TREES:

1. ROOTS
2. LEAF
3. BRANCH
4. NODE

TYPES OF TREE

1. BINARY TREE(HAS ONLY TWO CHILD NODES)
2. N-ARY TREE(ALSO KNOWN AS TRIE)(CAN HAVE N NUMBER OF CHILD NODES.)
3. BINARY TREE:

* TWO CHILD NODES (IF ANY OF THESE TWO IS MISSING THEN IT WILL BE CALLED AS INCOMPLETE BINARY TREE.)

(i)LEFT (ii)RIGHT

1.1 NORMAL BINAY TREE

1.2 BINARY SEARCH TREE

\* AVL TREE(UPDATED CONCEPT )

LEVEL AND HEIGHT(DEPTH):

(-1,0,1)

DFS AND BFS

TRAVERSE: (i) INORDER(lRr)

(ii) PREODER(Rlr)

(iii)POSTORDER(lrR)

NODE

|  |  |  |
| --- | --- | --- |
| LEFTCHILD | DATA | RIGHTCHILD |

CODE:

class node:

def \_\_init\_\_(self,data):

self.value=data

self.left=None

self.right=None

def preorder(root):

if root == None:

return

print(root.value,end= ' ')

preorder(root.left)

preorder(root.right)

def inorder(root):

if root==None:

return

inorder(root.left)

print(root.value,end=' ')

inorder(root.right)

def postorder(root):

if root==None:

return

postorder(root.left)

postorder(root.right)

print(root.value,end=' ')

if \_\_name\_\_ =="\_\_main\_\_":

root = node(1)

root.left=node(2)

root.right=node(3)

root.left.left=node(4)

root.left.right=node(5)

root.right.left=node(6)

root.right.right=node(7)

print('preorder :')

preorder(root)

print('\ninorder :')

inorder(root)

print('\npostorder :')

postorder(root)

OUTPUT:

preorder :

1 2 4 5 3 6 7

inorder :

4 2 5 1 6 3 7

postorder :

4 5 2 6 7 3 1

LEVEL ORDER TRAVERSER

EXPECTED OUTPUT:

1

23

4568

9 10 11

12 13

1.Q=1,N

2.LOOP UNTIL Q!EMPTY

3.C=Q.POP

4.IF NONE | ELIF

IF Q EMPTY

1. PRINT ROOT VALUE

ADD L AND R TO QUEUE

Code:

class node:

def \_\_init\_\_(self,data):

self.value=data

self.left=None

self.right=None

def Level\_order(root):

Q=[root]

Q.append(None)

while len(Q)>0:

cur=Q.pop(0)

if cur==None:

if len(Q)==0:

break

else:

print()

Q.append(None)

else:

print(cur.value, end=' ')

if cur.left!=None:

Q.append(cur.left)

if cur.right!=None:

Q.append(cur.right)

if \_\_name\_\_ == "\_\_main\_\_":

root=node(1)

root.left=node(2)

root.right=node(3)

root.left.left=node(4)

root.left.right=node(5)

root.right.left=node(6)

root.right.right=node(7)

root.left.right.left=node(8)

root.left.right.right=node(10)

root.right.right.right=node(11)

root.left.right.left.left=node(12)

root.left.right.left.right=node(13)

Level\_order(root)

Output:

1

2 3

4 5 6 7

9 10 11

12 13

Q2) for a given tree (i) print only leaf nodes

(ii) print height of the tree (iii) level order

1

/ \

2 5

/ \ \

3 4 6

/ / \

9 7 8

\ \

10 11

/ /

14 12

\

13

Code:

class node:

def \_\_init\_\_(self,data):

self.value=data

self.left=None

self.right=None

def Level\_order(root):

Q=[root]

Q.append(None)

while len(Q)>0:

cur=Q.pop(0)

if cur==None:

if len(Q)==0:

break

else:

print()

Q.append(None)

else:

print(cur.value, end=' ')

if cur.left!=None:

Q.append(cur.left)

if cur.right!=None:

Q.append(cur.right)

def Height(root):

if root==None:

return 0

LH=Height(root.left)

RH=Height(root.right)

H= max(LH,RH)+1

return H

def Leaf\_node(root):

if root==None:

return 0

if root.left is None and root.right is None:

print(root.value,end=' ')

Leaf\_node(root.left)

Leaf\_node(root.right)

if \_\_name\_\_ == "\_\_main\_\_":

root=node(1)

root.left=node(2)

root.right=node(5)

root.left.left=node(3)

root.left.right=node(4)

root.right.right=node(6)

root.left.right.left=node(9)

root.left.right.left.right=node(10)

root.left.right.left.right.left=node(14)

root.right.right.left=node(7)

root.right.right.right=node(8)

root.right.right.left.right=node(11)

root.right.right.left.right.left=node(12)

root.right.right.left.right.left.right=node(13)

Level\_order(root)

H=Height(root)

print('\nHeight of tree:',H)

print('leaf nodes:')

Leaf\_node(root)