Algoritmos e Estrutura de Dados II

Aulas 13 e 14

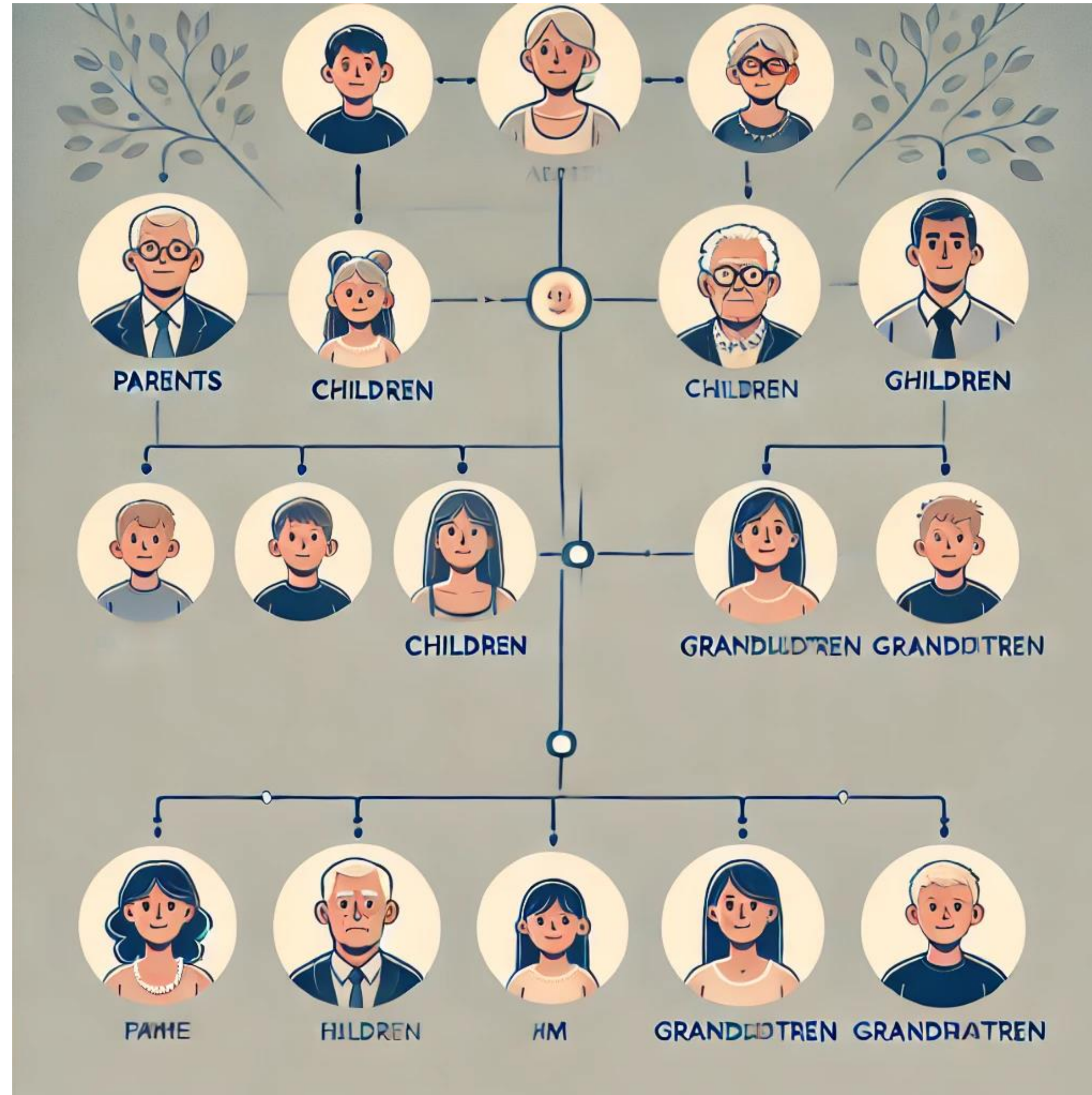
Árvores

Prof. Dr. Dilermando Piva Jr

2º Semestre - CDN

O que essas figuras tem em comum?

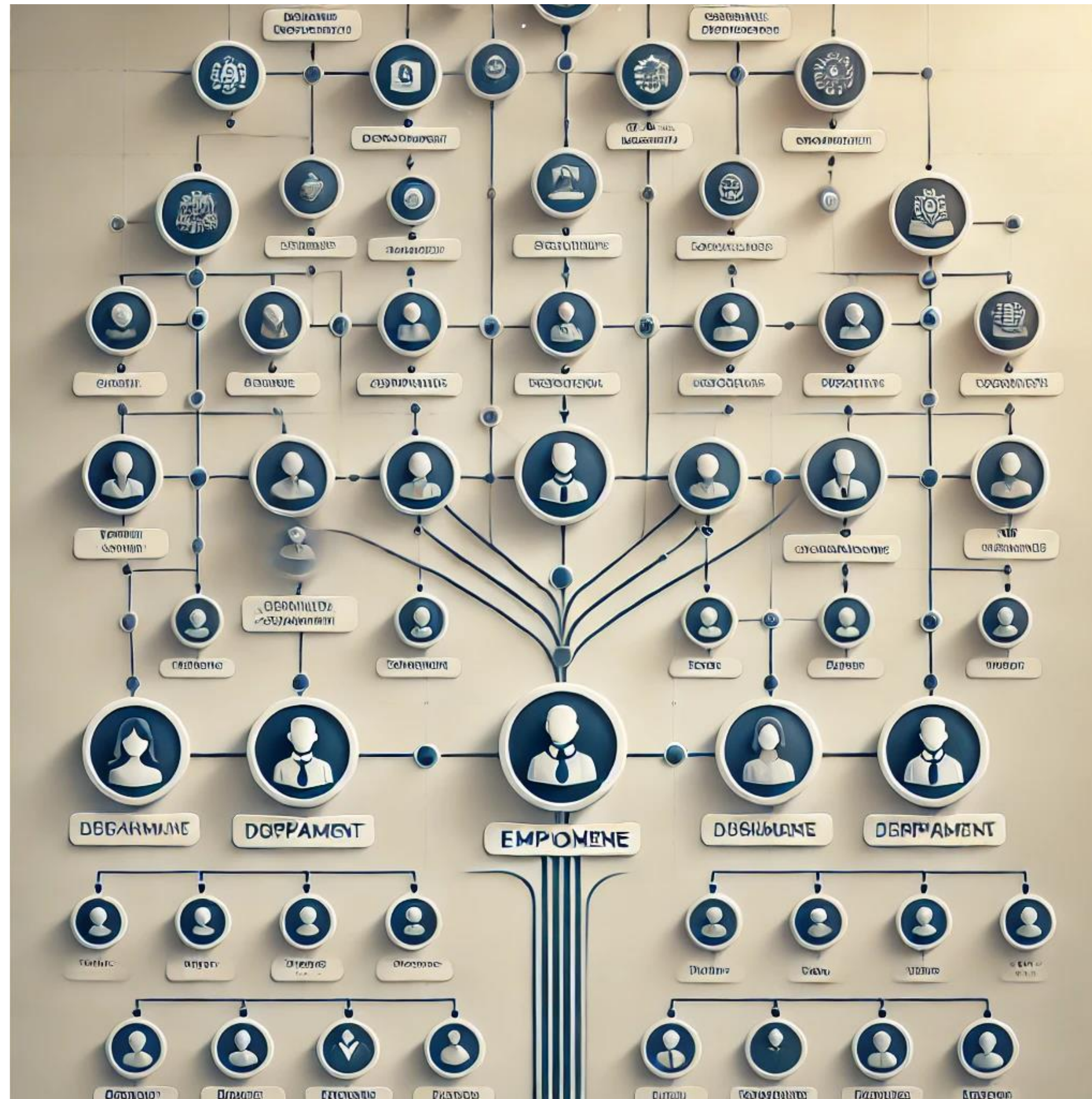
Árvore Genealógica



© Prof. Dr. Dilermando Piva Jr.

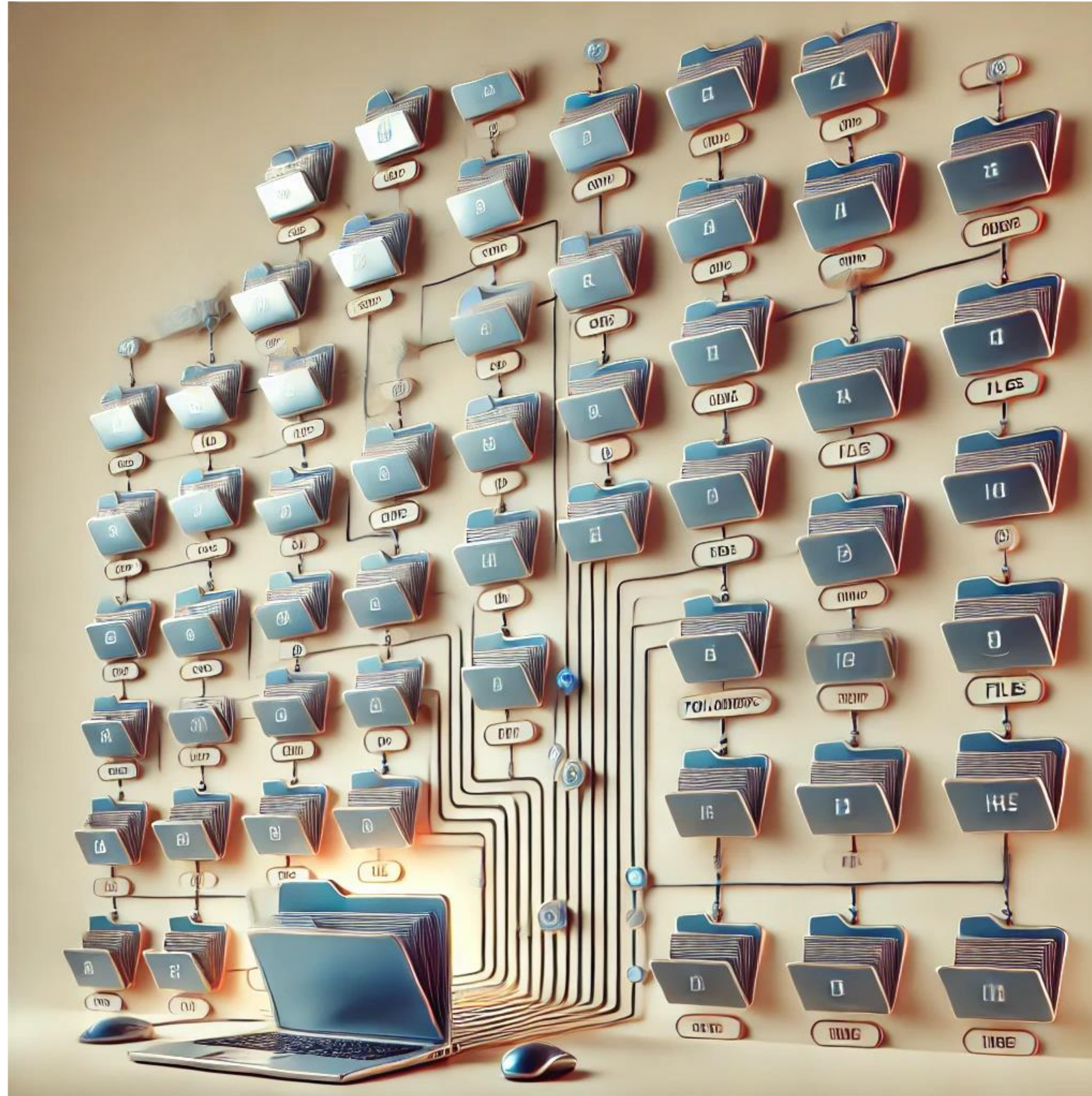
O que essas figuras tem em comum?

Árvore (estrutura) Organizacional



O que essas figuras tem em comum?

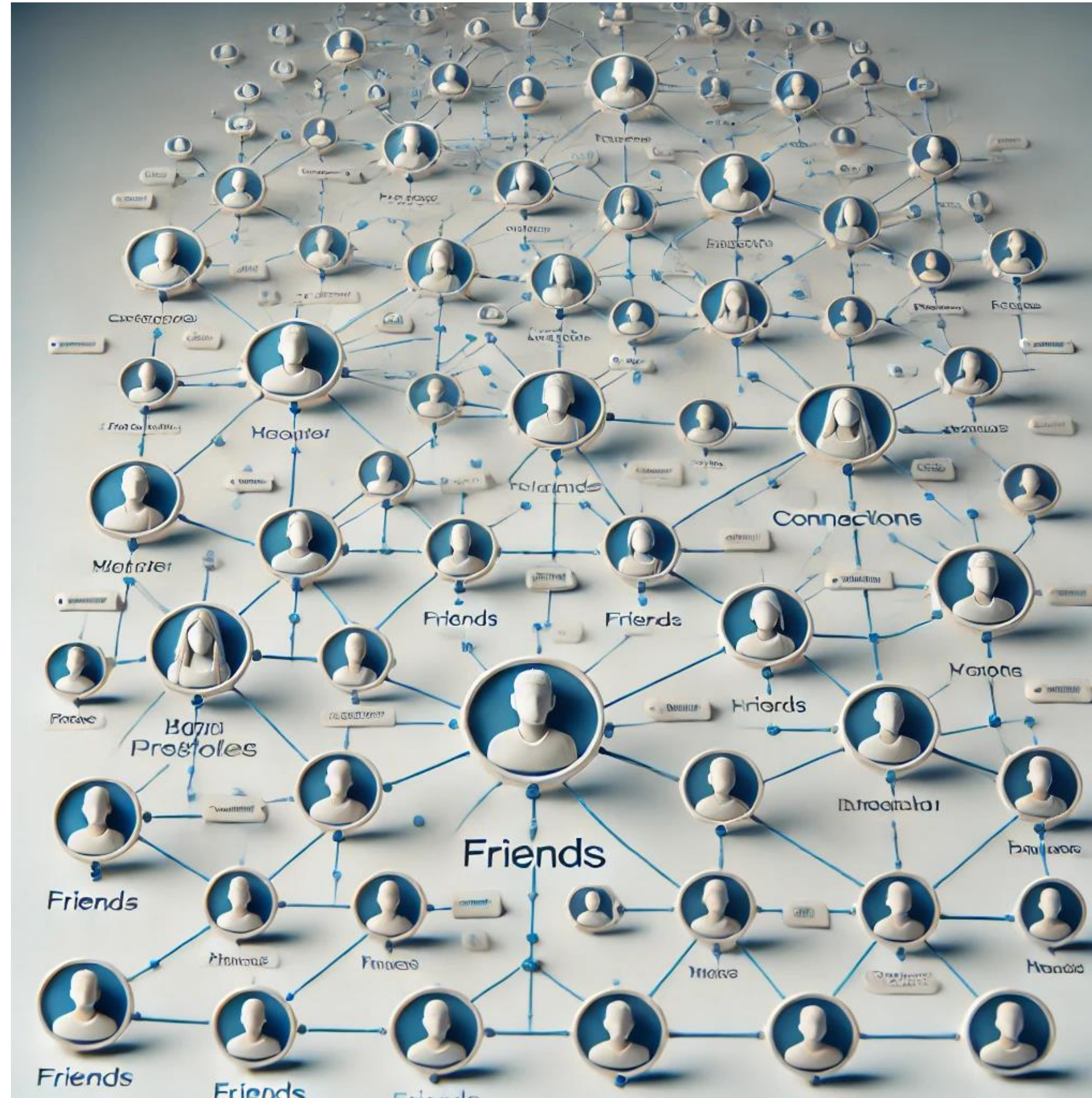
Árvore
De Diretórios



© Prof. Dr. Dilermando Piva Jr.

O que essas figuras tem em comum?

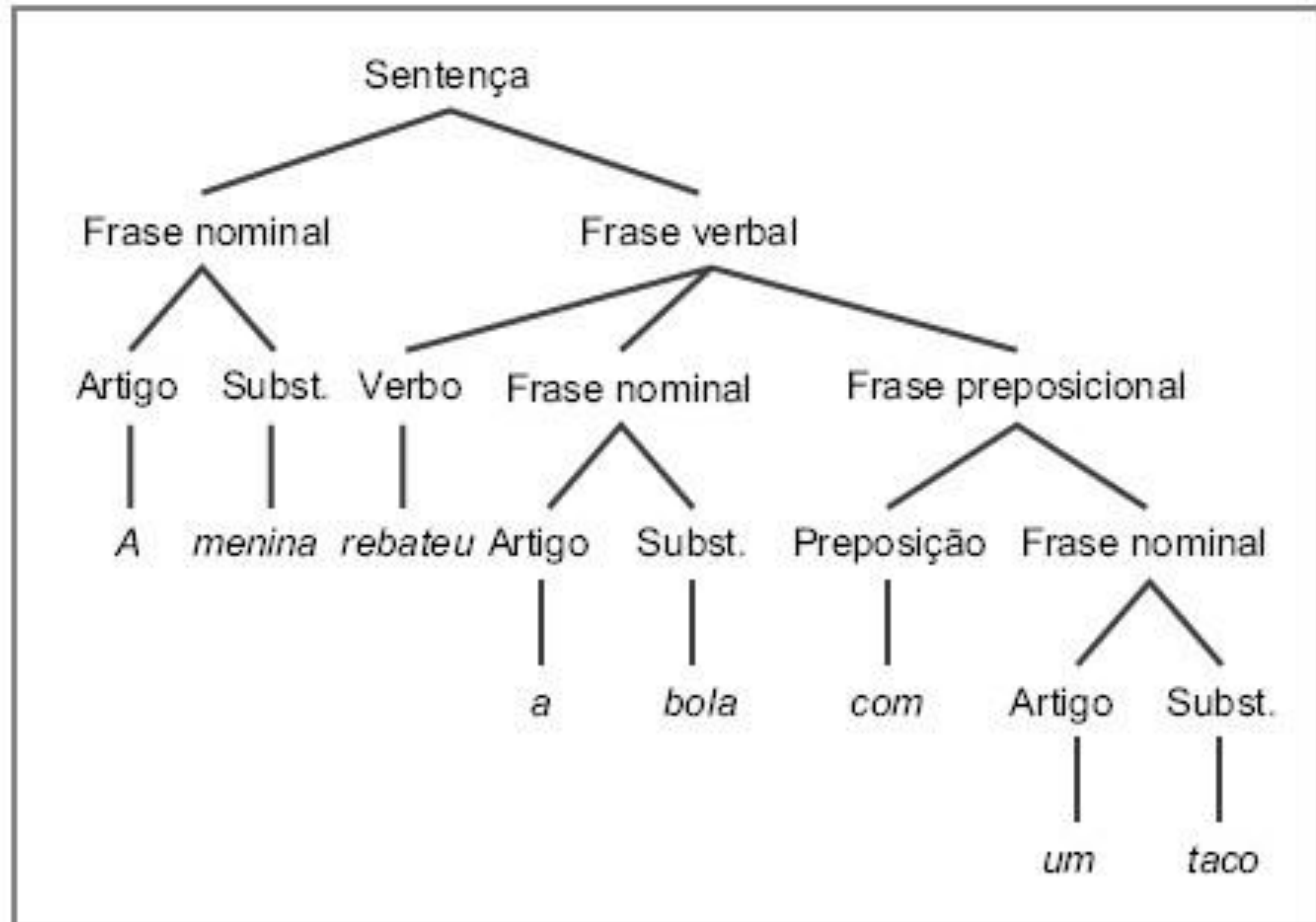
Árvore de Relacionamentos



© Prof. Dr. Dilermando Piva Jr.

O que essas figuras tem em comum?

Árvore de Análise



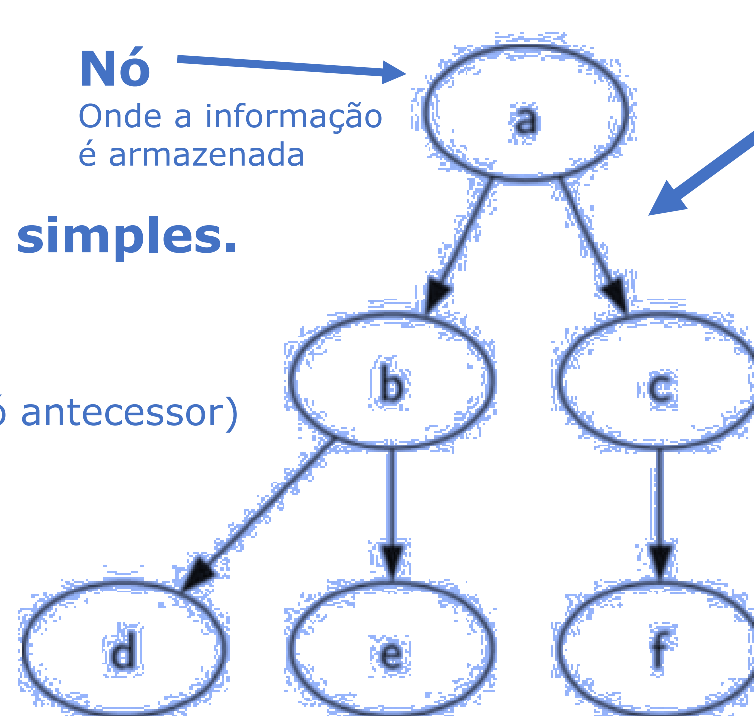
© Prof. Dr. Dilermando Piva Jr.

Árvores

Árvores são estruturas de dados muito utilizadas em diversas áreas da ciência da computação, como sistemas operacionais, bancos de dados e redes de computadores.

Uma estrutura de dados do tipo árvore possui uma raiz a partir da qual diferentes ramos conectam um conjunto de nós intermediários até as folhas da árvore.

Nó
Onde a informação é armazenada



Aresta
Conexões entre os diferentes nós

Nós Filhos e Pais
Nós Irmão

Essa figura ilustra uma árvore simples.
É composta por 6 nós.

a” é a raiz (único nó que não possui nó antecessor)

“d”, “e” e “f” são as folhas.
(nós que não possuem filhos)

Caminho
Sequencia ordenada de nós conectados por arestas
Ex: a → c → f

Nível de um Nó
Nº arestas a partir da raiz. “b” → 1

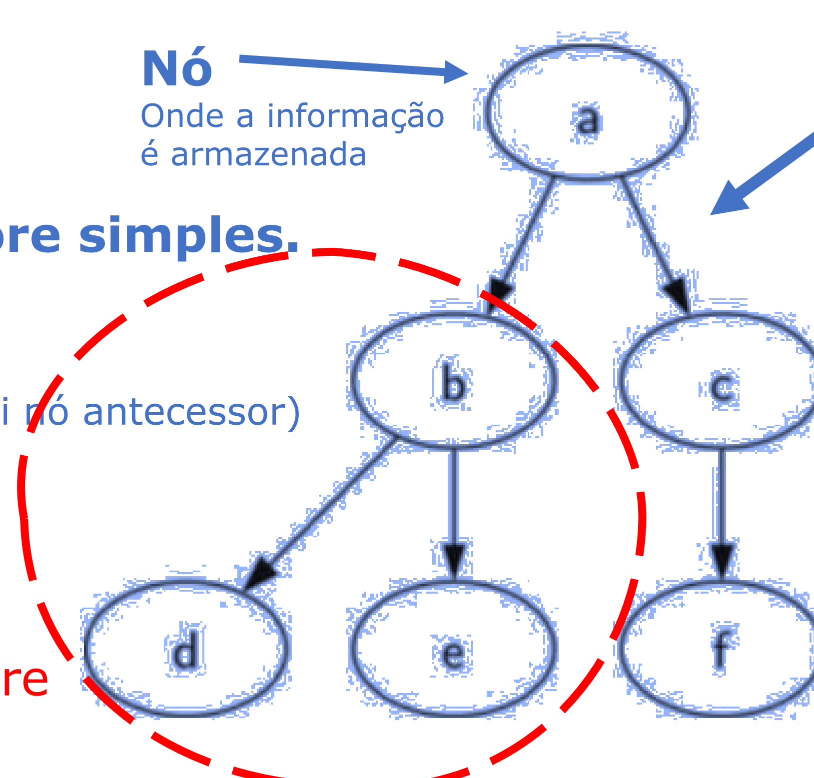
Altura
Maior nível de um nó da árvore.

© Prof. Dr. Dilermando Piva Jr.

Árvores

Árvores são estruturas de dados muito utilizadas em diversas áreas da ciência da computação, como sistemas operacionais, bancos de dados e redes de computadores.

Uma estrutura de dados do tipo árvore possui uma raiz a partir da qual diferentes ramos conectam um conjunto de nós intermediários até as folhas da árvore.



Nó
Onde a informação é armazenada

Aresta
Conexões entre os diferentes nós

Nós Filhos e Pais
Nós Irmão

Essa figura ilustra uma árvore simples.
É composta por 6 nós.

a" é a raiz (único nó que não possui nó antecessor)

"d", "e" e "f" são as folhas.
(nós que não possuem filhos)

Subárvore

Caminho
Sequencia ordenada de nós conectados por arestas
Ex: a → c → f

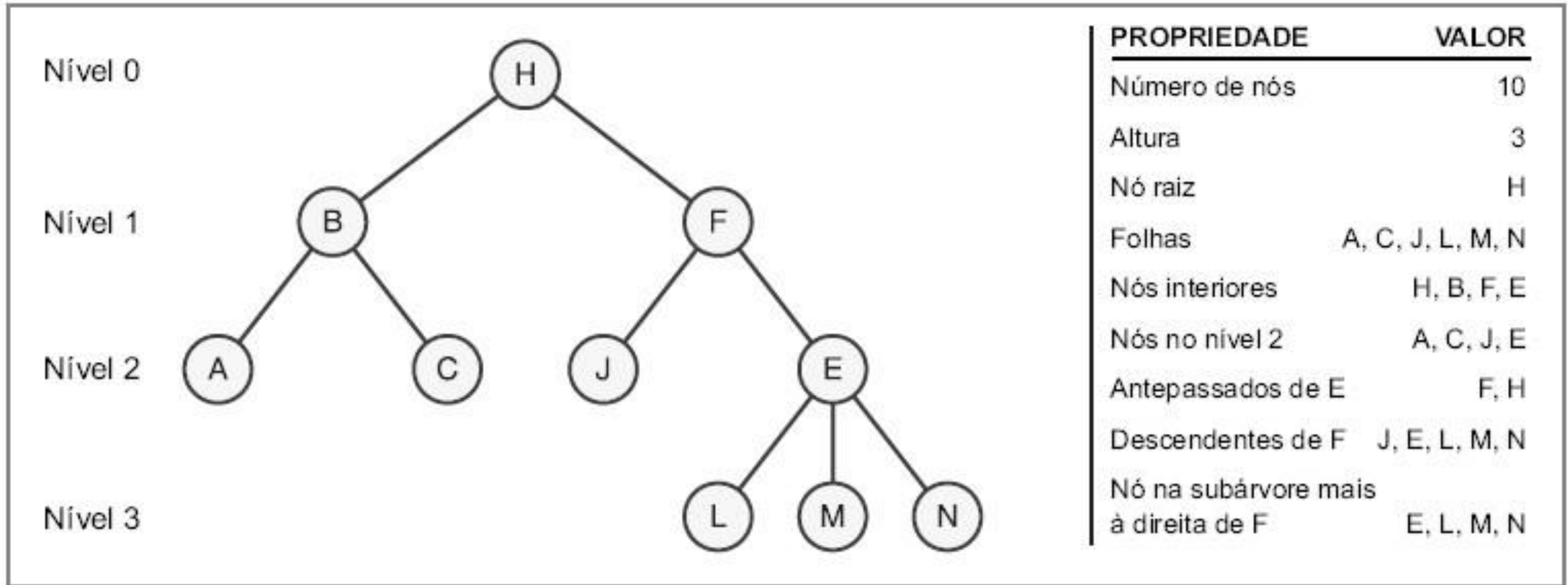
Nível de um Nó
Nº arestas a partir da raiz. "b" → 1

Altura
Maior nível de um nó da árvore.

© Prof. Dr. Dilermando Piva Jr.

Árvores

Recapitulando...



Árvores (definição formal 1)

Uma árvore consiste em um conjunto de nós e um conjunto de arestas que conectam pares de nós. Uma árvore possui as seguintes propriedades:

i) Toda árvore tem um nó designado de raiz, por onde a busca, a inserção e a remoção de elementos deve iniciar. Em outras palavras, é a porta de entrada para o conjunto de dados.

ii) Todo nó n, com exceção da raiz, é conectado por uma aresta a exatamente um nó pai p. Ou seja, cada nó da árvore tem precisamente um único pai.

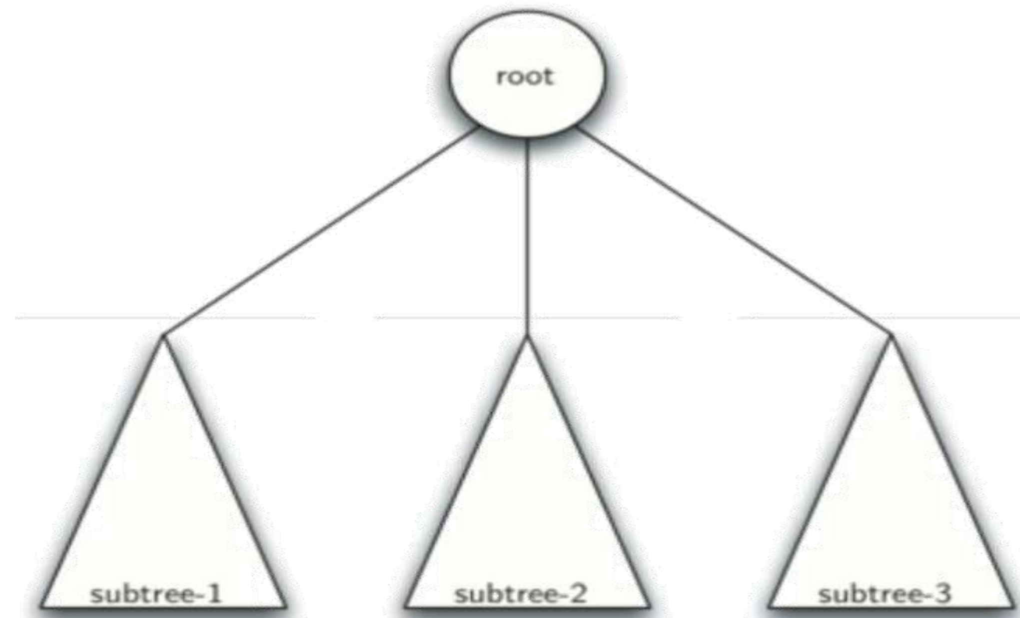
iii) Existe um único caminho saindo da raiz e chegando em um nó arbitrário da árvore. Ou seja, sempre que desejarmos acessar um determinado nó, iremos sempre pelo mesmo caminho.

iv) Se cada nó da árvore possui no máximo dois nós filhos, dizemos que a árvore é uma árvore binária.

© Prof. Dr. Dilermando Piva Jr.

Árvores (definição formal 2)

Uma árvore ou é vazia ou consiste de uma raiz com zero ou mais subárvores, cada uma sendo uma árvore. A raiz de cada subárvore é conectada a raiz da árvore pai por uma aresta.



Pela definição recursiva, sabemos que a árvore acima possui pelo menos 4 nós, uma vez que cada triângulo representando uma subárvore deve possuir uma raiz. Na verdade, ela pode ter muito mais nós do que isso, mas não sabemos pois não conhecemos a estrutura interna de cada subárvore.

© Prof. Dr. Dilermando Piva Jr.

Árvores binárias

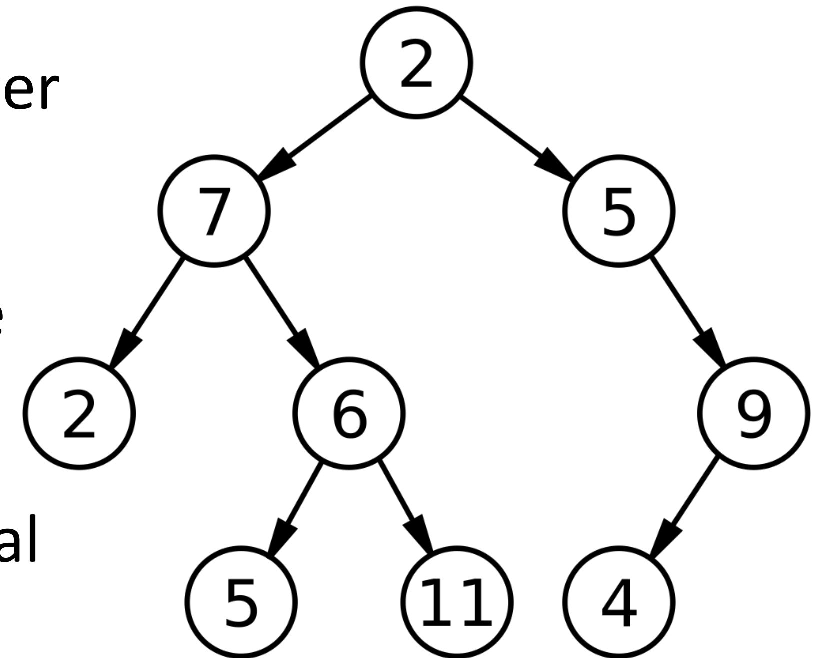
Os nós de uma árvore binária possuem graus zero, um ou dois. Um nó de grau zero é denominado folha.

Em uma árvore binária, por definição, cada nó poderá ter até duas folhas.

A profundidade de um nó é a distância deste nó até a raiz. Um conjunto de nós com a mesma profundidade é denominado nível da árvore. A maior profundidade de um nó, é a altura da árvore.

Uma árvore "estritamente binária" é uma árvore na qual todo nó tem zero ou duas folhas.

Existem autores, porém, que adotam essa definição para o termo quase completa, e utilizam o termo completa apenas para árvores em que todos os níveis têm o máximo número de elementos.

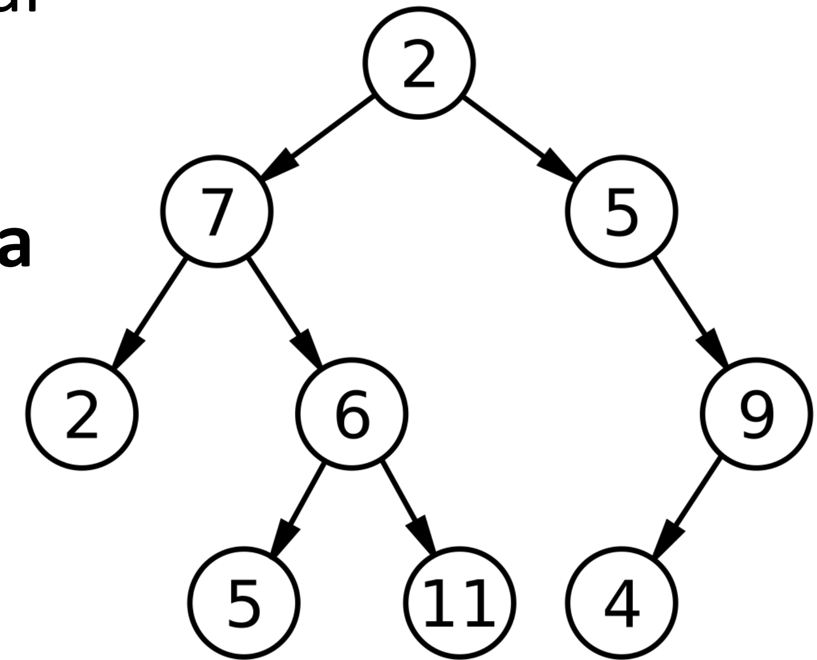


© Prof. Dr. Dilermando Piva Jr.

Árvores binárias

Existem basicamente duas formas de implementar árvores binárias em Python:

1) utilizando uma **lista de sublistas**, ou
2) utilizando encadeamento lógico (como na **lista encadeada**)



Principais Métodos:

Métodos	Descrição
BinaryTree()	cria uma nova instância da árvore binária
get_left_child()	retorna a subárvore a esquerda do nó corrente
get_right_child()	retorna a subárvore a direita do nó corrente
set_root_val(val)	armazena um valor no nó corrente
get_root_val()	retorna o valor armazenado no nó corrente
insert_left(val)	cria uma nova árvore binária a esquerda do nó corrente
insert_right(val)	cria uma nova árvore binária a direita do nó corrente

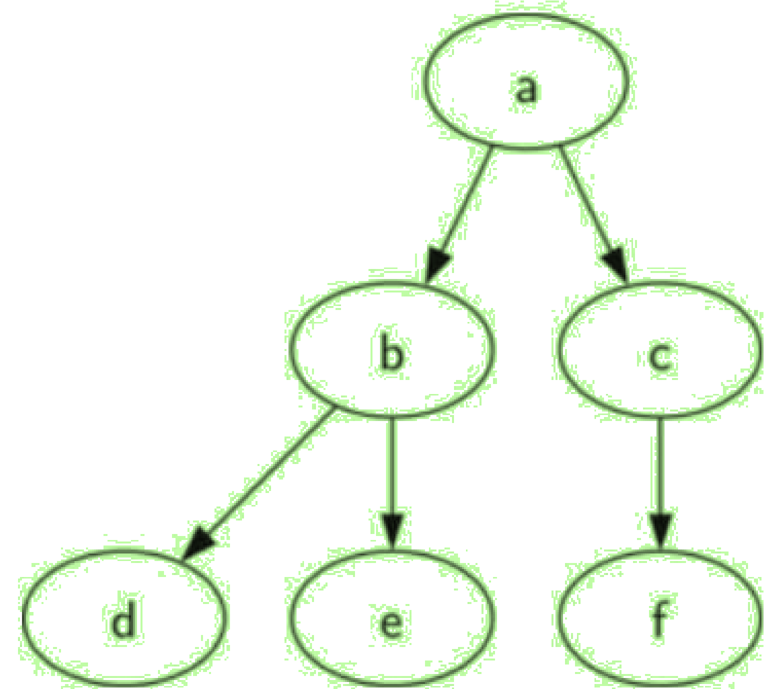
© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS DE LISTAS

Em uma representação de lista de sublistas, o valor armazenado na raiz é sempre o primeiro elemento da lista. O segundo elemento da lista será a sublista que representa a subárvore a esquerda. Analogamente, o terceiro elemento da lista será a sublista que representa a subárvore a direita.

Para ilustrar como fica a representação da árvore da figura ao lado, apresentamos o código a seguir.

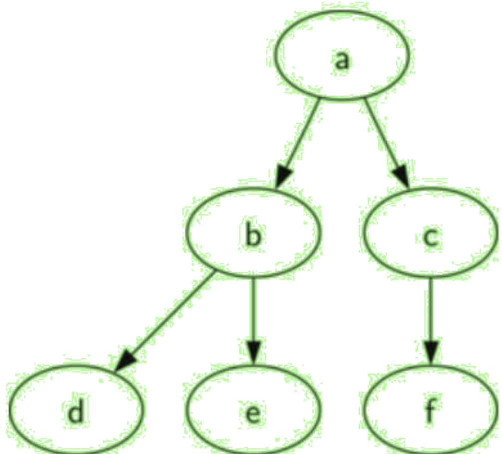
```
# Representação de árvore como lista de sublistas
my_tree =   ['a',   # raiz
                ['b',   # subárvore esquerda
                    ['d', []  []],
                    ['e'. [], []]
                ],
                ['c',   # subárvore direita
                    ['f', [], []],
                    []
                ]
            ]
```



© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS DE LISTAS

```
# Representação de árvore como lista de sublistas
my_tree =   ['a',   # raiz
                ['b',   # subárvore esquerda
                    ['d', []  []],
                    ['e'. [], []]
                ],
                ['c',   # subárvore direita
                    ['f', [], []],
                    []
                ]
            ]
```



Uma propriedade muito interessante desta representação é que a estrutura de uma lista representando uma subárvore possui a mesma estrutura de uma árvore (lista), de modo que define uma representação recursiva.

```
print(my_tree)
print('Subárvore esquerda = ', my_tree[1])
print('Raiz = ', my_tree[0])
print('Subárvore direita = ', my_tree[2])
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS DE LISTAS (IMPLEMENTAÇÃO)

```python
# Cria árvore binária com raiz r
def binary_tree(r):
    return [r, [], []]

# Insere novo ramo a esquerda da raiz
def insert_left(root, new_branch):
    # Analisa a subárvore a esquerda
    t = root.pop(1)
    # Se a subárvore a esquerda não é vazia
    if len(t) > 1:
        # Insere na posição 1 da raiz (esquerda)
        # Novo ramo será a raiz da subárvore a esquerda
        # Adiciona t na esquerda do novo ramo
        root.insert(1, [new_branch, t, []])
    else:
        # Se t for vazia, não há subárvore a esquerda
        root.insert(1, [new_branch, [], []])
    return root
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS DE LISTAS (IMPLEMENTAÇÃO)

```
# Insere novo ramo a direita da raiz
def insert_right(root, new_branch):
    # Analisa a subárvore a direita
    t = root.pop(2)
    # Se a subárvore a direita não é vazia
    if len(t) > 1:
        # Insere na posição 2 da raiz (direita)
        # Novo ramo será a raiz da subárvore a direita
        # Adiciona t na direita do novo ramo
        root.insert(2, [new_branch, [], t])
    else:
        # Se t for vazia, não há subárvore a direita
        root.insert(2, [new_branch, [], []])
    return root
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS DE LISTAS (IMPLEMENTAÇÃO)

```python
def get_root_val(root):
    return root[0]

def set_root_val(root, new_val):
    root[0] = new_val

def get_left_child(root):
    return root[1]

def get_right_child(root):
    return root[2]
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS DE LISTAS (TESTANDO A IMPL.)

```python
# Cria árvore binária
r = binary_tree(3)
# Adiciona subárvore a esquerda
insert_left(r, 4)
# Adiciona subárvore a esquerda
insert_left(r, 5)
# Adiciona subárvore a direita
insert_right(r, 6)
# Adiciona subárvore a direita
insert_right(r, 7)
print(r)

# Obtém subárvore a esquerda da raiz
l = get_left_child(r)
print(l)
# Muda a raiz da subárvore a esquerda
set_root_val(l, 9)
print(r)
# Insere a esquerda da subárvore a esquerda
insert_left(l, 11)
print(r)
```

© Prof. Dr. Dilermando Piva Jr.

VAMOS PARA A PRÁTICA ?!!!

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS DE LISTAS (EXERCÍCIO)

Considere o seguinte código em Python.

```
x = binary_tree('a')
insert_left(x,'b')
insert_right(x,'c')
insert_right(get_right_child(x), 'd')
insert_left(get_right_child(get_right_child(x)), 'e')
```

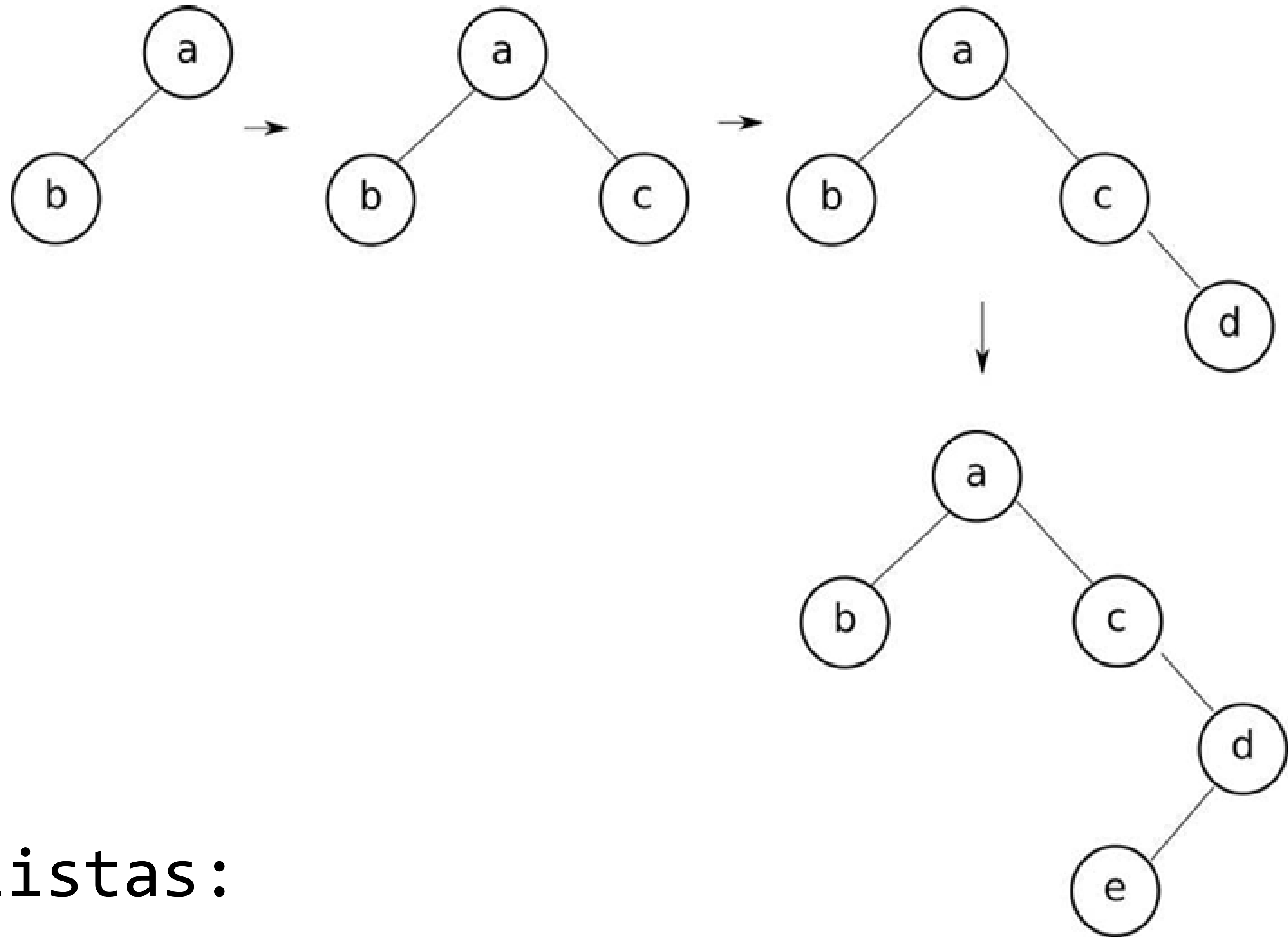
Desenhe a árvore resultante e forneça a representação usando lista de listas

© Prof. Dr. Dilermando Piva Jr.

AGORA É COM VOCÊ !!!!

© Prof. Dr. Dilermando Piva Jr.

Árvore Resultante:



Representação com listas:

[‘a’, [‘b’, [], []], [‘c’, [], [‘d’, [‘e’, [], []], []]]]

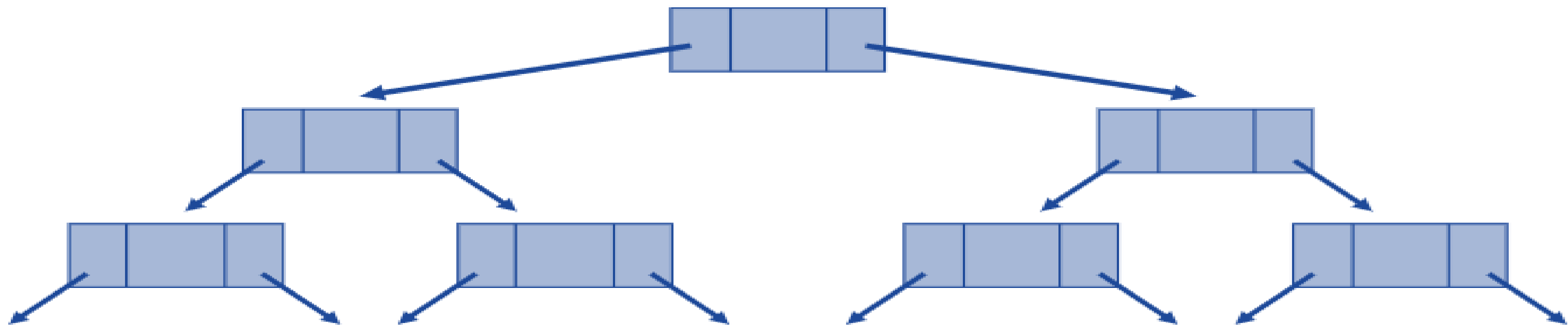
© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: LISTAS LIGADAS (Representação com referências)

Apesar de interessante, a representação de árvores binárias utilizando lista de listas não é muito intuitiva, principalmente quando o número de nós da árvore cresce.

O número de sublistas fica tão elevado que é muito fácil cometer erros e equívocos durante a manipulação da estrutura.

Sendo assim, veremos agora, como representar árvores binárias utilizando referências e um encadeamento lógico similar ao adotado nas listas duplamente encadeadas, em que cada nó possui duas referências: uma para o nó filho a esquerda e outra para o nó filho a direita.



© Prof. Dr. Dilermando Piva Jr.

Árvores Binárias...
Uma primeira implementação

```python
class BinaryTree:
    # Construtor
    def __init__(self, valor):
        self.key = valor
        self.left_child = None
        self.right_child = None

    # Insere nó a esquerda
    def insert_left(self, valor):
        # Se nó corrente não tem filho a esquerda, OK
        if self.left_child == None:
            self.left_child = BinaryTree(valor)
        else:
            # Se tem filho a esquerda, pendura subárvore
            # a esquerda do nó corrente na esquerda do
            # novo nó recem criado
            temp = BinaryTree(valor)
            temp.left_child = self.left_child
            self.left_child = temp
```

```python
    # Insere nó a direita
    def insert_right(self, valor):
        # Se nó corrente não tem filho a direita, OK
        if self.right_child == None:
            self.right_child = BinaryTree(valor)
        else:
            # Se tem filho a direita, pendura subárvore
            # a direita do nó corrente na direita do
            # novo nó recem criado
            temp = BinaryTree(valor)
            temp.right_child = self.right_child
            self.right_child = temp

    # Obtém filho a direita
    def get_right_child(self):
        return self.right_child
    # Obtém filho a esquerda
    def get_left_child(self):
        return self.left_child
    # Atualiza valor do nó corrente
    def set_root_val(self, valor):
        self.key = valor
    # Obtém valor do nó corrente
    def get_root_val(self):
        return self.key
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binárias...

Testando...

```python
# Cria nó com valor 'a'
r = BinaryTree('a')
print(r.get_root_val())
print(r.get_left_child())
print(r.get_right_child())

# Insere nó com valor 'b' a esquerda da raiz
r.insert_left('b')
print(r.get_left_child().get_root_val())

# Insere nó com valor 'c' na direita da raiz
r.insert_right('c')
print(r.get_right_child().get_root_val())

# Insere nó com valor 'd' a esquerda no filho a esquerda da raiz
r.get_left_child().insert_left('d')
print(r.get_left_child().get_left_child().get_root_val())

# Insere nó com valor 'e' a direita no filho a esquerda da raiz
r.get_left_child().insert_right('e')
print(r.get_left_child().get_right_child().get_root_val())
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

Há basicamente 3 formas de navegar por uma árvore binária: preorder, inorder e postorder.

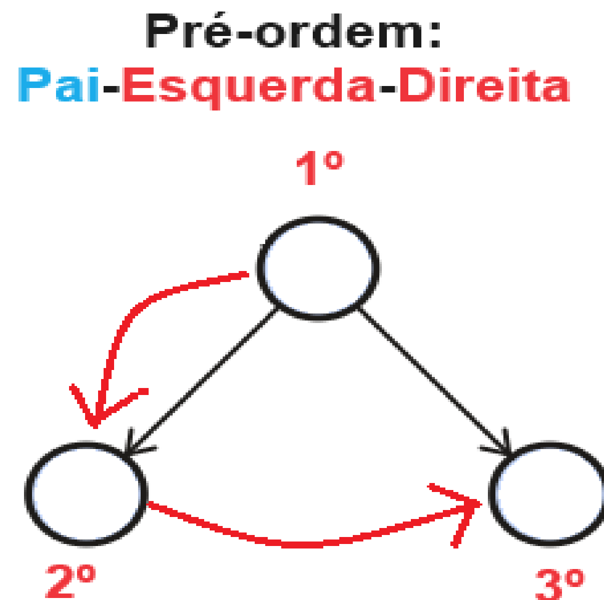
- **Preorder**: visitamos primeiramente a raiz da árvore, depois recursivamente a subárvore a esquerda e finalmente recursivamente a subárvore a direita.
- **Inorder**: visitamos recursivamente a subárvore a esquerda, depois passamos pela raiz da árvore e por fim visitamos recursivamente a subárvore a direita.
- **Postorder**: visitamos recursivamente a subárvore a esquerda, depois visitamos recursivamente a subárvore a direita e por fim passamos pela raiz

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

Pré-ordem: No percurso pré-ordem, ou “pre order”, os nós são visitados na seguinte ordem: primeiro o nó pai, em seguida o filho esquerdo e, por último, o filho direito.

Essa técnica é útil para criar uma cópia da árvore ou para realizar operações de pré-processamento antes de visitar os nós filhos.

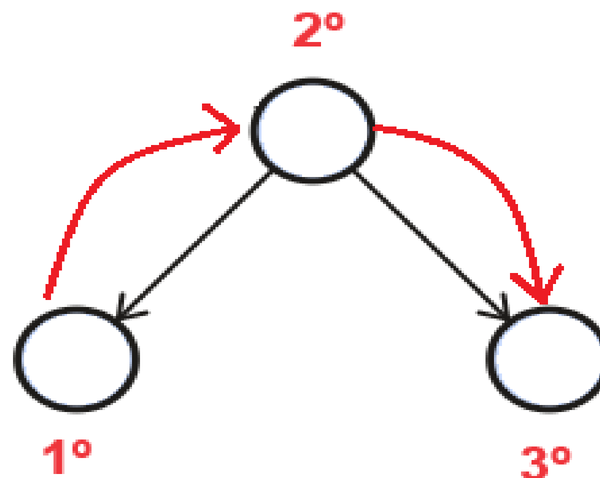


© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

Em ordem (Simétrico): No percurso em ordem, também conhecido como “in order”, a árvore é percorrida de forma que os nós sejam visitados na seguinte ordem: primeiro o filho esquerdo, depois o nó pai e, por fim, o filho direito.

Essa técnica é comumente utilizada para obter os elementos de uma árvore binária de busca em ordem crescente.

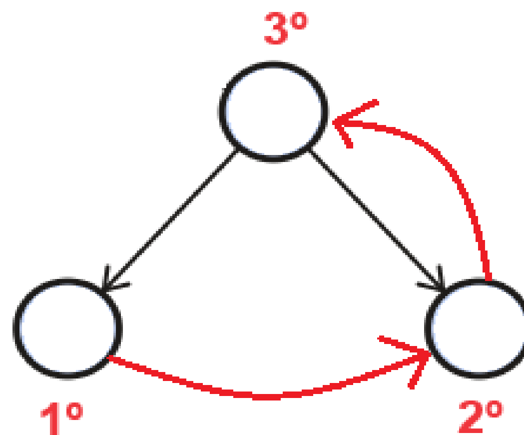


© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

Pós-ordem: O percurso pós-ordem, ou “post order”, envolve a visita aos nós na seguinte ordem: primeiro o filho esquerdo, depois o filho direito e, por último, o nó pai.

Essa técnica é frequentemente empregada em cálculos que requerem informações dos nós filhos antes de processar o nó pai, como a avaliação de expressões aritméticas.



© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

A dica pra entender rapidamente esses métodos de percursos em árvores binárias é seguir essas duas regras:

- O nó filho da esquerda é sempre visitado antes que o da direita:
- O nome do método se refere ao momento em que o nó pai é visitado, ou seja:
  - **Pré-ordem**: o pai é visitado antes dos filhos, ou seja, Pai-Esquerda-Direita;
  - **Em ordem**: o pai é visitado entre os filhos, ou seja, Esquerda-Pai-Direita;
  - **Pós-ordem**: o pai é visitado após os filhos, ou seja, Esquerda-Direita-Pai;

Pré-ordem: Pai - Esquerda - Direita

Em ordem: Esquerda - Pai - Direita

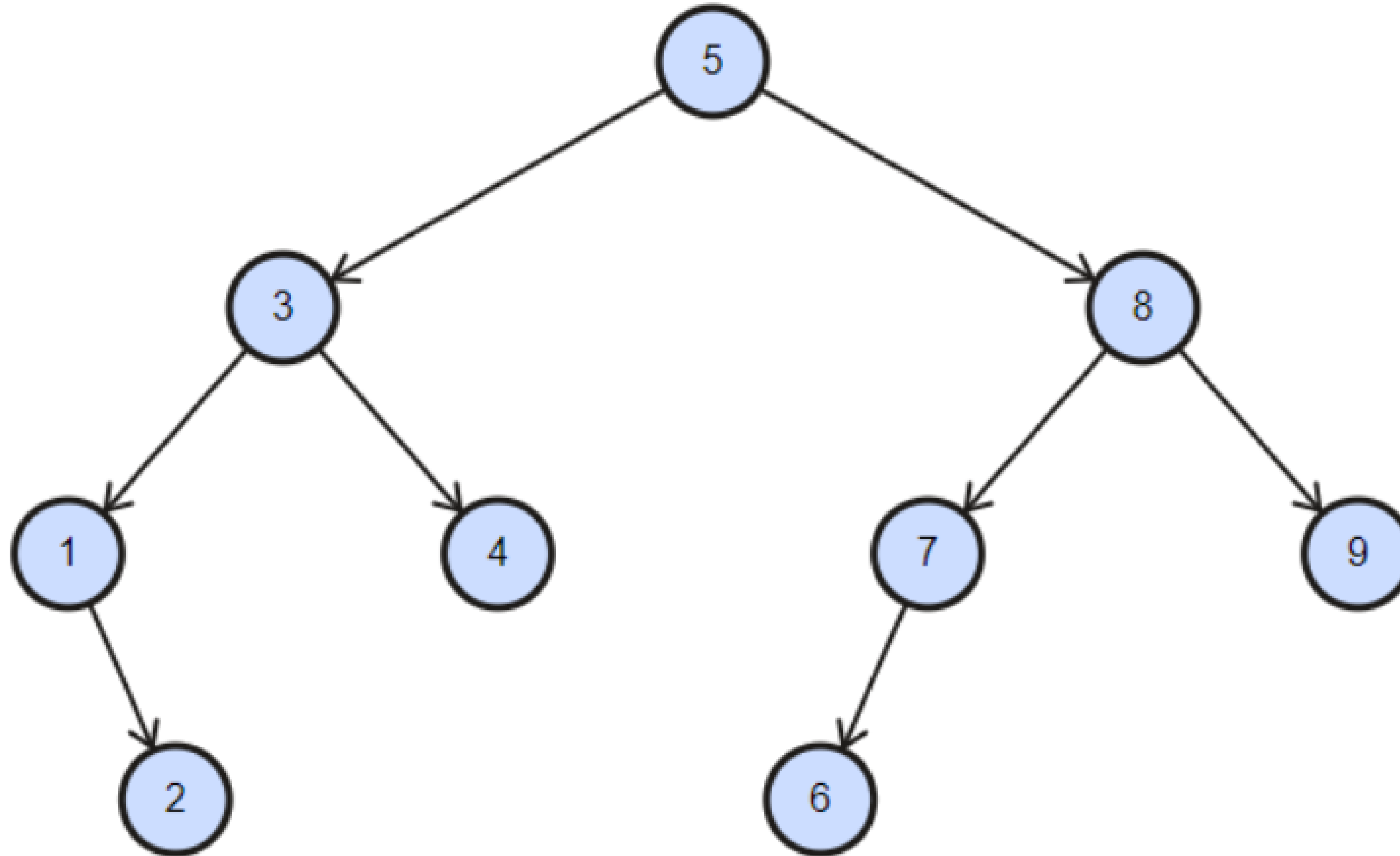
Pós-ordem: Esquerda - Direita - Pai

Obs: O nó esquerdo é visitado antes do nó direito em *qualquer método* de percurso em árvore binária.

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

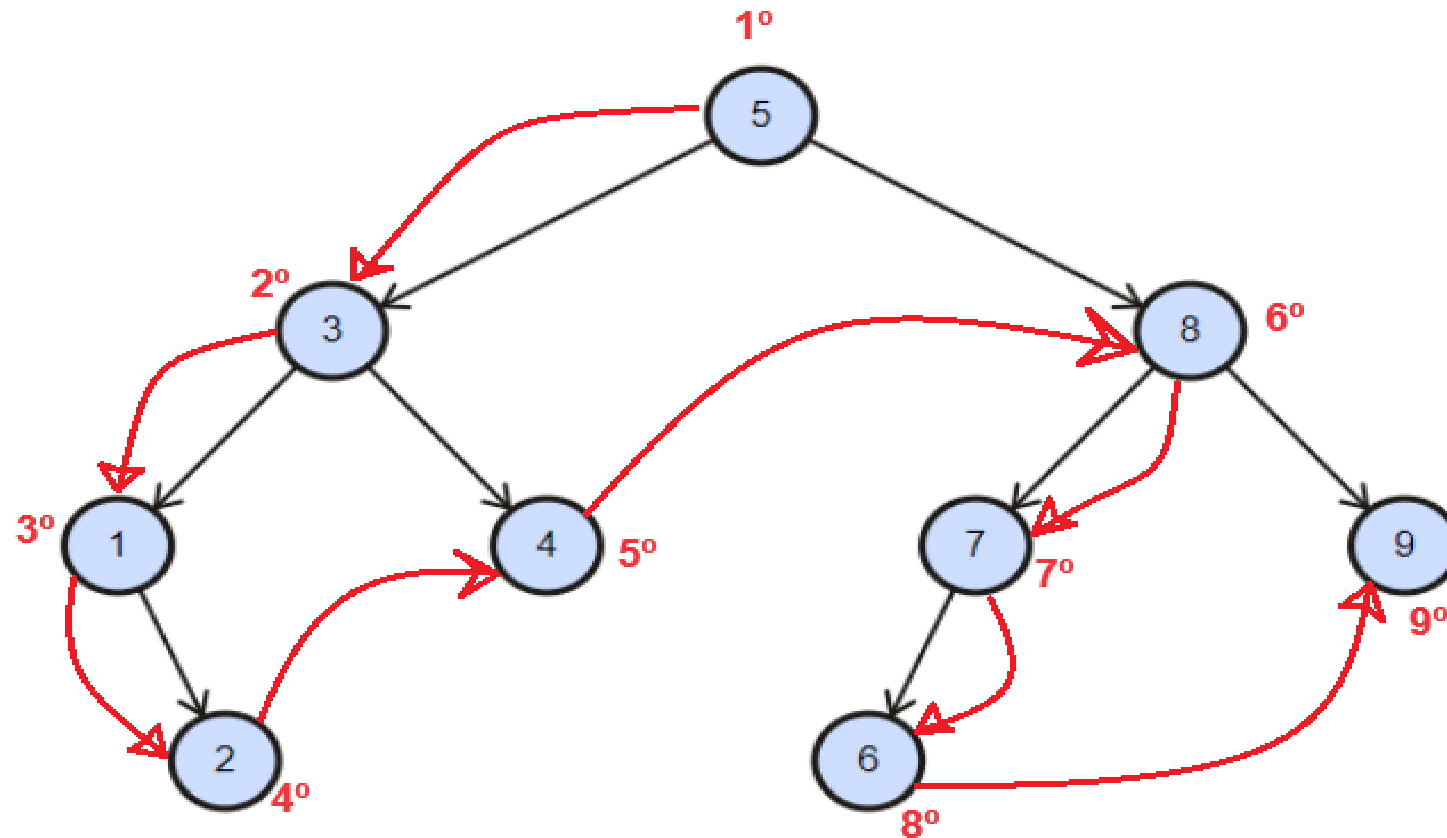
Exemplo:



© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

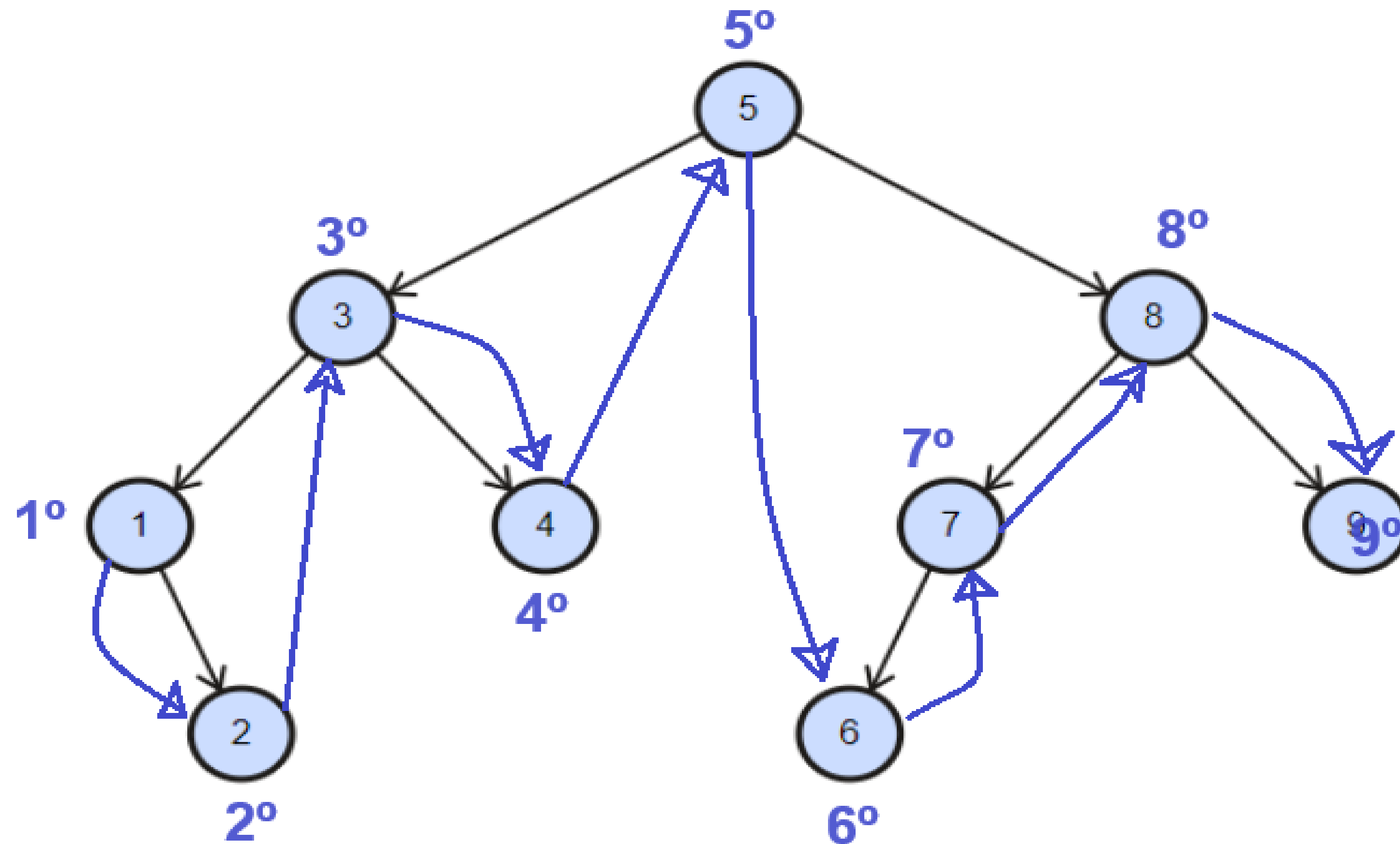
Exemplo:



© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE

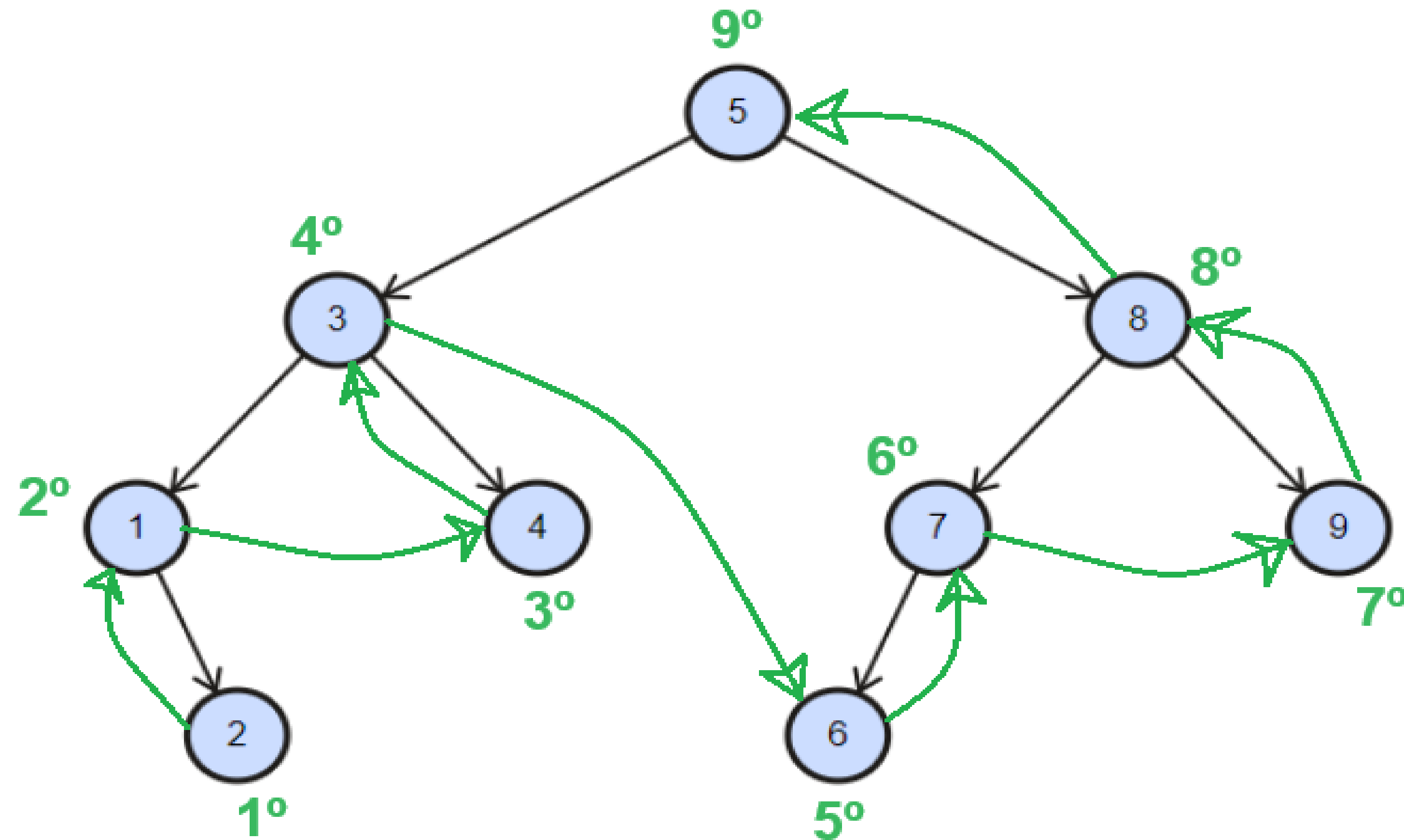
Exemplo:



Árvores Binária: PERCORRENDO UMA ÁRVORE

Exemplo:

Percurso Pós-ordem



1	2	3	4	5	6	7	8	9
2	**1**	**4**	**3**	**6**	**7**	**9**	**8**	**5**

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE *(implementação)*

```
# Percorre uma árvore binária em Preorder
def preorder(self):
    # Imprime valor da raiz
    print(self.key)
    # Visita subárvore a esquerda
    if self.left_child:
        self.left.preorder()
    # Visita subárvore a direita
    if self.right_child:
        self.right.preorder()
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE *(implementação)*

```
# Percorre uma árvore binária em Inorder
def inorder(self):
    # Visita subárvore a esquerda
    if self.left_child:
        self.left.inorder()
    # Imprime valor da raiz
    print(self.key)
    # Visita subárvore a direita
    if self.right_child:
        self.right.inorder()
```

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária: PERCORRENDO UMA ÁRVORE *(implementação)*

```
# Percorre uma árvore binária em Postorder
def postorder(self):
    # Visita subárvore a esquerda
    if self.left_child:
        self.left.postorder()
    # Visita subárvore a direita
    if self.right_child:
        self.right.postorder()
    # Imprime valor da raiz
    print(self.key)
```

© Prof. Dr. Dilermando Piva Jr.

VAMOS PARA A PRÁTICA ?!!!

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária de Busca: LISTAS LIGADAS

Apesar de funcional, a estrutura de dados implementada pela classe `BinaryTree` não é otimizada para busca de elementos no conjunto.

Para essa finalidade, veremos que existe uma classe de árvores mais adequada: **as árvores binárias de busca** (BynarySearchTree).

Uma árvore binária de busca implementa um TDA Map (tipo de dados abstrado – Map), que é uma estrutura que mapeia uma chave a um valor. Neste tipo de estrutura de dados, nós não estamos interessados na localização exata dos elementos na árvore, mas sim em utilizar a estrutura da árvore binária para realizar busca de maneira eficiente.

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária de Busca: LISTAS LIGADAS

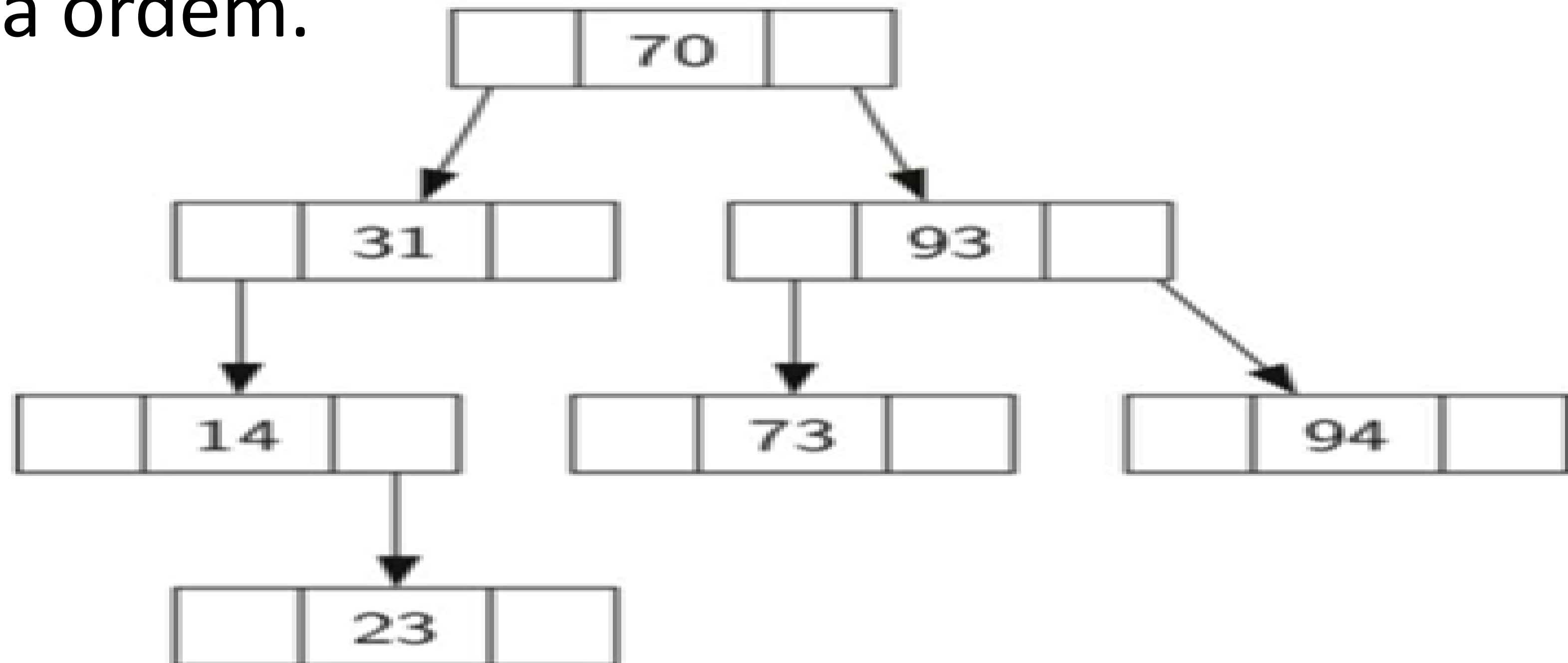
A seguir apresentamos os principais métodos da classe `BinarySearchTree`.

Método	Operação
Map()	cria um mapeamento vazio
put(key, val)	adiciona um novo par chave-valor ao mapeamento (se chave já existe, atualiza o valor referente a ela)
get(key)	retorna o valor associado a chave
del map[key]	deleta o par chave-valor do mapeamento
len()	retorna o número de pares chave-valor no mapeamento
in	retorna True se chave pertence ao mapeamento (key in map)

© Prof. Dr. Dilermando Piva Jr.

Árvores Binária de Busca: LISTAS LIGADAS

Propriedade chave: em uma árvore binária de busca, chaves menores que a chave do nó pai devem estar na subárvore a esquerda e chaves maiores que a chave do nó pai devem estar na subárvore a direita. Por exemplo, suponha que desejamos criar uma árvore binária de busca com os seguintes valores: 70, 31, 93, 94, 14, 23, 73, nessa ordem.



© Prof. Dr. Dilermando Piva Jr.

VAMOS PARA A PRÁTICA ?!!!

© Prof. Dr. Dilermando Piva Jr.