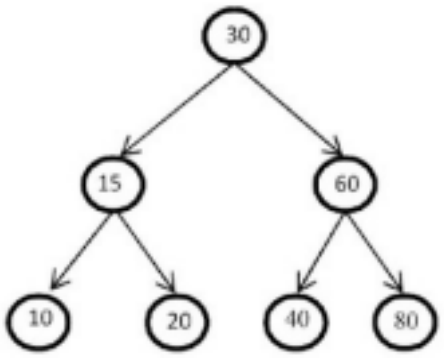
UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA - CCT
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – DCC
DCC405 – ESTRUTURA DE DADOS II
PROF. MSC. ACAUAN C. RIBEIRO
Aluno: GUILHERME LUCAS PEREIRA BERNARDO

EXERCÍCIO - Aula 05 – Árvores
Reconstrução e BST

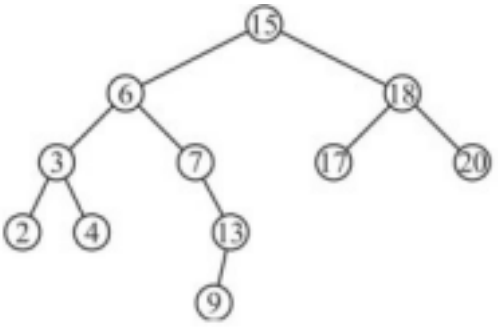


a seguir.

Assinale a alternativa que apresenta, corretamente, a sequência de inserção que gera essa ABB.

a) 30, 15, 40, 10, 20, 60, 80
b) 30, 15, 40, 10, 20, 80, 60
c) 30, 15, 60, 10, 20, 40, 80 ✅
d) 30, 60, 20, 80, 15, 10, 40
e) 30, 60, 40, 10, 20, 15, 80

2) Analise a árvore a seguir e responda as questões 2.1 e 2.2:



Thomas H. Cormen *et al.* Algoritmos: teoria e prática. Editora Campus, v. 2, 2002, p. 207.

2.1) Mostre os percursos

Pré-ordem: **15 6 3 2 4 7 13 9 18 17 20 None**

In-Ordem: **2 3 4 6 7 9 13 15 17 18 20 None**

Pós-ordem: **2 4 3 9 13 7 6 17 20 18 15 None**

2.2) De acordo com a figura anterior, o procedimento

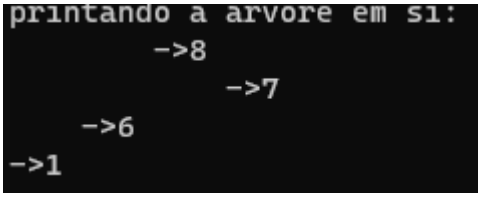
```
CONSULTA (x)
1  while esquerda [x] ≠ NIL
2     do x ← esquerda [x]
3  return x
```

realiza, na árvore, a consulta de:

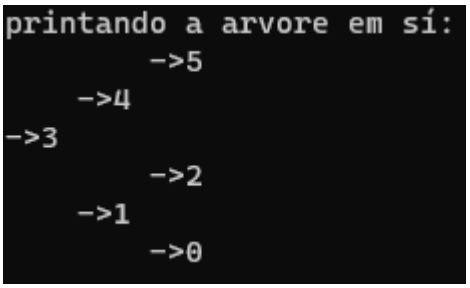
A) search **B)minimum**✅ C)maximum D)sucessor

3) Reconstrua graficamente as árvores a partir dos percursos **pré-ordem** e **in ordem** e em seguida apresente o percurso em **pós-ordem**.

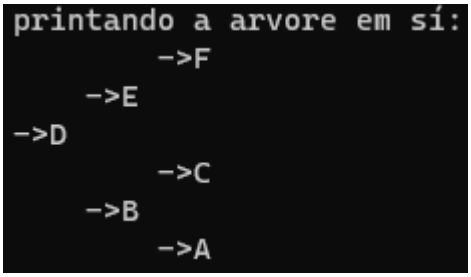
3.1)
Pré: 1 6 7 8
In: 1 6 8 7
Pós: 7 8 6 1



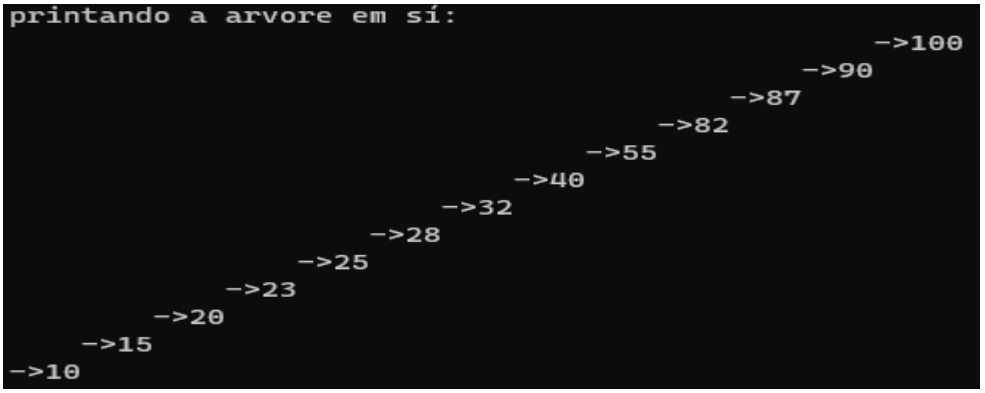
3.2)
Pré: 0 1 3 4 2 5
In: 3 1 4 0 5 2
Pós: 0 2 1 5 4 3



3.3)
Pré: A B D E C F
In: D B E A F C
Pós: A C B F E D



3.4)
Pré: 40 25 20 10 15 23 32 28 82 55 90 87 100
In: 10 15 20 23 25 28 32 40 55 82 87 90 100
Pós: 100 90 87 82 55 40 32 28 25 23 20 15 10



4) Tendo como base o algoritmo de árvore binária de busca visto em sala, implemente os seguintes métodos:

- altura() : retorna a altura tendo como base o nó passado como referencia ✅

- minimo() : retornar o menor valor da arvore ✅

- maximo() : retornar o maior valor da arvore ✅

- remocao() : implemente um método que remova um elemento passado como parâmetro ✅

R: import time

```
#->estrutura do nó<-
class Node:
    def __init__(self, value=None):
        self.value = value
        self.left = None
        self.right = None

def insertNode(root, value):
    if root is None:
        return Node(value)
    else:
        if value < root.value:
            root.left = insertNode(root.left, value)
        elif value > root.value:
            root.right = insertNode(root.right, value)
    return root


#->percursos de profundidade<-
def preOrdem(root):
    if root:
        print(root.value, end = " ")
```

```
        preOrdem(root.left)
        preOrdem(root.right)

def inOrdem(root):
    if root:
        inOrdem(root.left)
        print(root.value, end = " ")
        inOrdem(root.right)

def posOrdem(root):
    if root:
        posOrdem(root.left)
        posOrdem(root.right)
        print(root.value, end = " ")



#->printando a arvore de ladinho<-
def printTree(root, level = 0):
    if root is not None:
        printTree(root.right, level+1)
        print(' ' * 4*level + '->' + str(root.value))
        printTree(root.left, level+1)



#->achar o menor valor da arvore<-
def findMin(root):
    if root is None:
        return None
```

```
    while root.left != None:
        root = root.left
    return root



#->achar o maior valor da arvore<-
def findMax(root):
    if root is None:
        return None
    while root.right != None:
        root = root.right
    return root



#->achar a altura total da arvore<-
def findHeight(root):
  if root is None:
      return 1
  leftH = findHeight(root.left)
  rightH = findHeight(root.right)
  return max(leftH, rightH) + 1

#->remover 1 node da arvore<-
def deleteNode(root, value):
    if root is None: return root

    #se o valor a ser deletado for menor que root então ele está à esquerda de root
    elif (value < root.value):
```

```
        root.left = deleteNode(root.left, value)


    #se o valor a ser deletado for maior que root entao ele fica à direita de root
    elif(value > root.value):
        root.right = deleteNode(root.right, value)


    #se o valor a ser deletado é igual a root entao esse valor SERÁ deletado
    else:


        #código do prof
        #Caso1: Nó sem filhos(folha)
        if root.left and root.right is None:
            root = None
            return root


        #caso2: Nó com 1 filho
        elif root.left is None:
            temp = root
            root = root.right
            temp = None
            return root
        elif root.right is None:
            temp = root
            root = root.left
            temp = None
            return root
        else:
```

```
            # if root.left is None: #TENTATIVA 1 -FUNCIONA-
funciona pro caso 1 e 2 de uma vez só
            #     temp = root.right
            #     root = None
            #     return temp
            # elif root.right is None:
            #     temp = root.left
            #     root = None
            #     return temp

            #Caso3: Nó com 2 filhos
            #pegamos o sucessor em ordem crescente que é o menor
nó na subarvore direita
            temp = findMin(root.right)

            #daí pegamos o sucessor em ordem crescente e passamos
para o valor de root
            root.value = temp.value

            #e deletamos o sucessor em ordem
            root.right = deleteNode(root.right, temp.value)
    return root

#->percurso em level(BFS)<-
import queue
q = queue.Queue()
def levelOrder(root):
    if root is None: return None
```

```
    q.put(root)
    while not q.empty():
        current = q.queue[0]
        print(current.value, end = " ")
        if current.left is not None: q.put(current.left)
        if current.right is not None: q.put(current.right)
        q.get()



#->!implementando a arvore!<-
root = None
root = insertNode(root, 10)
root = insertNode(root, 15)
root = insertNode(root, 20)
root = insertNode(root, 23)
root = insertNode(root, 25)
root = insertNode(root, 28)
root = insertNode(root, 32)
root = insertNode(root, 40)
root = insertNode(root, 55)
root = insertNode(root, 82)
root = insertNode(root, 87)
root = insertNode(root, 90)
root = insertNode(root, 100)


#->!imprimindo tudo!<-
time.sleep(1)
print("printando a arvore em sí:")
```

```
time.sleep(1)

printTree(root)

time.sleep(1)

print("\nagora printando estatisticas básicas:")

time.sleep(1)

print("Min:     ", findMin(root).value)

print("Max:     ", findMax(root).value)

print("Altura:  ", findHeight(root),"\n")

time.sleep(1)

print("agora em ordem de nivel")

time.sleep(1)

print(levelOrder(root))

time.sleep(1)

print("\npor PreOrdem:")

time.sleep(1)

print(preOrdem(root))

time.sleep(1)

print("\npor InOrdem:")

time.sleep(1)

print(inOrdem(root))

time.sleep(1)

print("\npor PosOrdem:")

time.sleep(1)

print(posOrdem(root))


#parte de exclusão brabissima(testagem)

print("\n\n")

print("----------------------------------------")
```

```
print("\n\n")


time.sleep(1.5)
print("agora trabalhando com exclusão de nós!")
time.sleep(1)
print("vamos deletar o 40!")
time.sleep(1)
deleteNode(root, 40)
print("printando a arvore em sí:")
time.sleep(1)
printTree(root)
time.sleep(1)
print("\nagora printando estatisticas básicas:")
time.sleep(1)
print("Min:     ", findMin(root).value)
print("Max:     ", findMax(root).value)
print("Altura:  ", findHeight(root),"\n")
time.sleep(1)
print("agora em ordem de nivel")
time.sleep(1)
print(levelOrder(root))
time.sleep(1)
print("\npor PreOrdem:")
time.sleep(1)
print(preOrdem(root))
time.sleep(1)
print("\npor InOrdem:")
time.sleep(1)
```

```
print(inOrdem(root))

time.sleep(1)

print("\npor PosOrdem:")

time.sleep(1)

print(posOrdem(root))
```

print do console:

```
printando a arvore em sí:
                                                    ->100
                                                ->90
                                            ->87
                                        ->82
                                    ->55
                                ->40
                            ->32
                        ->28
                    ->25
                ->23
            ->20
        ->15
->10

agora printando estatisticas básicas:
Min:       10
Max:       100
Altura:    14

agora em ordem de nivel
10 15 20 23 25 28 32 40 55 82 87 90 100 None

por PreOrdem:
10 15 20 23 25 28 32 40 55 82 87 90 100 None

por InOrdem:
10 15 20 23 25 28 32 40 55 82 87 90 100 None

por PosOrdem:
100 90 87 82 55 40 32 28 25 23 20 15 10 None
```

```
------------------------------------



agora trabalhando com exclusão de nós!
vamos deletar o 40!
printando a arvore em sí:
                                                    ->100
                                                ->90
                                            ->87
                                        ->82
                                    ->55
                                ->32
                            ->28
                        ->25
                    ->23
                ->20
            ->15
        ->10

agora printando estatisticas básicas:
Min:      10
Max:      100
Altura:   13
```

```
agora em ordem de nivel
10 15 20 23 25 28 32 55 82 87 90 100 None

por PreOrdem:
10 15 20 23 25 28 32 55 82 87 90 100 None

por InOrdem:
10 15 20 23 25 28 32 55 82 87 90 100 None

por PosOrdem:
100 90 87 82 55 32 28 25 23 20 15 10 None
```