Avl

October 3, 2023

1 AVL. Insertion and Deletion Algorithm

[ ]:
```
class Node:
    def __init__(self, data, height):
        self.left   = None
        self.right  = None
        self.data   = data
        self.height = height
```

[ ]:
```
def height(root):
    if root == None:
        return 0
    else:
        return root.height
```

[ ]:
```
def newNode(key):
    node = Node(key,1)
    return node
```

[ ]:
```
def getBalance(root):
    if root == None:
        return 0
    return height(root.left)-height(root.right)
```

[ ]:
```
def minValueNode(root):
    current = root
    while(current.left != None):
        current = current.left
    return current
```

[ ]:
```
# x is a node
def leftRotate(x):
    y        = x.right
    T2       = y.left
    y.left   = x
    x.right  = T2
    x.height = max(height(x.left),height(x.right))+1
```

```
    y.height = max(height(y.left),height(y.right))+1

    return y
```

[ ]:
```
# y is a node
def rightRotate(y):
    x          = y.left
    T2         = x.right
    x.right    = y
    y.left     = T2

    y.height   = max(height(y.left), height(y.right))+1
    x.heigh    = max(height(x.left), height(x.right))+1

    return x
```

[ ]:
```
def insertNode(root, key):

    if root == None:
        return newNode(key)

    if key <= root.data:
        root.left = insertNode(root.left, key)
    #elif key > root.data:
    else:
        root.right = insertNode(root.right, key)
    #else:
        #return root

    # Update the balance factor of each node balance tree

    root.height = 1 + max(height(root.left),height(root.right))
    balance     = getBalance(root)

    # Rotations

    if balance > 1 and key < root.left.data:
        return rightRotate(root) # RR

    if balance < -1 and key > root.right.data:
        return leftRotate(root) # LL

    if balance > 1 and key > root.left.data: # LR
        root.left = leftRotate(root.left)
        return rightRotate(root)

    if balance < -1 and key < root.right.data: # RL
```

```
        root.right = rightRotate(root.right)
        return leftRotate(root)

    return root
```

[ ]:
```
def bst_to_latex_pstricks(root):

    latex_code = "\\documentclass{article}\n"
    latex_code += "\\usepackage{pstricks,pst-tree,pst-node}\n"
    latex_code += "\\begin{document}\n"
    latex_code += "\\begin{center}\n"
    latex_code += f"\\pstree[levelsep=1,nodesep=3pt]{{\n"
    def traverse(node):
        nonlocal latex_code
        if node:
            latex_code += f"\\Tr[name={node.data}]{{{node.data}}}}}{{\n"

            if node.left:
                latex_code += f"\\pstree{{ "
                traverse(node.left)
                latex_code += f"}}"
            else:
                latex_code += f"\\Tr[name={None}]{{N}}\n"

            if node.right:
                latex_code += f"\\pstree{{ "
                traverse(node.right)
                latex_code += f"}}"
            else:
                latex_code += f"\\Tr[name={None}]{{N}}\n"

    traverse(root)
    latex_code += f"}}\n"
    latex_code += f"\\end{{center}} \n"
    latex_code += f"\\end{{document}} \n"

    return latex_code
```

[ ]:
```
Lista = [9, 27, 5, 14, 38, 12, 3, 19, 42, 8, 31, 16, 7, 22, 36]
```

[ ]:
```
Bst = newNode(Lista.pop(0))
for l in Lista:

    Bst = insertNode(Bst, l)
```