UniEVANGÉLICA
UNIVERSIDADE EVANGÉLICA DE GOIÁS

Estrutura de Dados

Árvores Binárias de Busca

Conceitos Introdutórios

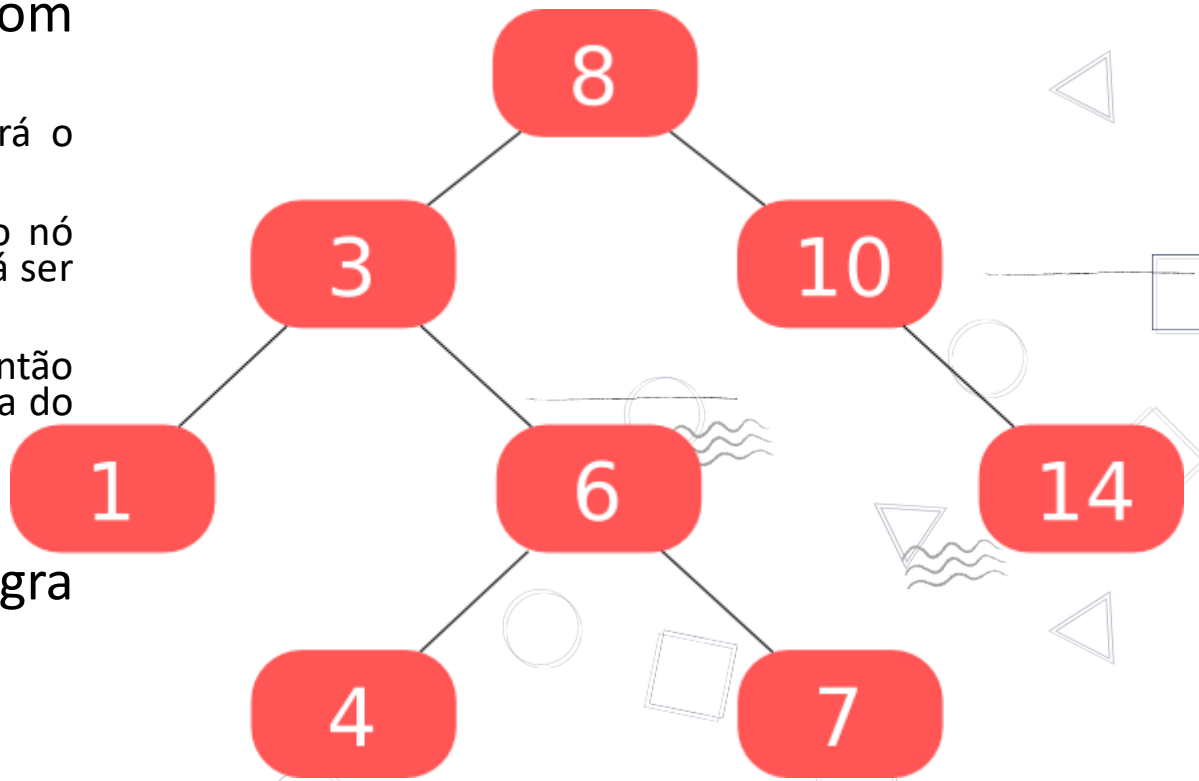
Prof.(a): Me. William P. Santos Júnior

Árvores Binárias de Busca

Árvores de busca são árvores onde seus nós(vértices) são organizados de acordo com algumas propriedades:

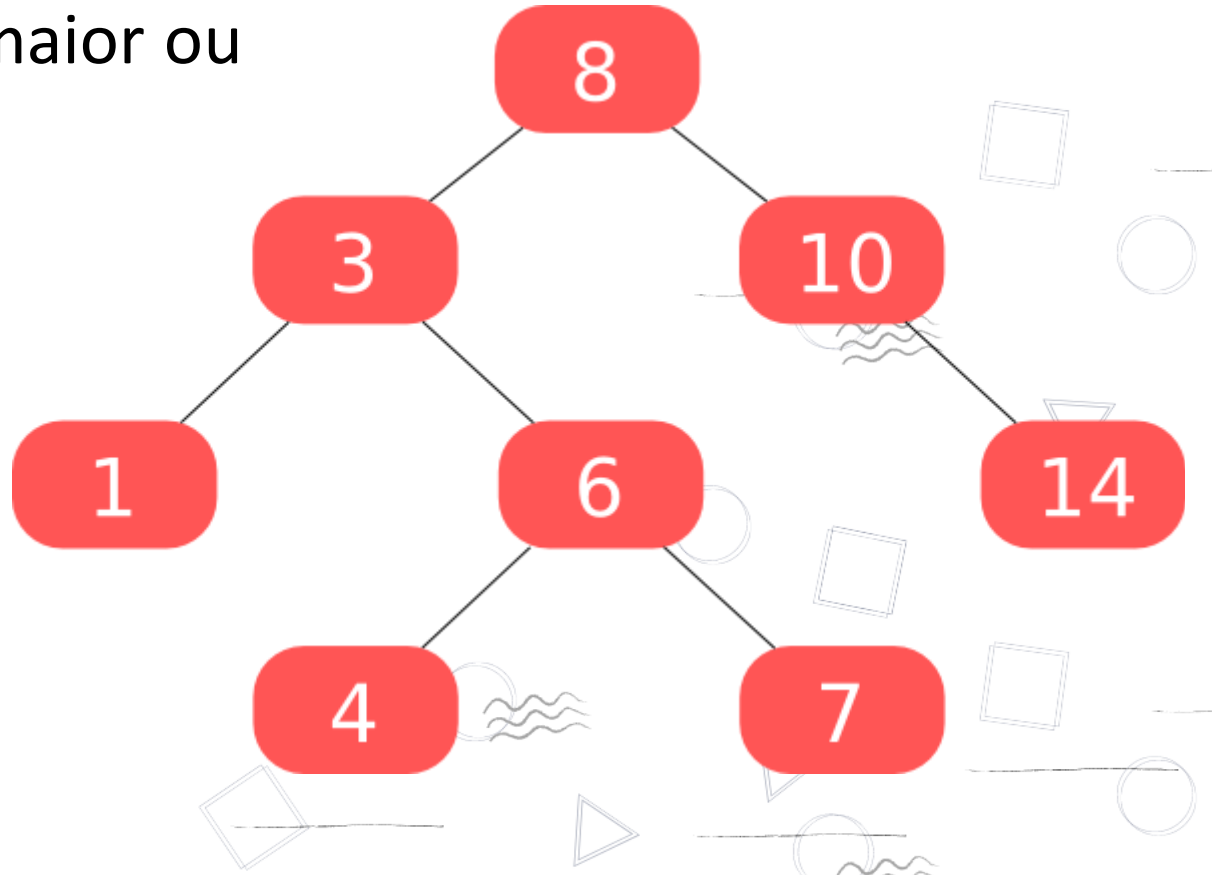
- Se não existir ainda uma raiz, então o novo elemento será o próprio nó raiz;
- Deve-se comparar o próximo elemento a ser inserido com o nó raiz, se esse novo elemento for *menor* que o nó raiz ele deverá ser inserido na sub-árvore esquerda do nó raiz.
- Se o novo elemento a ser inserido for *maior* que o nó raiz, então esse novo elemento deverá ser inserido na sub-árvore a direita do nó raíz.

As demais inserções seguem a regra recursivamente.



Árvores Binárias de Busca

O filho do primeiro nó tem que ter a chave menor que seu pai e o filho a esquerda tem que ter a chave maior ou igual ao de seu pai;



Árvores Binárias de Busca

Implementação

```
class No:
  def __init__(self, valor):
    self.valor    = valor
    self.esquerda = None
    self.direita  = None

  def mostra_no(self):
    print(self.valor)
```

```
class ArvoreBinariaBusca:
  def __init__(self):
    self.raiz = None
```

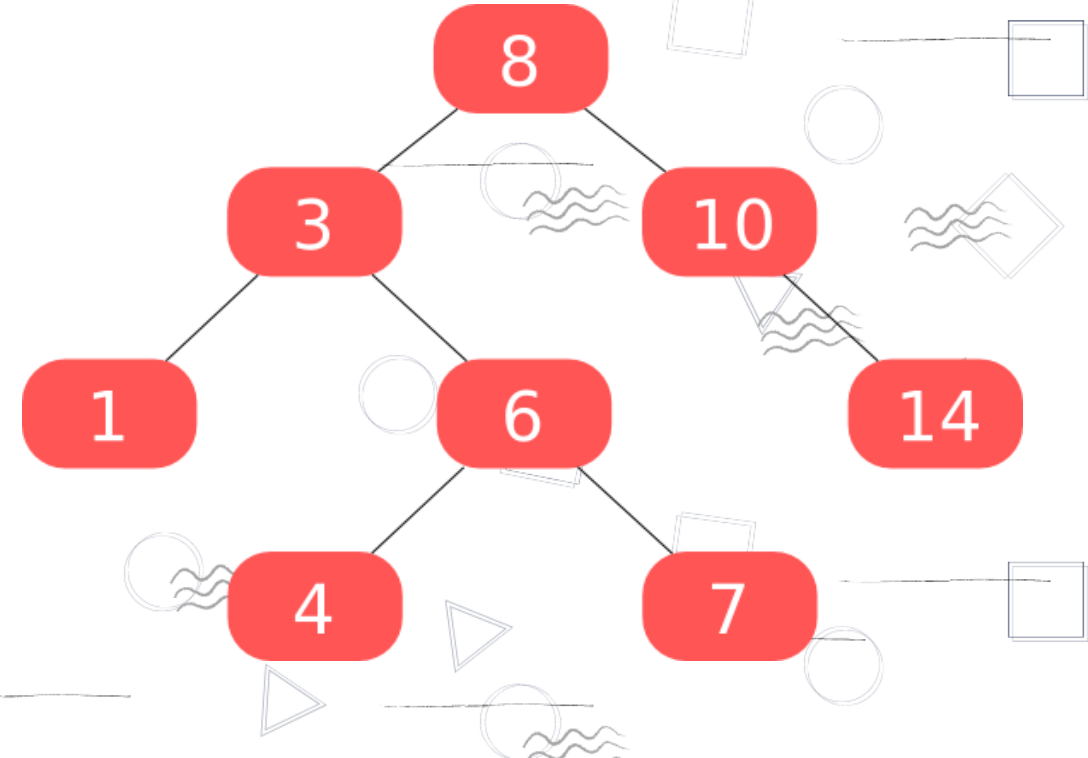
Árvores Binárias de Busca - Inserção

O local para inserir um novo nó deve ser encontrado;

Segue-se o caminho da raiz até o devido nó, que será pai do novo nó;

Quando o nó pai for localizado, o novo nó será conectado como seu filho, a esquerda ou a direita dependendo do valor da chave do nó que pode ser menor ou maior que a chave do nó pai.

Ex: https://visualgo.net/en/bst



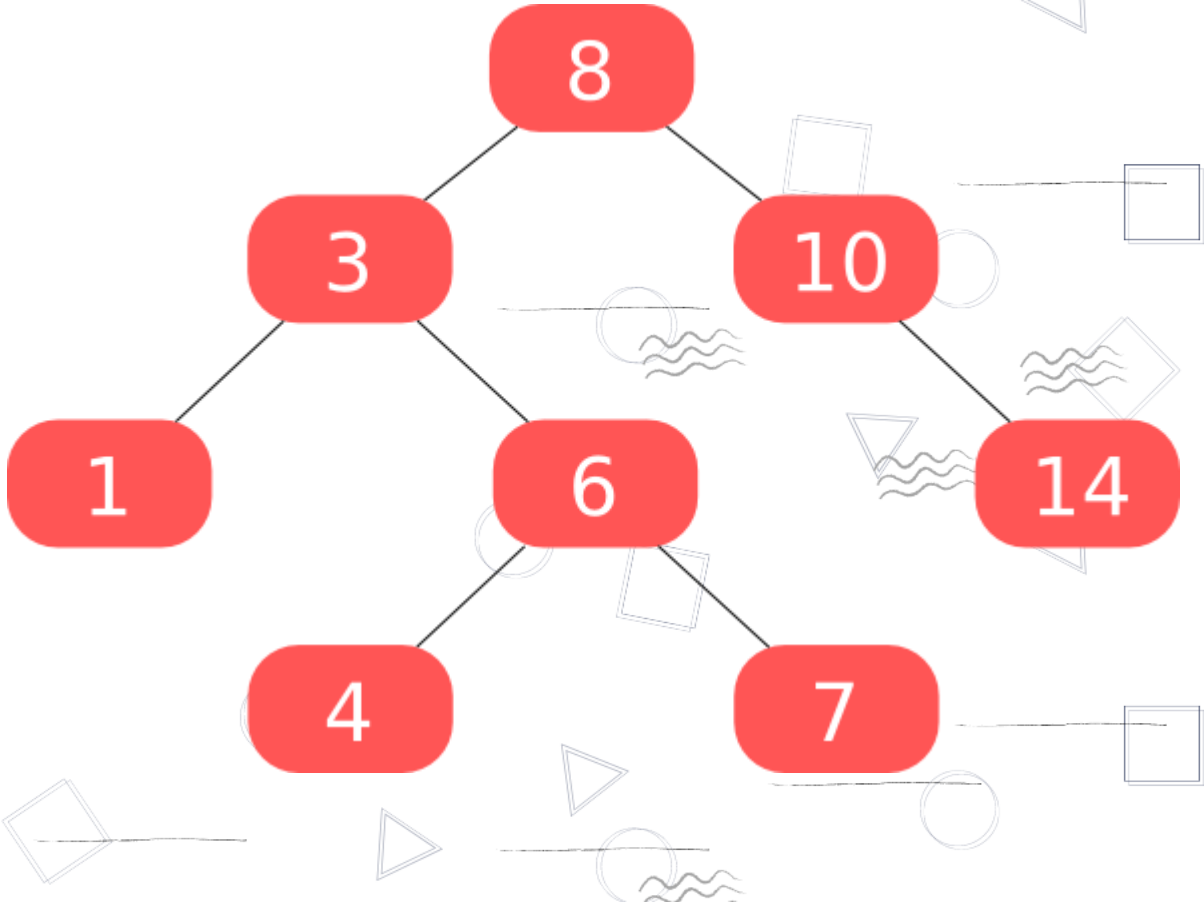
Árvores Binárias de Busca - Inserção

```
[1]  class No:
       def __init__(self, valor):
         self.valor    = valor
         self.esquerda = None
         self.direita  = None

       def mostra_no(self):
         print(self.valor)
```

```
class ArvoreBinariaBusca:
  def __init__(self):
    self.raiz = None

  def inserir(self, valor):
    novo = No(valor)
    #verificando se a árvore está vazia
    if self.raiz == None:
      self.raiz = novo
    else:
      atual = self.raiz
      while True:
        pai = atual
        # percorrendo a Esquerda
        if valor < atual.valor:
          atual = atual.esquerda
          if atual == None:
            pai.esquerda = novo
            return
        # percorrendo a Direita
        else:
          atual = atual.direita
          if atual == None:
            pai.direita = novo
            return
```



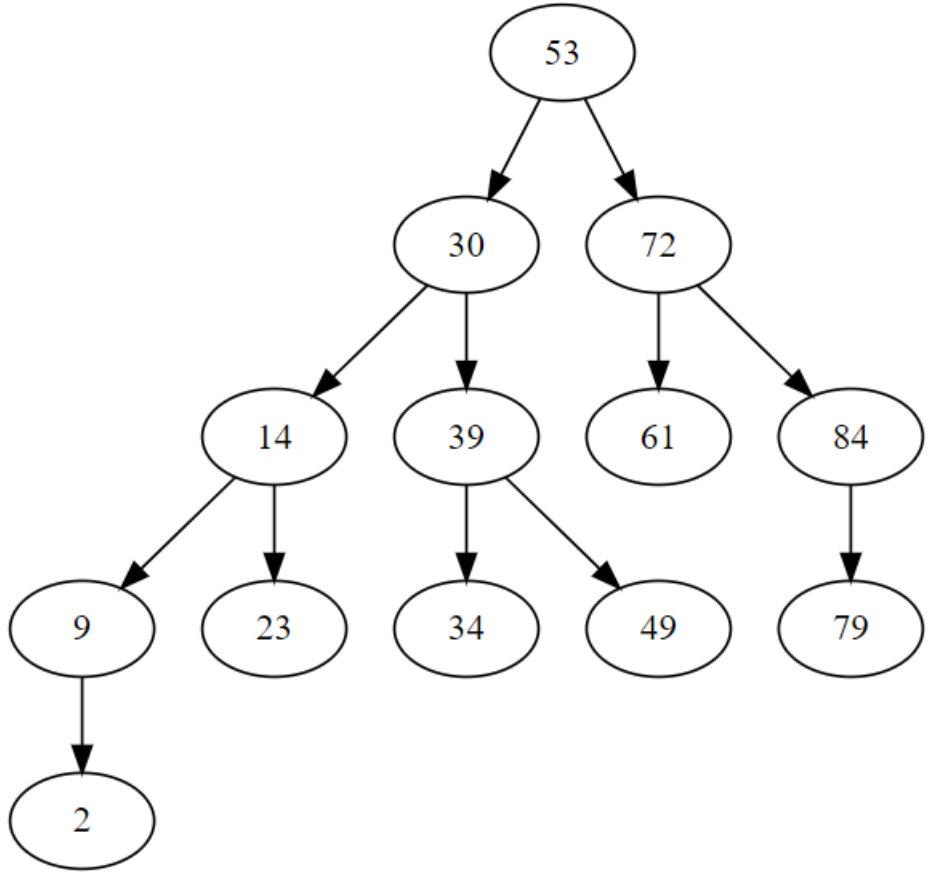
Travessia Pré-Ordem

Primeiro a raiz da árvore é visitada, e recursivamente faz uma travessia na sub-árvore de esquerda, seguido da travessia da sub-árvore da direita.

Segue-se então o seguinte padrão

raiz -> esquerda -> direita

```
#implementação da Travessia Pré-Ordem
# Raiz -> esquerda -> direita
def pre_ordem(self, no):
  if no != None:
    print(no.valor)
    self.pre_ordem(no.esquerda)
    self.pre_ordem(no.direita)
```



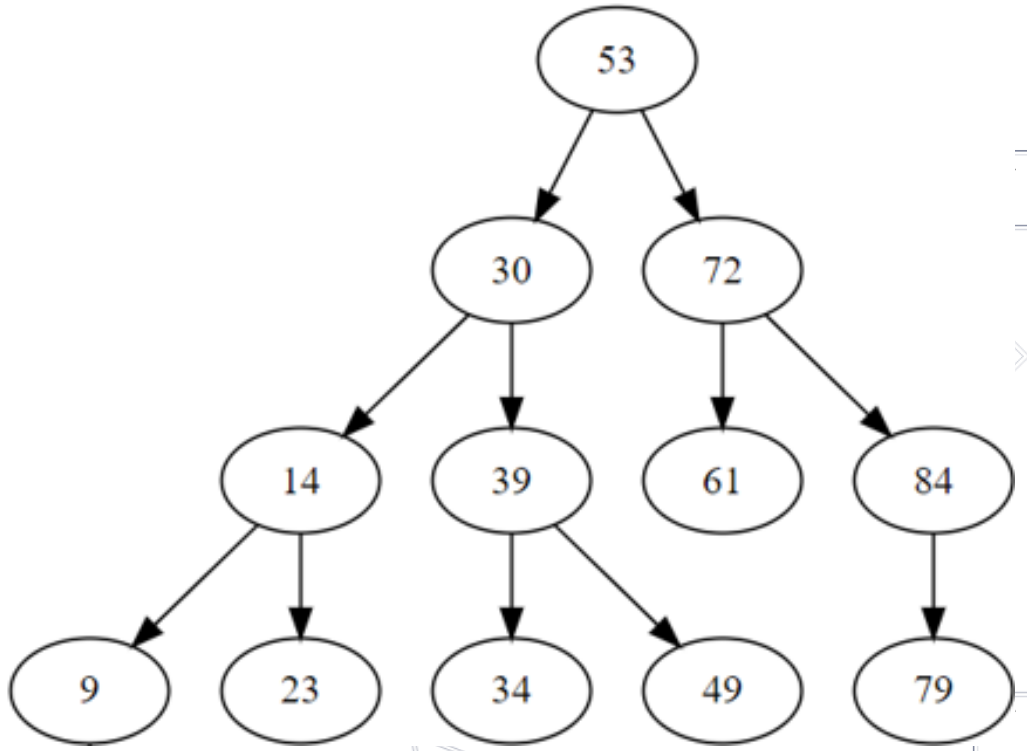
Travessia Pré-Ordem

```python
#implementação da Travessia Pré-Ordem
# Raiz -> esquerda -> direita
def pre_ordem(self, no):
  if no != None:
    print(no.valor)
    self.pre_ordem(no.esquerda)
    self.pre_ordem(no.direita)
```

Travessia Pré-Ordem

```python
arvore.pre_ordem(arvore.raiz)
```

```
53
30
14
9
2
23
39
34
49
72
61
84
79
```

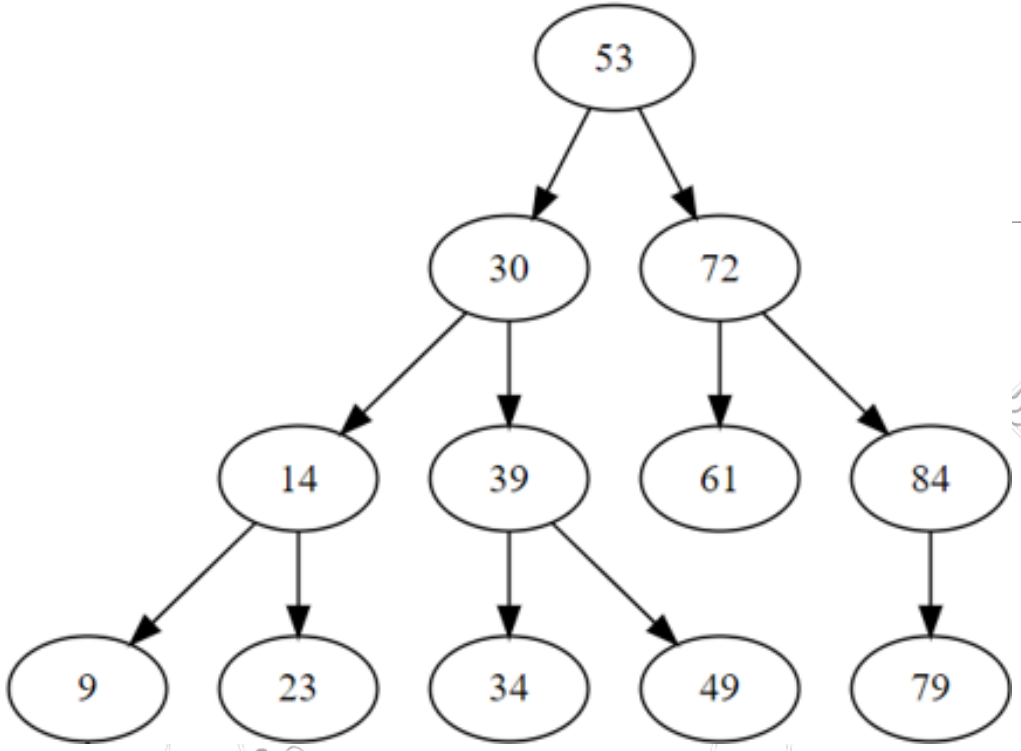


Travessia Em Ordem

Recursivamente faz a travessia na sub-árvore esquerda, visita a raiz e faz uma travessia recursiva na sub-árvore da direita

esquerda -> raiz -> direita

```
#implementação da Travessia Em Ordem
# esquerda -> Raiz -> direita
def em_ordem(self, no):
  if no != None:
    self.em_ordem(no.esquerda)
    print(no.valor)
    self.em_ordem(no.direita)
```



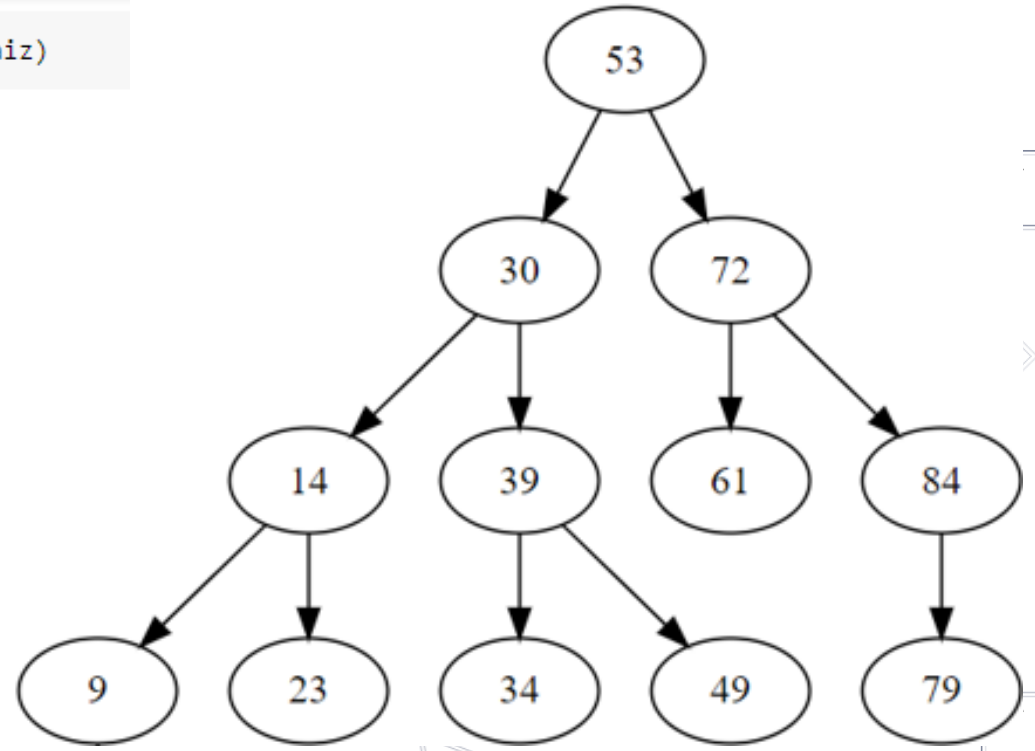
Travessia Em Ordem

```python
#implementação da Travessia Em Ordem
# esquerda -> Raiz -> direita
def em_ordem(self, no):
  if no != None:
    self.em_ordem(no.esquerda)
    print(no.valor)
    self.em_ordem(no.direita)
```

Travessia Em Ordem

```
[ ] arvore.em_ordem(arvore.raiz)

9
14
23
30
34
39
49
53
61
72
79
84
```

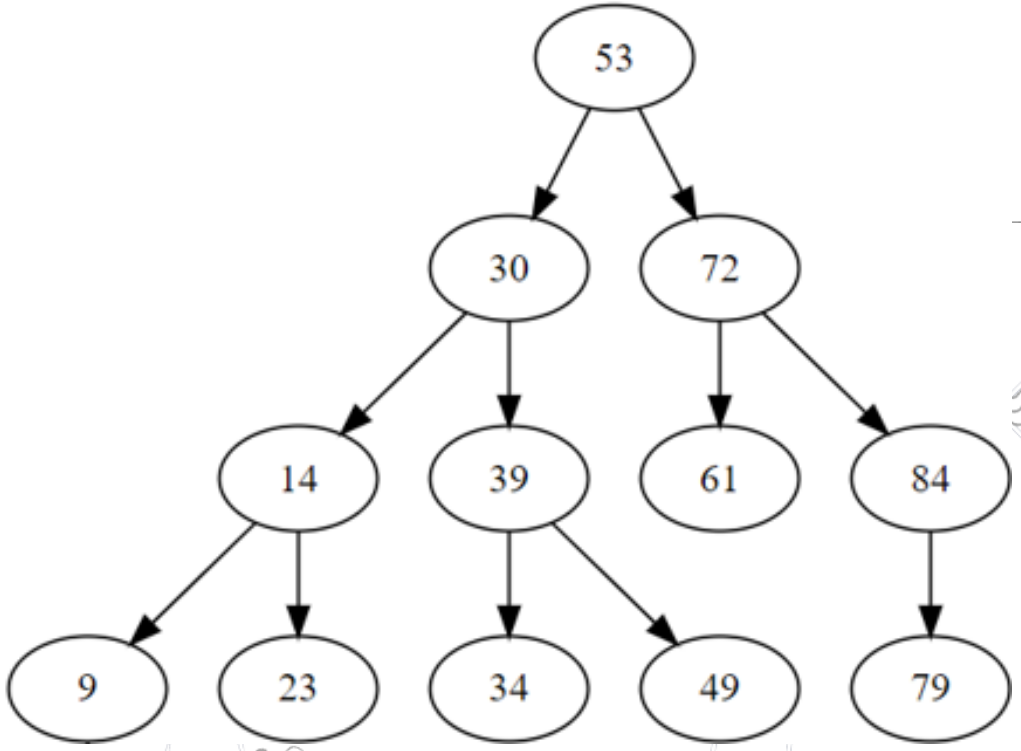


Travessia Pós-Ordem

Recursivamente faz a travessia na sub-árvore esquerda, faz uma travessia recursiva na sub-árvore da direita e por fim visita a raiz.

esquerda -> direita -> raiz

```
#implementação da Travessia Pós-Ordem
# esquerda -> direita -> Raiz
def pos_ordem(self, no):
  if no != None:
    self.pos_ordem(no.esquerda)
    self.pos_ordem(no.direita)
    print(no.valor)
```



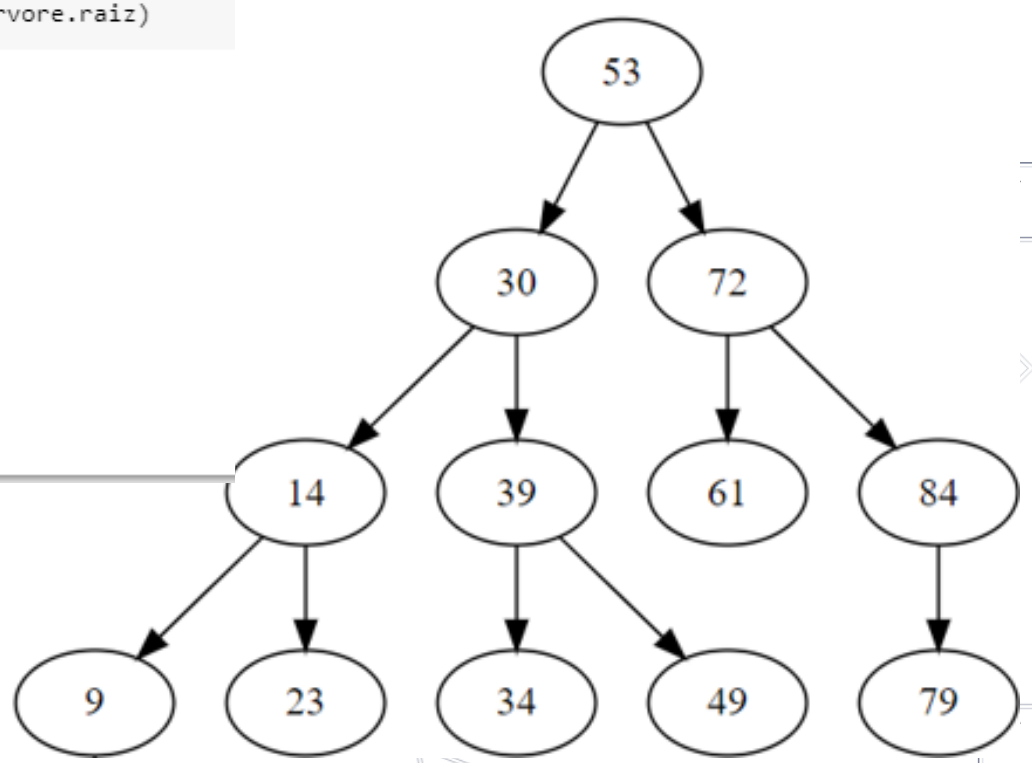
Travessia Pós-Ordem

```python
#implementação da Travessia Pós-Ordem
# esquerda -> direita -> Raiz
def pos_ordem(self, no):
  if no != None:
    self.pos_ordem(no.esquerda)
    self.pos_ordem(no.direita)
    print(no.valor)
```

Travessia Pós-Ordem

```python
arvore.pos_ordem(arvore.raiz)
```

```
9
23
14
34
49
39
30
61
79
84
72
53
```



UniEVANGÉLICA

UNIVERSIDADE EVANGÉLICA DE GOIÁS