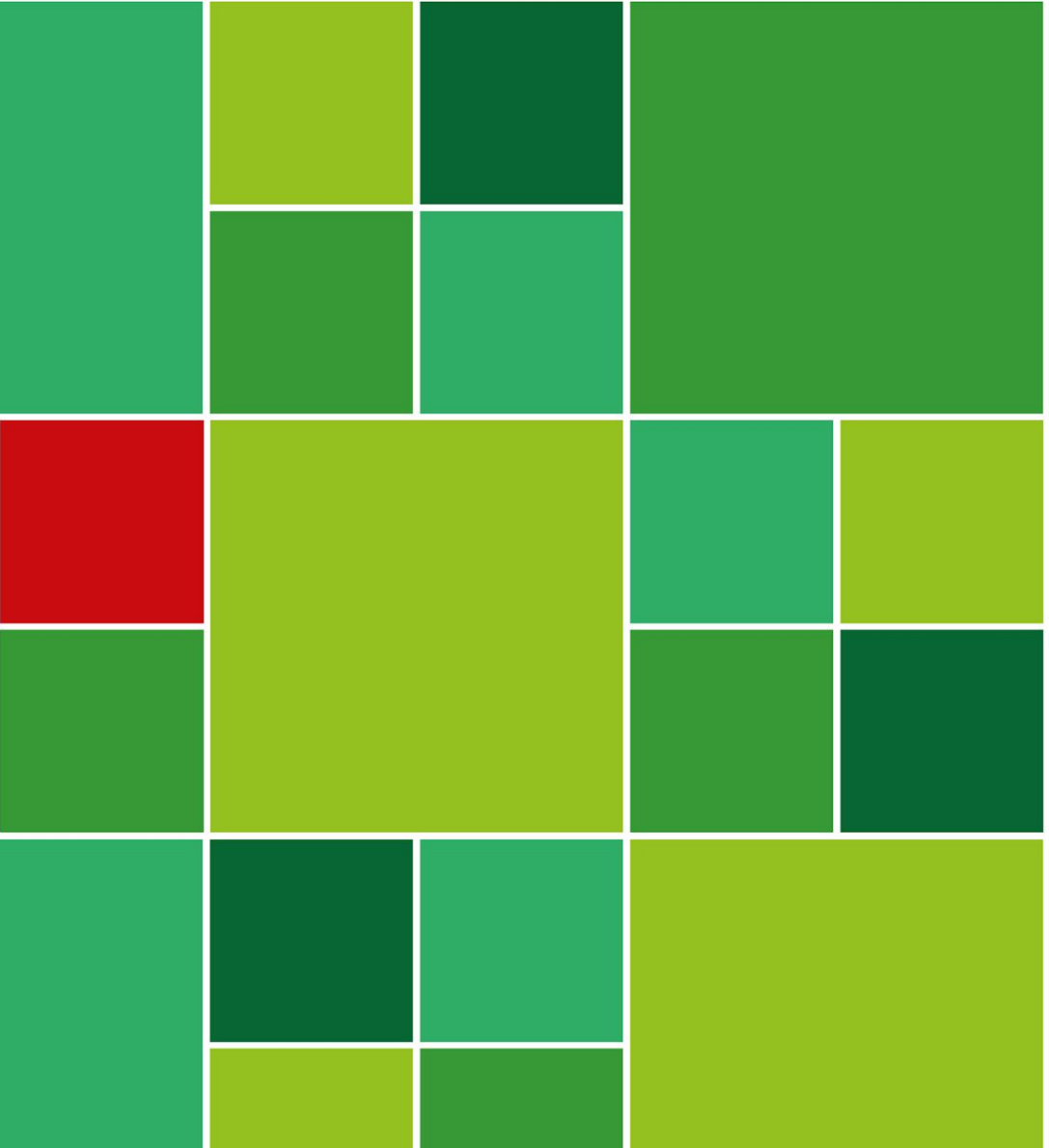


Estrutura de Dados

Árvores

Prof. Dr. Danilo Barbosa

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO

O que vamos ver nessa aula?

- Árvores
  - Árvores binárias
  - Árvores binárias de busca

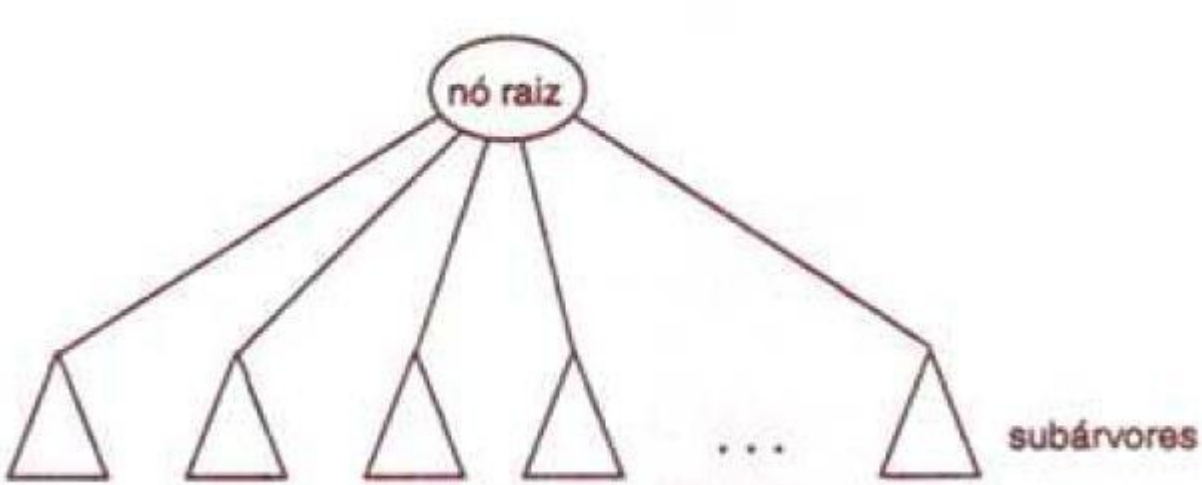
Árvores

- Até agora vimos estruturas que podem ser chamadas de lineares
  - Listas, tuplas, dicionários, listas encadeadas
- Elas são bastante importantes, mas elas não são adequadas para representar dados que devem ser dispostos de maneira hierárquica
  - Como arquivos armazenados em um computador
- Árvores são estruturas adequadas para representação de hierarquias
  - Armazenamento de dados em computador
  - Algoritmos de busca e ordenação
  - Expressões aritméticas e análise sintática

Árvores

- A forma mais natural de definir uma estrutura de árvore é usando a recursividade
  - Uma árvore é composta por um conjunto de nós
  - Existe um nó r, denominado raiz, que contém zero ou mais subárvores, cujas raízes são ligadas diretamente a r
  - Esses nós raízes da subárvores são ditos filhos do nó pai r
  - Nós com filhos são chamados de nós internos
  - Nós que não tem filhos são chamados de folhas ou nós externos
  - Nós são ligados por arcos

Árvores

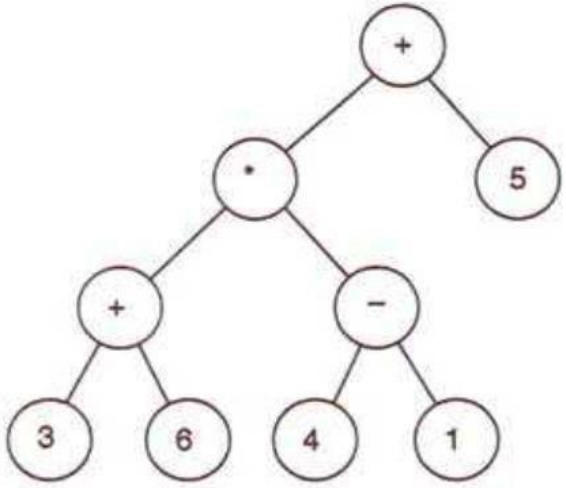


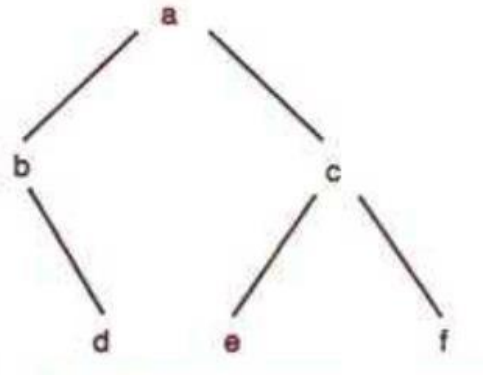
Árvores binárias

- Árvores binárias são árvores nas quais cada nó tem, no máximo, dois filhos

- De maneira recursiva, podemos definir uma árvore binária como sendo:
  - Uma árvore vazia; ou
  - Um nó raiz tendo duas subárvores, identificadas como subárvore da direita e subárvore da esquerda

Árvores binárias





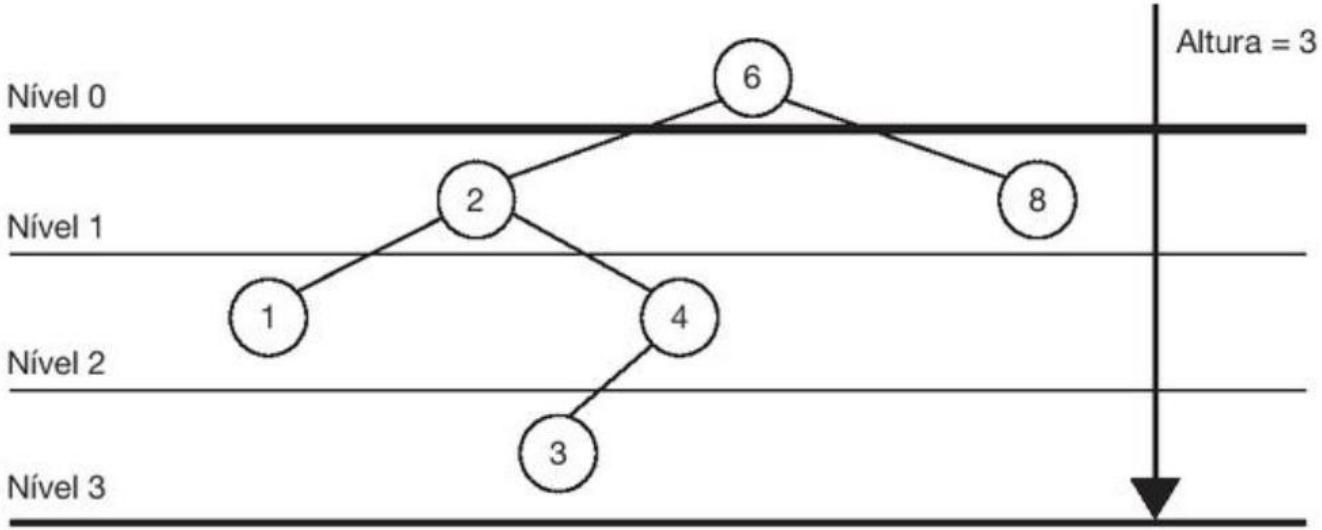
Árvores binárias

- Uma propriedade fundamental de todas as árvores é que só existe um caminho da raiz para qualquer nó

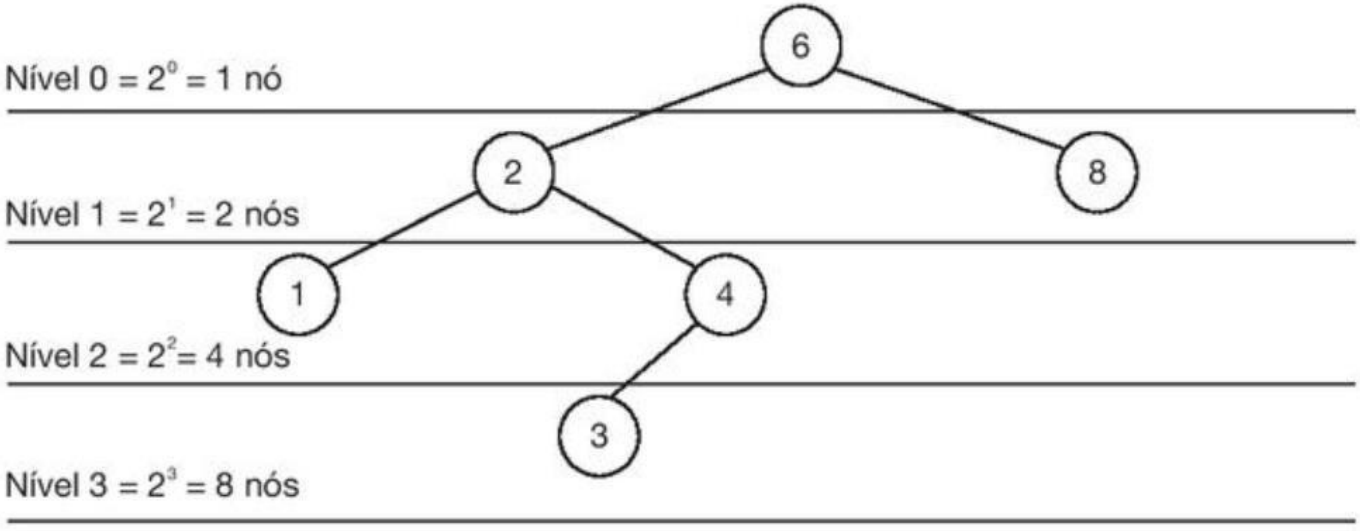
- Podemos definir a altura de uma árvore como sendo o comprimento mais longo da raiz até uma das folhas

- A altura dos exemplos anteriores
  - Árvore da esquerda tem altura 3
    - Árvore da direita tem altura 2

Árvores binárias

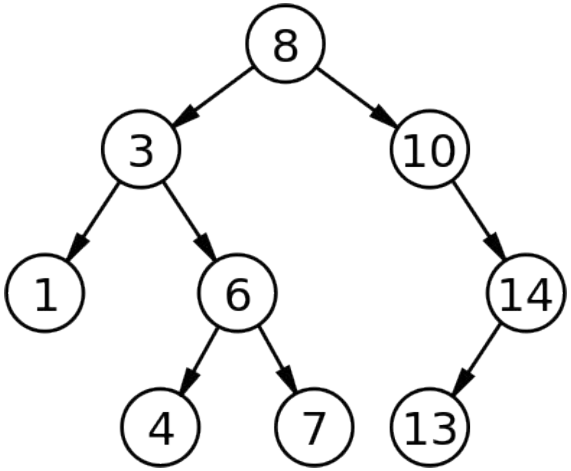


Árvores binárias



Árvores binárias de busca

- Árvore binária de busca é uma estrutura de dados de árvore binária, onde todos os nós da subárvore esquerda possuem um valor numérico inferior ao nó raiz e todos os nós da subárvore direita possuem um valor superior ao nó raiz



Árvores binárias de busca

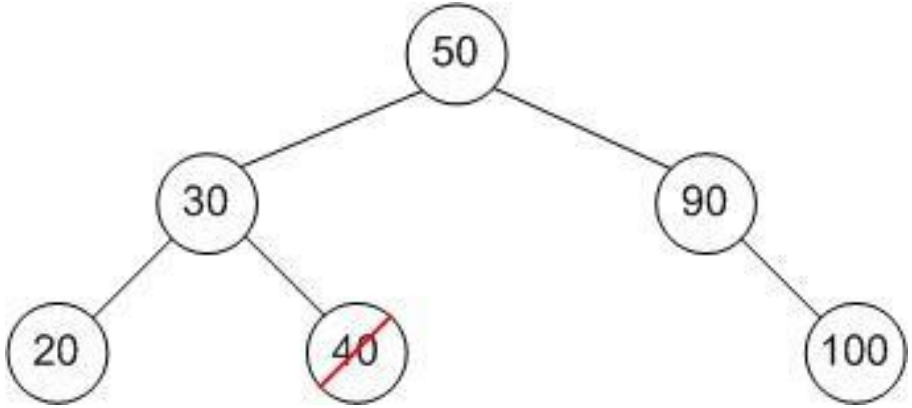
- Operação de **busca**
  - Começa examinando a raiz
    - Se a árvore está vazia, o valor procurado não pode existir na árvore
    - Se o valor é igual a raiz, a busca é encerrada
  - Se o valor for menor do que a raiz, a busca segue pela subárvore esquerda
  - Em caso contrário, a busca segue pela subárvore direita
  - Repete-se esse processo até o valor ser encontrado ou a subárvore ser nula
    - Se o valor não for encontrado até a busca chegar na subárvore nula, então o valor não está presente na árvore

Árvores binárias de busca

- Operação de **inserção**
  - Se o valor for menor do que a raiz, caminha-se pela subárvore esquerda
  - Em caso contrário, caminha-se pela subárvore direita
  - Repete-se esse processo até encontrar uma subárvore ser nula
    - Ao encontrar essa subárvore nula, insere o nó como uma folha

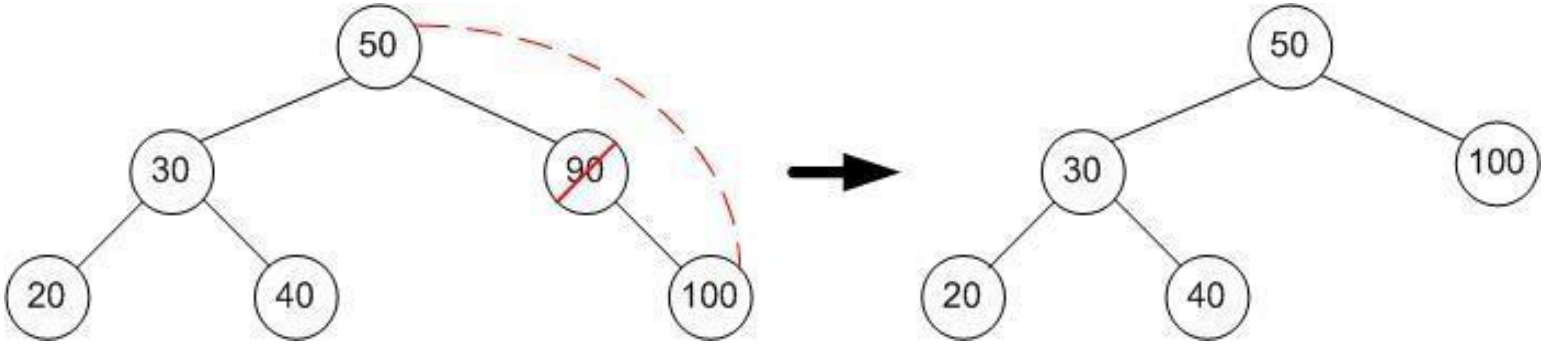
Árvores binárias de busca

- Operação de **remoção**
  - Tem 3 casos
    - Todos os casos, começam com uma busca pelo nó a ser removido
  - Remoção da folha
    - Basta remover a folha da árvore



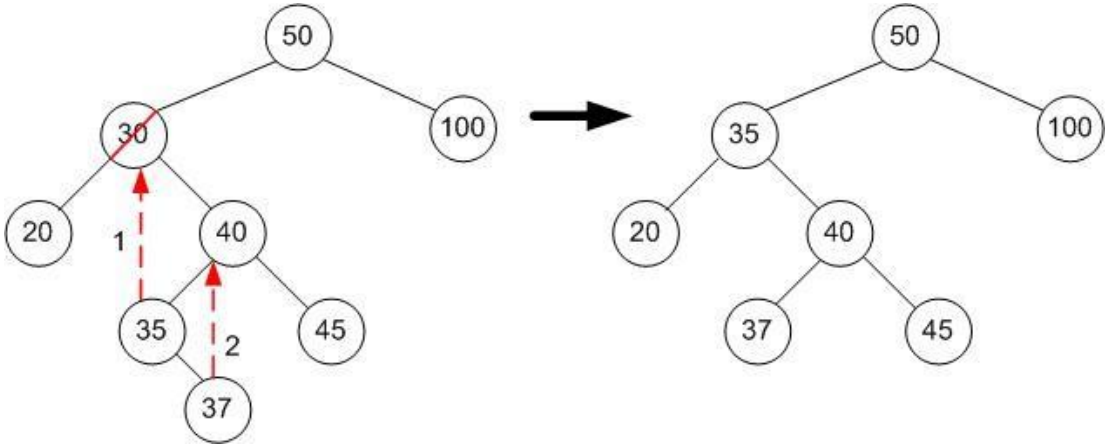
Árvores binárias de busca

- Operação de **remoção**
  - Remoção de nó com 1 filho
    - O filho ocupa a posição do pai



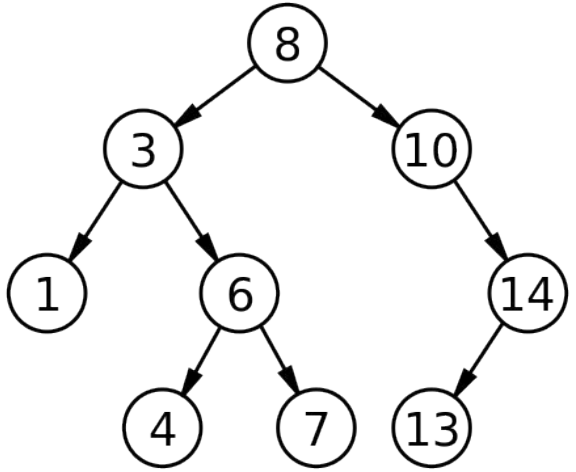
Árvores binárias de busca

- Operação de **remoção**
  - Remoção de nó com 2 filhos
    - Substitui-se o nó a ser removido pelo seu sucessor (nó mais a esquerda da subárvore a direita) ou seu antecessor (nó mais a direita da subárvore a esquerda)



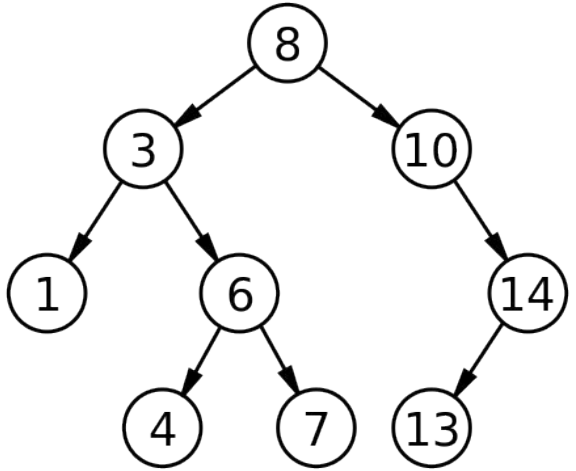
Árvores binárias de busca

- Percurso
  - Pré-ordem (ou profundidade):
    - Visita a raiz
    - Percorre a subárvore esquerda em pré-ordem
    - Percorre a subárvore direita em pré-ordem
  - Pré-ordem
    - 8, 3, 1, 6, 4, 7, 10, 14, 13



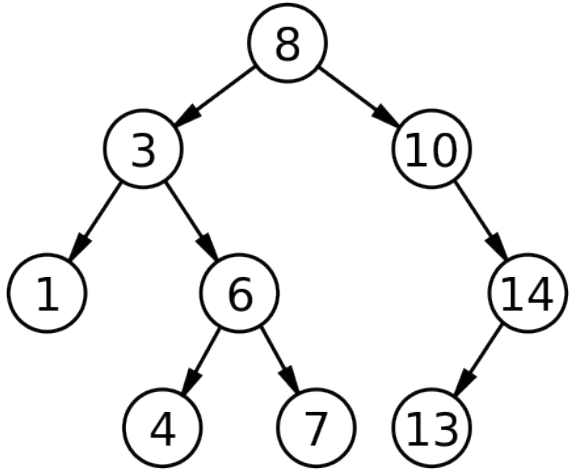
Árvores binárias de busca

- Percurso
  - Ordem Simétrica:
    - Percorre a subárvore esquerda em ordem simétrica
    - Visita a raiz
    - Percorre a subárvore direita em ordem simétrica
  - Ordem Simétrica
    - 1, 3, 4, ,6, 7, 8, 10, 13, 14



Árvores binárias de busca

- Percurso
  - Pós-ordem:
    - Percorre a subárvore esquerda em pós-ordem
    - Percorre a subárvore direita em pós-ordem
    - Visita a raiz
  - Pós-ordem
    - 1, 4, 7, 6, 3, 13, 14, 10, 8



Atividade

- Código das árvores binárias de busca

Atividade

Classes

```
class No:
    def __init__(self, key, dir, esq):
        self.item = key
        self.dir = dir
        self.esq = esq

class Tree:

    def __init__(self):
        self.root = No(None,None,None)
        self.root = None
```

Atividade

Inserir

```
def inserir(self, v):
    novo = No(v,None,None) # cria um novo Nó
    if self.root == None:
        self.root = novo
    else: # se nao for a raiz
        atual = self.root
        while True:
            anterior = atual
            if v < atual.item: # ir para esquerda
                atual = atual.esq
                if atual == None:
                    anterior.esq = novo
                    return
            # fim da condição ir a esquerda
            else: # ir para direita
                atual = atual.dir
                if atual == None:
                    anterior.dir = novo
                    return
            # fim da condição ir a direita
```

Atividade

Buscar

```
def buscar(self, chave):
    if self.root == None:
        return None # se arvore vazia
    atual = self.root # começa a procurar desde raiz
    while atual.item != chave: # enquanto nao encontrou
        if chave < atual.item:
            atual = atual.esq # caminha para esquerda
        else:
            atual = atual.dir # caminha para direita
        if atual == None:
            return None # encontrou uma folha -> sai
    return atual  # terminou o laço while e chegou aqui é po
```

Atividade

Remover

```
def remover(self, v):
    if self.root == None:
            return False # se arvore vazia
    atual = self.root
    pai = self.root
    filho_esq = True
    # ****** Buscando o valor **********
    while atual.item != v: # enquanto nao encontrou
            pai = atual
            if v < atual.item: # caminha para esquerda
                    atual = atual.esq
                    filho_esq = True # é filho a esquerda? sim
            else: # caminha para direita
                    atual = atual.dir
                    filho_esq = False # é filho a esquerda? NAO
            if atual == None:
                    return False # encontrou uma folha -> sai
     # fim laço while de busca do valor
    # ...
```

Atividade

Caminhar

- Pré ordem

```
def preOrder(self, atual):
    if atual != None:
        print(atual.item,end=" ")
        self.preOrder(atual.esq)
        self.preOrder(atual.dir)
```

Atividade

Caminhar

- Em ordem

```
def inOrder(self, atual):
    if atual != None:
        self.inOrder(atual.esq)
        print(atual.item,end=" ")
        self.inOrder(atual.dir)
```

Atividade

Caminhar

- Pós ordem

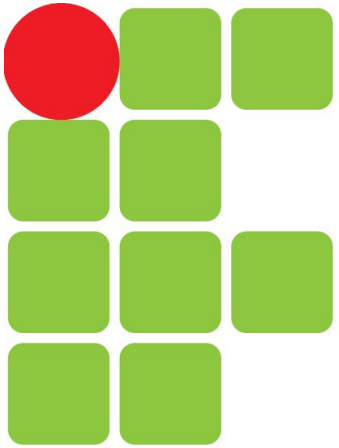
```
def posOrder(self, atual):
    if atual != None:
        self.posOrder(atual.esq)
        self.posOrder(atual.dir)
        print(atual.item,end=" ")
```

Atividade

- Tomando por base a implementação para a árvore binária fornecida, implemente métodosque retornem:
  a. O valor mínimo armazenado na árvore binária
  b. O valor máximo armazenado na árvore binária
  c. A média dos valores armazenados na árvore binária
  d. O antecessor de um nó a ser removido da árvore binária

Referência

- CELES FILHO, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução da Estrutura de Dados: com técnicas de programação em C. Rio de Janeiro: Elsevier, 2014.
- ASCENCIO, A. F. G. & ARAÚJO, G. S. Estrutura de Dados: algoritmos, análise de complexidade e implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010.



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO