd, x)  
    else:  
        r.se = insere (r.se, x)  
    return r
```

Inserir

Inserir
23

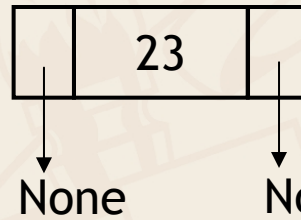
Inserir

Inserir
23

	23	
--	----	--

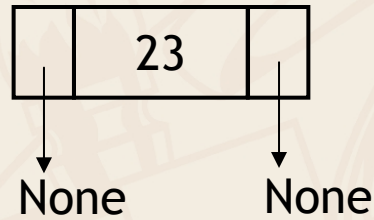
Inserir

Inserir
23



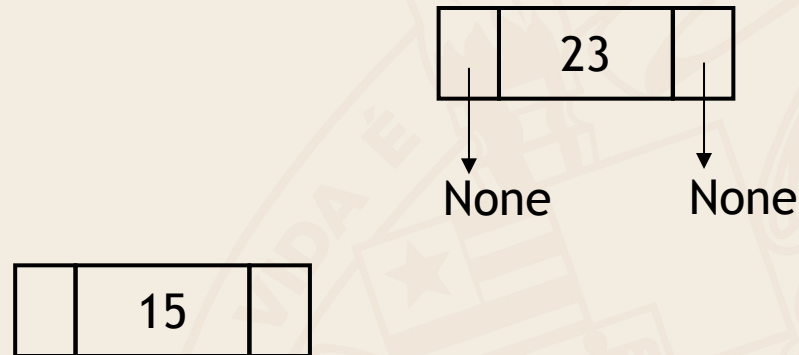
Inserir

Inserir
23
Inserir
15



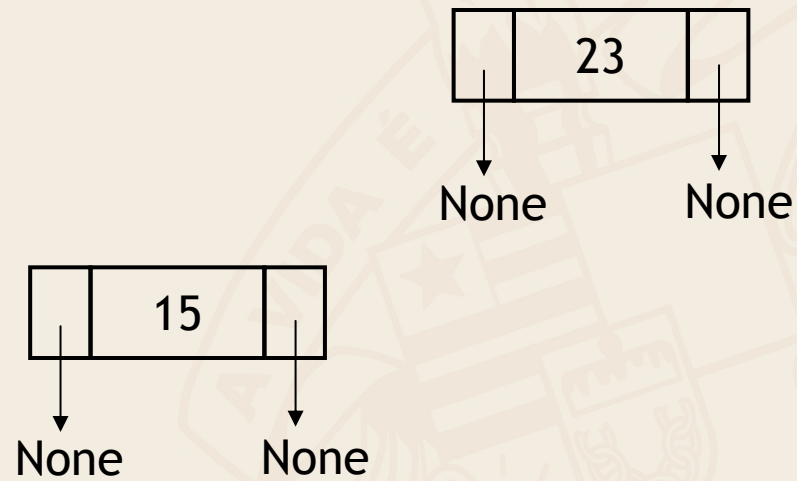
Inserir

Inserir
23
Inserir
15



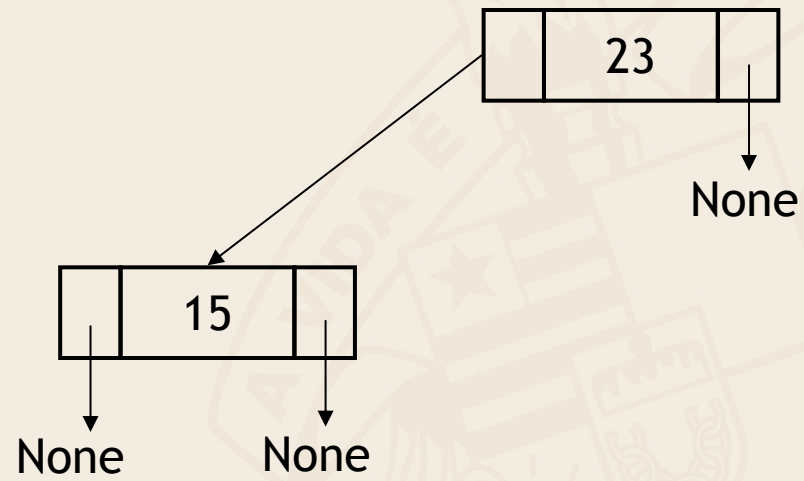
Inserir

Inserir
23
Inserir
15



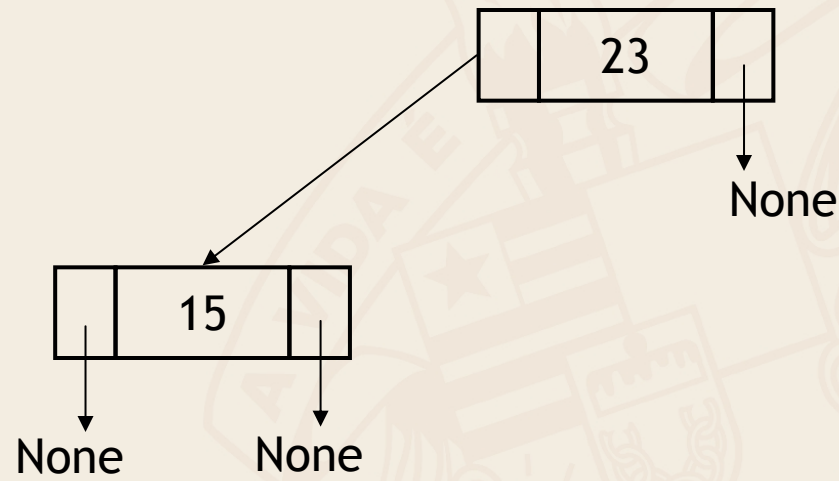
Inserir

Inserir
23
Inserir
15



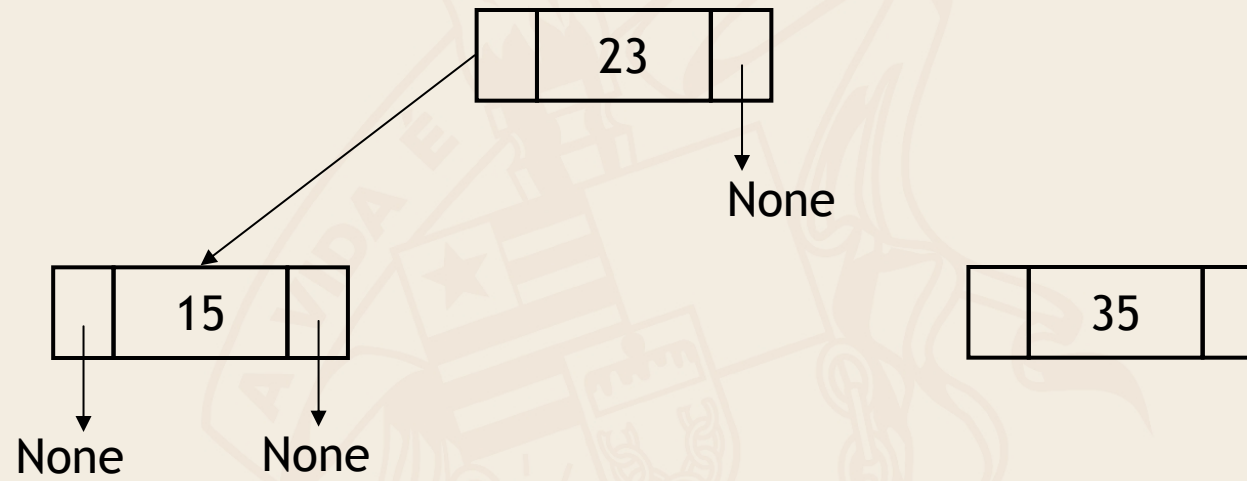
Inserir

Inserir
23
Inserir
15
Inserir
35



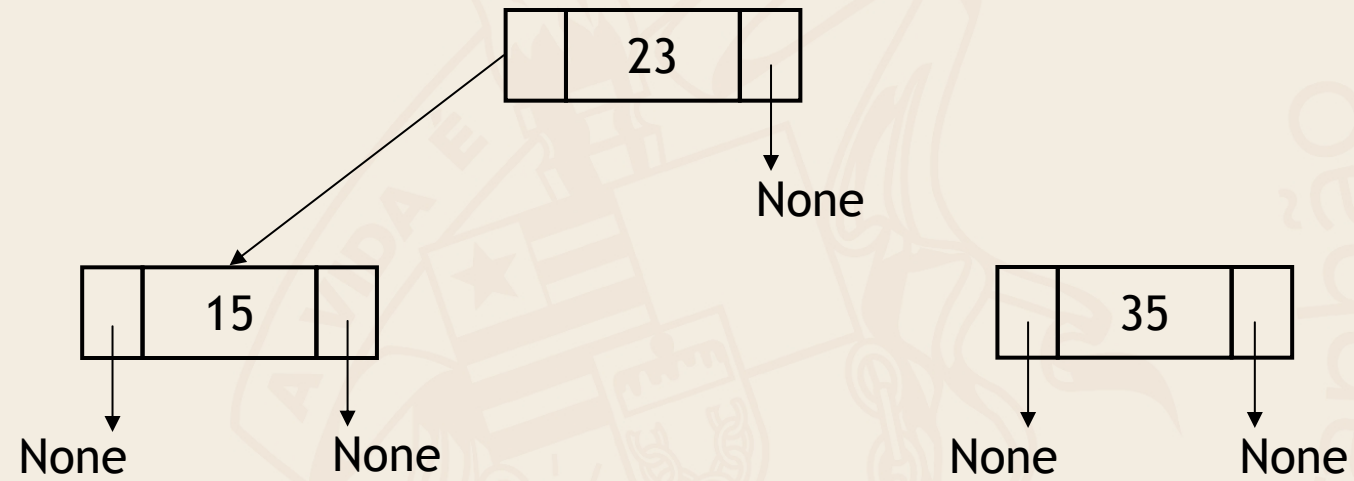
Inserir

Inserir
23
Inserir
15
Inserir
35



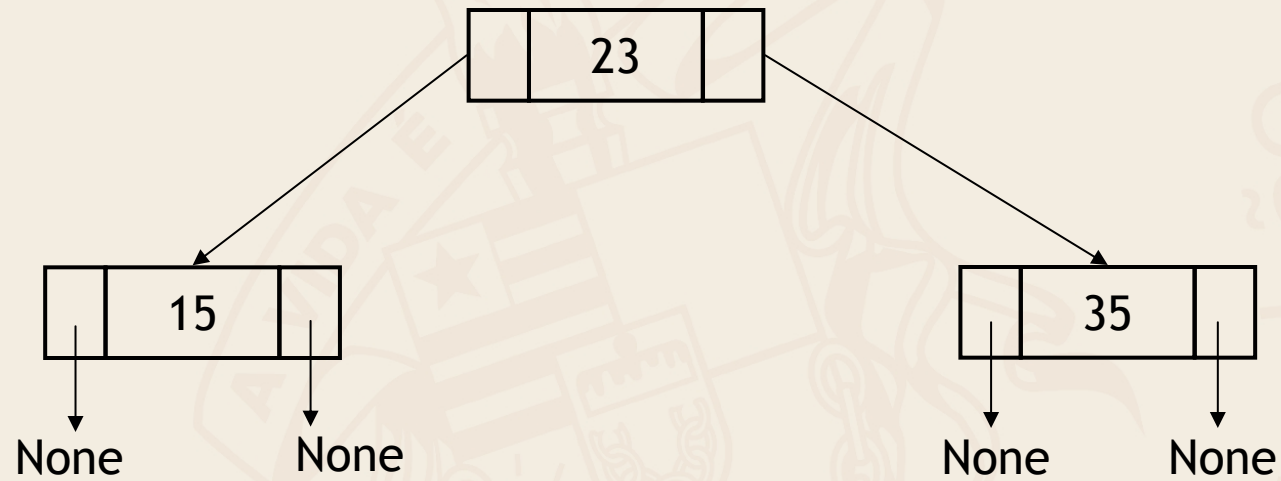
Inserir

Inserir
23
Inserir
15
Inserir
35



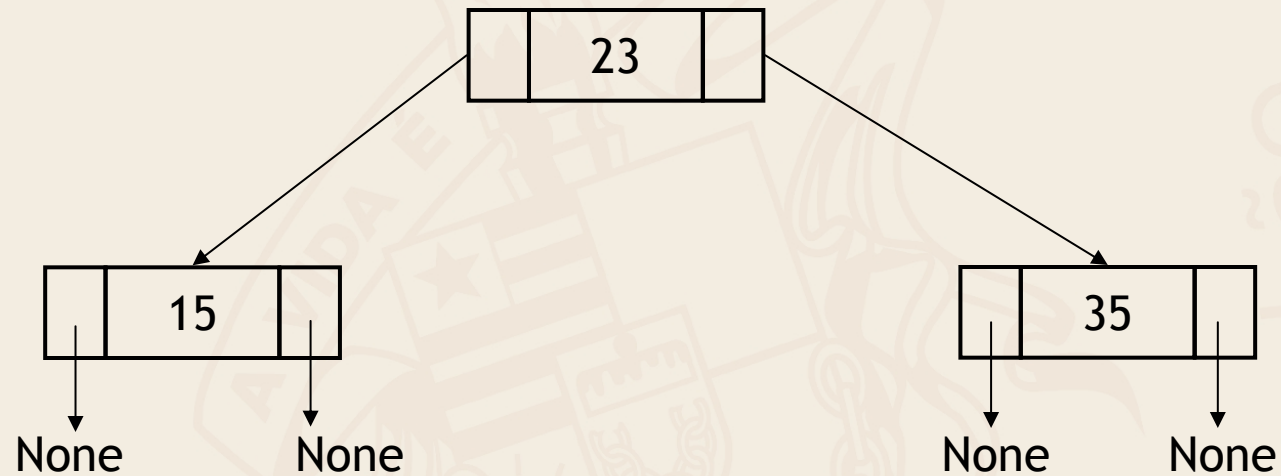
Inserir

Inserir
23
Inserir
15
Inserir
35



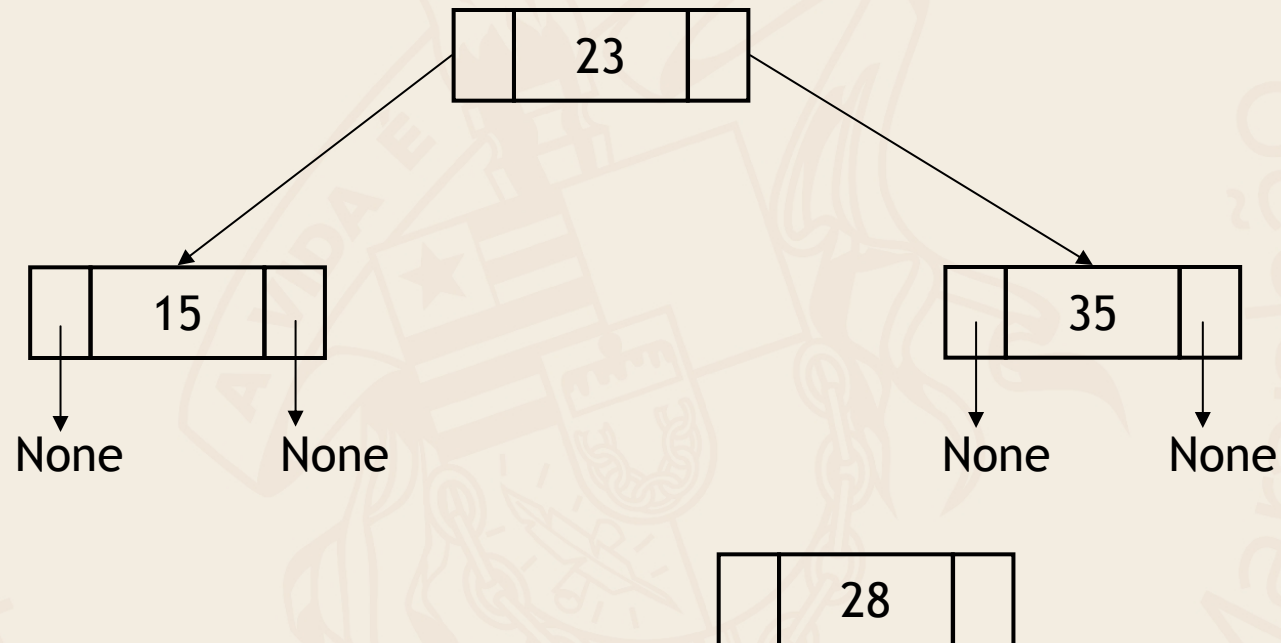
Inserir

Inserir
23
Inserir
15
Inserir
35
Inserir
28



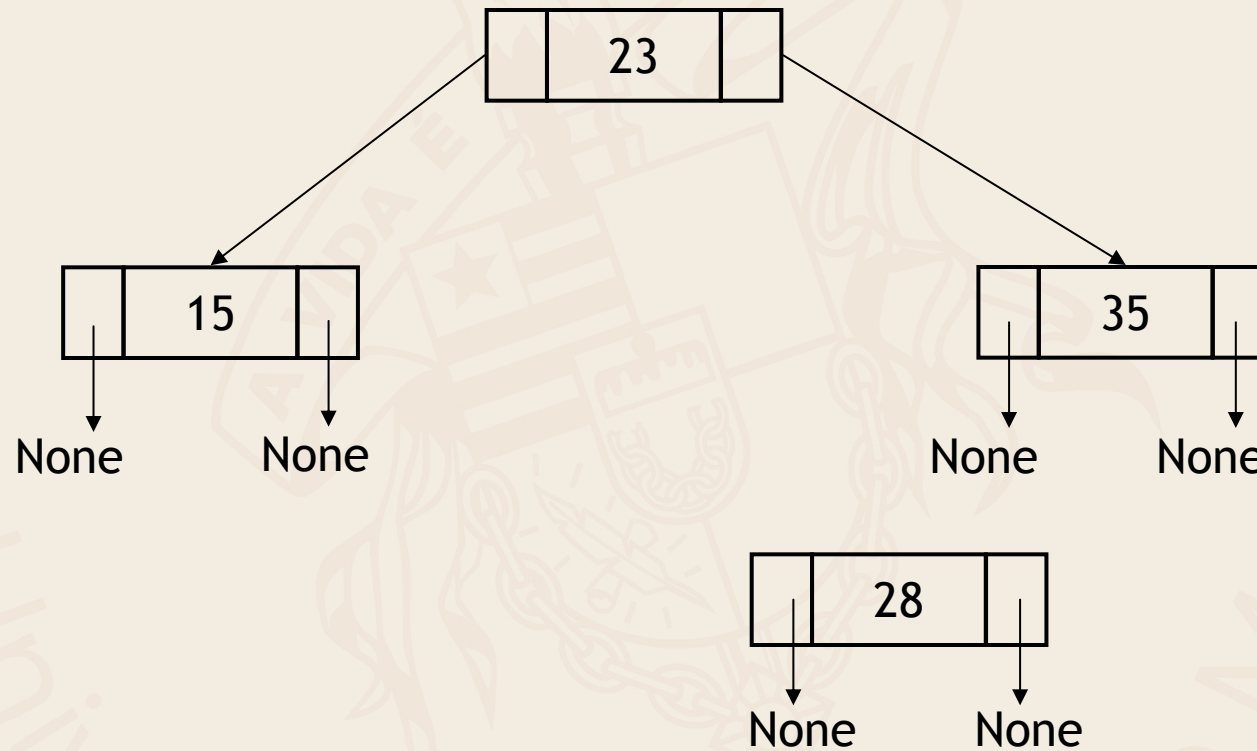
Inserir

Inserir
23
Inserir
15
Inserir
35
Inserir
28



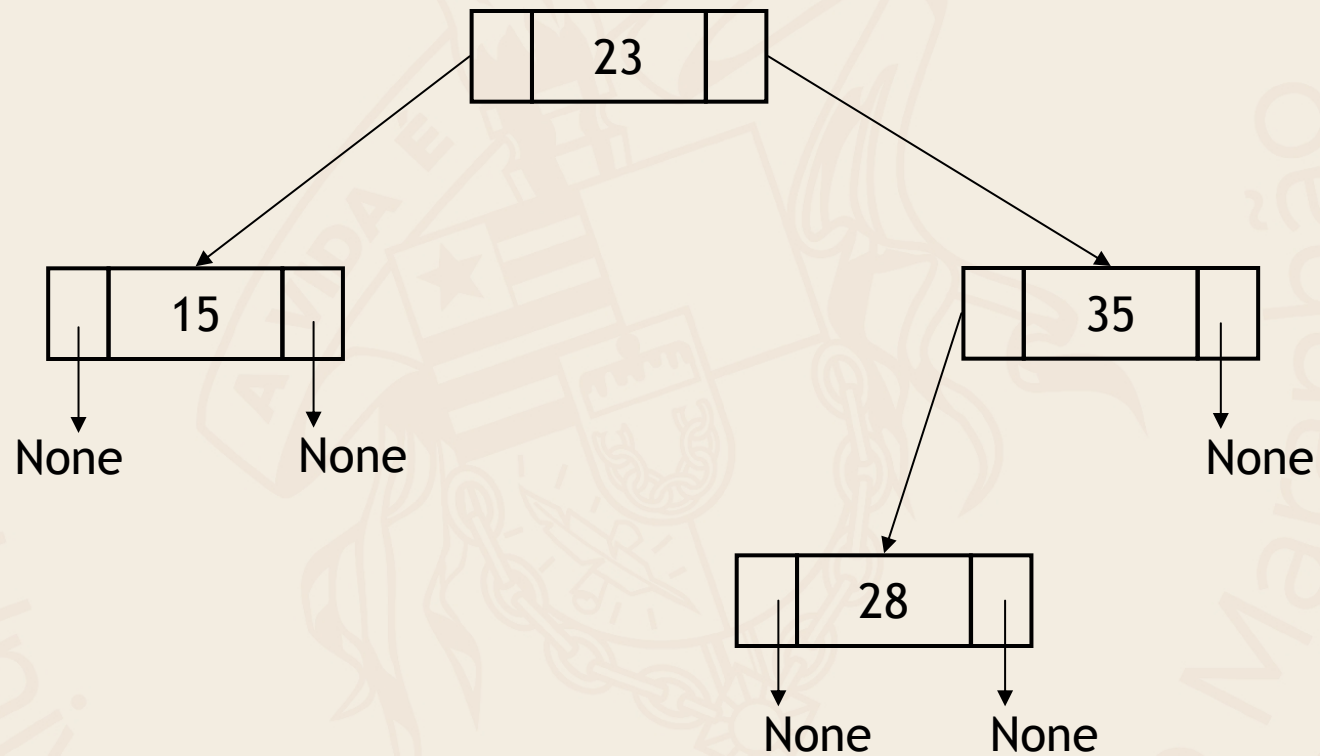
Inserir

Inserir
23
Inserir
15
Inserir
35
Inserir
28



Inserir

Inserir
23
Inserir
15
Inserir
35
Inserir
28



Árvore Binária de Busca: remoção

O desafio da remoção é manter a propriedade de árvore binária de busca após a remoção.

Caso 1: o nó a ser removido não tem filho

Caso 2: o nó a ser removido tem um único filho

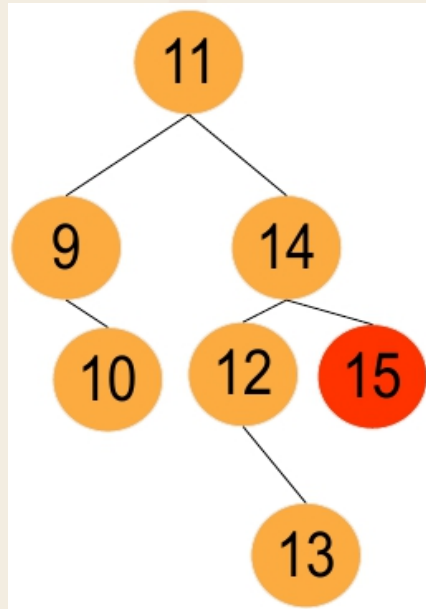
Caso 3: o nó a ser removido tem 2 filhos

Árvore Binária de Busca: remoção

Caso 1: o nó a ser removido não tem filho

Remove-se o nó

Quem apontava pra ele recebe NULL

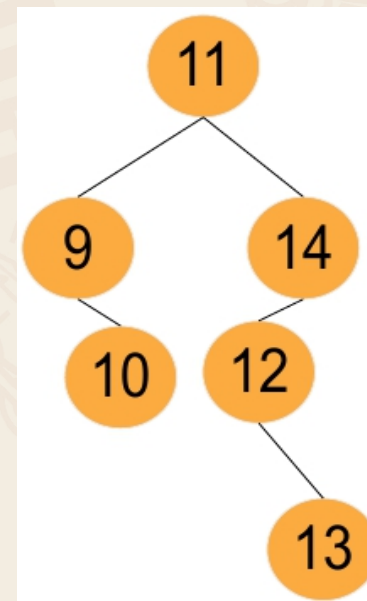
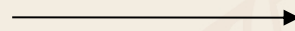
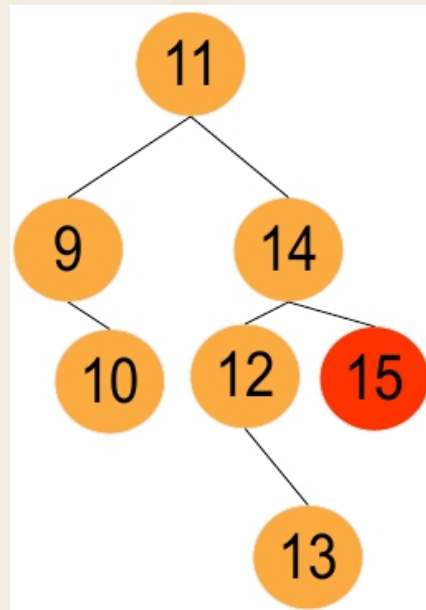


Árvore Binária de Busca: remoção

Caso 1: o nó a ser removido não tem filho

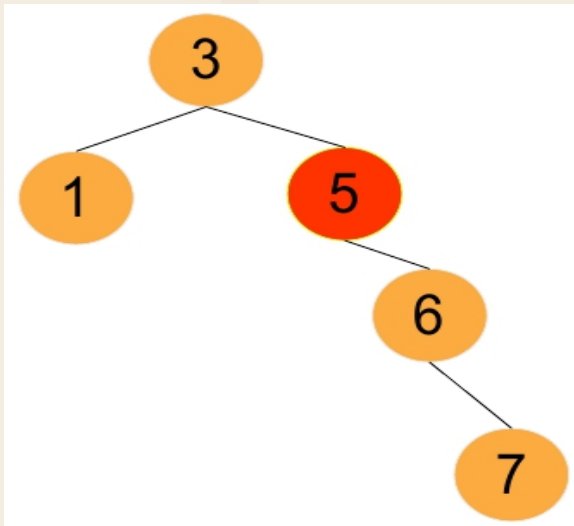
Remove-se o nó

Quem apontava pra ele recebe NULL



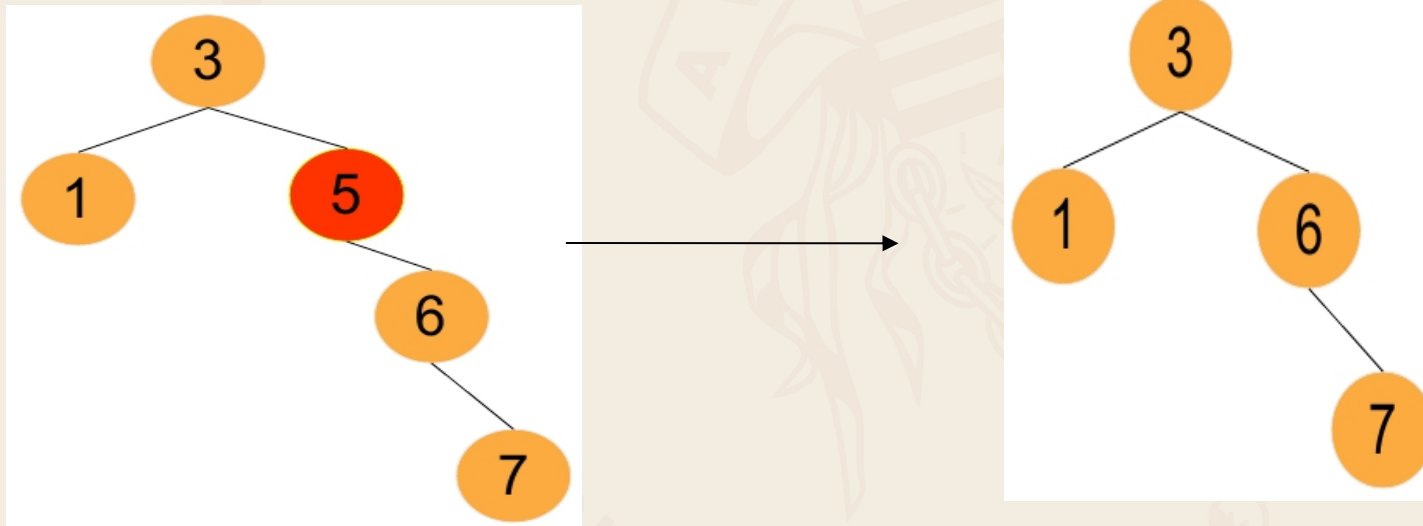
Árvore Binária de Busca: remoção

Caso 2 (remover C): o nó a ser removido tem um único filho
“Puxa-se” o filho para o lugar do pai
Remove-se o nó



Árvore Binária de Busca: remoção

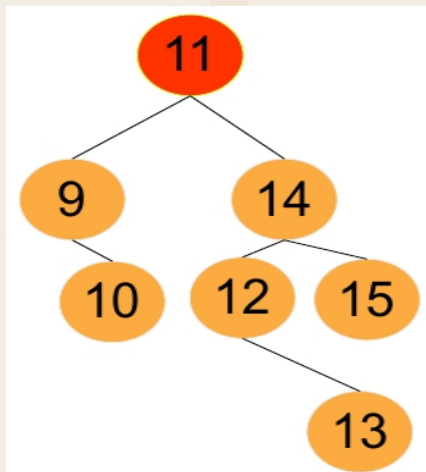
Caso 2 (remover C): o nó a ser removido tem um único filho
“Puxa-se” o filho para o lugar do pai
Remove-se o nó



Árvore Binária de Busca: remoção

Caso 3 (remover K): o nó a ser removido tem 2 filhos

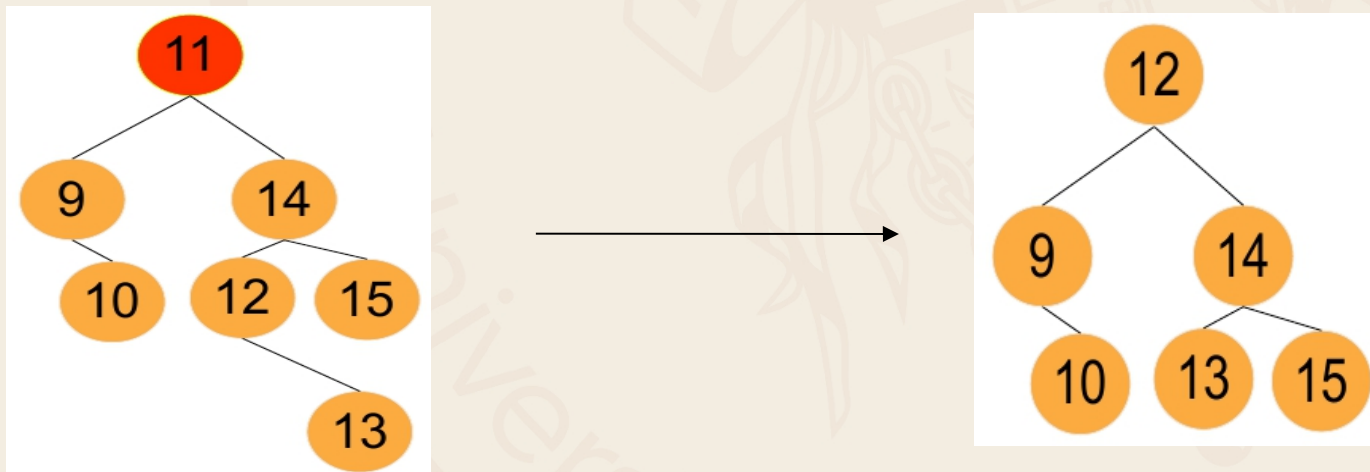
- Pode-se substituir o valor do nó a ser retirado pelo valor sucessor (o nó mais à esquerda da subárvore direita) ou
- Pelo valor antecessor (o nó mais à direita da subárvore esquerda), removendo-se aí o nó sucessor (ou antecessor).



Árvore Binária de Busca: remoção

Caso 3 (remover K): o nó a ser removido tem 2 filhos

- Pode-se substituir o valor do nó a ser retirado pelo valor sucessor (o nó mais à esquerda da subárvore direita) ou
- Pelo valor antecessor (o nó mais à direita da subárvore esquerda), removendo-se aí o nó sucessor (ou antecessor).



Árvore Binária de Busca: Exercícios

Faça passo a passo no Papel:

Insira o seguinte conjunto de dados:

[10,30,15,5,49,52,35]

Da árvore resultante, remova o seguinte conjunto de dados:

[10,5,52]

Na árvore resultante, insira:

[100, 80, 70]

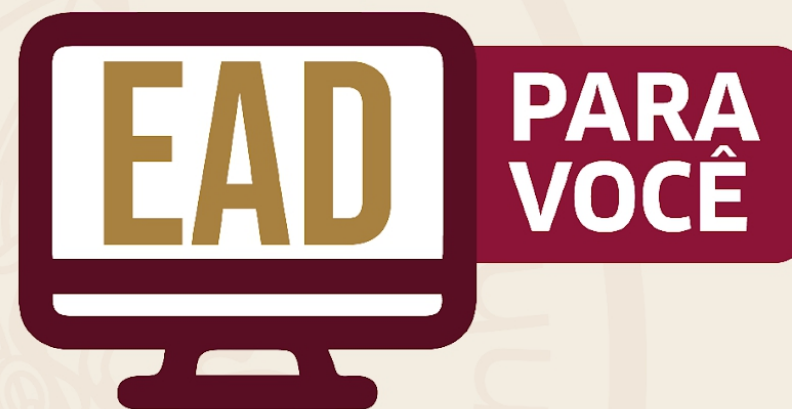
Remova:

[15,49,35]



dted

DIRETORIA DE TECNOLOGIAS
NA EDUCAÇÃO



www.eadparavc.dinte.ufma.br

 @ufma.eadparavoce