mad-pract-2-binary-tree-bst

June 28, 2024

[1]:
```
# 1. Install binarytree package
!pip install binarytree
```

```
Collecting binarytree
  Downloading binarytree-6.5.1-py3-none-any.whl (18 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-
packages (from binarytree) (0.20.3)
Requirement already satisfied: setuptools>=60.8.2 in
/usr/local/lib/python3.10/dist-packages (from binarytree) (67.7.2)
Collecting setuptools-scm[toml]>=5.0.1 (from binarytree)
  Downloading setuptools_scm-8.1.0-py3-none-any.whl (43 kB)
                             43.7/43.7 kB
1.9 MB/s eta 0:00:00
Requirement already satisfied: packaging>=20 in
/usr/local/lib/python3.10/dist-packages (from setuptools-
scm[toml]>=5.0.1->binarytree) (24.1)
Requirement already satisfied: tomli>=1 in /usr/local/lib/python3.10/dist-
packages (from setuptools-scm[toml]>=5.0.1->binarytree) (2.0.1)
Installing collected packages: setuptools-scm, binarytree
Successfully installed binarytree-6.5.1 setuptools-scm-8.1.0
```

[2]:
```
# 2. Creating a Node
from binarytree import Node
root = Node(3)
root.left = Node(6)
root.right = Node(8)
print("The binary tree created is: ", root)
```

```
The binary tree created is:
  3
 / \
6   8
```

[3]:
```
# 3. ALTERNATE WAY OF CREATING NODES IN A BINARY TREE
from binarytree import Node
root = Node(1) #create a root node
root.left = Node(2)  #create left child
```

```
root.right = Node(3)  #create right child
print("The binary tree created is: ", root)
```

```
The binary tree created is:
  1
 / \
2   3
```

[4]:
```
# 4. Getting binary tree
# Getting list of nodes
print('List of nodes in the order root,left,right :', list(root))
# Getting inorder of nodes
print('Inorder of nodes :', root.inorder)
# Getting preorder of nodes
print('Preorder of nodes :', root.preorder)
# Getting inorder of nodes
print('Postorder of nodes :', root.postorder)
# Checking tree properties
print('Size of tree :', root.size)
print('Height of tree :', root.height)
# Get all properties at once
print('Properties of tree : \n', root.properties)
```

```
List of nodes in the order root,left,right : [Node(1), Node(2), Node(3)]
Inorder of nodes : [Node(2), Node(1), Node(3)]
Preorder of nodes : [Node(1), Node(2), Node(3)]
Postorder of nodes : [Node(2), Node(3), Node(1)]
Size of tree : 3
Height of tree : 1
Properties of tree :
 {'height': 1, 'size': 3, 'is_max_heap': False, 'is_min_heap': True,
'is_perfect': True, 'is_strict': True, 'is_complete': True, 'leaf_count': 2,
'min_node_value': 1, 'max_node_value': 3, 'min_leaf_depth': 1, 'max_leaf_depth':
1, 'is_balanced': True, 'is_bst': False, 'is_symmetric': False}
```

[5]:
```
# 5. Creating/ building binary tree from given list
from binarytree import build
# List of nodes
nodes =[3, 6, 8, 2, 11, 4, 13]
# Building the binary tree
binary_tree = build(nodes)
print('Binary tree from list :\n',binary_tree)
# Getting list of nodes from binarytree
print('\nList from binary tree :', binary_tree.values)
```

```
Binary tree from list :
```

```
    ___3__
   /      \
  6        8
 / \      / \
2   11   4   13
```

```
List from binary tree : [3, 6, 8, 2, 11, 4, 13]
```

[6]:
```
# 6A. Creating/ building binary tree from given list
from binarytree import build
# List of nodes
nodes =['a', 'b', 'c', 'd', 'e', 'f', 'g']
# Building the binary tree
binary_tree = build(nodes)
print('Binary tree from list :\n',binary_tree)
# Getting list of nodes from binarytree
print('\nList from binary tree :', binary_tree.values)
```

```
Binary tree from list :
```

```
    __a__
   /     \
  b       c
 / \     / \
d   e   f   g
```

```
List from binary tree : ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

[7]:
```
# 6B. ALTERNATE WAY OF BUILDING A TREE
from binarytree import build
values=[1,2,3,4,5,6,7]
tree = build(values)
print(tree)
```

```
    __1__
   /     \
  2       3
 / \     / \
4   5   6   7
```

[8]:
```
# 6C. ALTERNATE WAY OF BUILDING A TREE
from binarytree import build
values=['v1','v2','v3','v4','v5','v6','v7']
tree = build(values)
```

```
print(tree)
```

```
     ____v1___
    /         \
  _v2         _v3
 /   \       /   \
v4    v5    v6    v7
```

[9]:
```
# 7. Create a random binary tree of any height
from binarytree import tree
root = tree()
print("Binary tree of any height :")
print(root)
# Create a random binary tree of given height
root2 = tree(height = 2)
print("Binary tree of given height :")
print(root2)
# Create a random perfect binary tree of given height
root3 = tree(height = 2, is_perfect = True)
print("Perfect binary tree of given height :")
print(root3)
# Create a random perfect binary tree of given height
root4 = tree(height = 3, is_perfect = True)
print("Perfect binary tree of given height :")
print(root4)
```

```
Binary tree of any height :

       _____1____
      /          \
  ___11__         14___
 /       \       /     \
9         3     7      _13
 \       /     /      /   \
  10    4     6      12    2

Binary tree of given height :

3__
   \
    6
   / \
  5   1

Perfect binary tree of given height :
```

```
    __6__
   /     \
  1       0
 / \     / \
5   4   3   2
```

```
Perfect binary tree of given height :
```

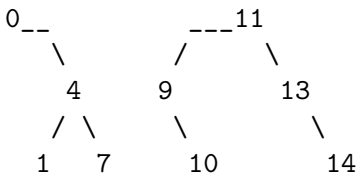
```
         ________3________
        /                 \
    __8___              ___12__
   /      \            /       \
  1       _10        _0         4
 / \     /   \      /  \       / \
5   6   11    7    14   13    9   2
```

0.1 Binary Search Tree

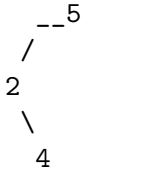
[10]:
```
# 8. Create a random BST of any height
from binarytree import bst
root = bst()
print(type(root))
print('Node list',root.values)
print('BST of any height : \n', root)
# Create a random BST of  given height
root2 = bst(height = 2)
print('Node list',root2.values)
print('BST of given height : \n', root2)
# Create a random perfect BST of given height
root3 = bst(height = 2, is_perfect = True)
print('Node list',root3.values)
print('Perfect BST of given height : \n', root3)
# Create a random perfect BST of given height
root4 = bst(height = 3)
print('Node list',root4.values)
print('Perfect BST of given height : \n', root4)
# Create a random perfect BST of given height
root5 = bst(height = 4, is_perfect=True)
print('Node list',root5.values)
print('Perfect BST of given height : \n', root5)
```

```
<class 'binarytree.Node'>
Node list [8, 0, 11, None, 4, 9, 13, None, None, 1, 7, None, 10, None, 14]
BST of any height :
```

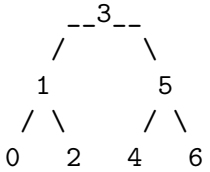
```
  ______8_____
 /            \
```



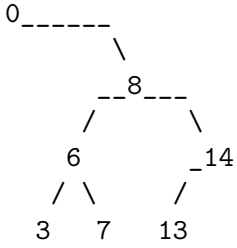
Node list [5, 2, None, None, 4]
BST of given height :



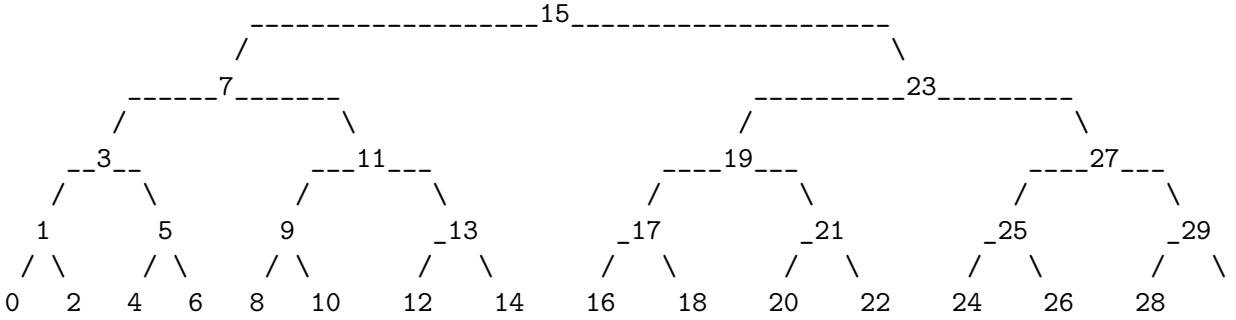
Node list [3, 1, 5, 0, 2, 4, 6]
Perfect BST of given height :



Node list [0, None, 8, None, None, 6, 14, None, None, None, None, 3, 7, 13]
Perfect BST of given height :



Node list [15, 7, 23, 3, 11, 19, 27, 1, 5, 9, 13, 17, 21, 25, 29, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
Perfect BST of given height :



30

0.2 Searching in BST

[11]:
```
# 9. Searching an element "key" in the created random BST of any height
from binarytree import bst
# Create a random BST of  given height which may be perfect or non perfect
root6 = bst(height = 3, is_perfect=True)
print('Node list',root6.values)
print('BST of given height : \n', root6)
try:
  print('value is found at posiiton =', root6.values.index(8))
except ValueError:
  print("value not found")
```

```
Node list [7, 3, 11, 1, 5, 9, 13, 0, 2, 4, 6, 8, 10, 12, 14]
BST of given height :

          ______7_______
         /              \
      __3__            ___11___
     /     \          /        \
    1       5        9         _13
   / \     / \      / \       /   \
  0   2   4   6    8   10    12    14

value is found at posiiton = 11
```

0.3 Insertion in BST

[12]:
```
# 10. Inserting an element in a random BST (perfect or non perfect) of given␣
 ↪height
# Function to inser the node in BST
def insert_bst(root, val):
  if root is None:
    return Node(val)
  if val < root.value:
    root.left = insert_bst(root.left, val)
  else:
    root.right = insert_bst(root.right, val)
  return root
# Create a perfect BST of given height using bst() function
root = bst(height=3,is_perfect=True)
print('BST before insertion :',root)
# Insert values to the left into the BST
insert_values1 = [2, 1, 3, 4]
for value in insert_values1:
```

```
  root = insert_bst(root, value)
print("BST after inserting values to the left:")
print(root)
# Insert values to the right into the BST
insert_values2 = [28, 32, 44, 75]
for value in insert_values2:
  root = insert_bst(root, value)
print("BST after inserting values to the right:")
print(root)
```

```
BST before insertion :
        ______7_______
       /              \
    __3__           ___11___
   /     \         /        \
  1       5       9         _13
 / \     / \     / \       /   \
0   2   4   6   8   10    12    14

BST after inserting values to the left:

            __________7_______
           /                  \
    ______3______           ___11___
   /             \         /        \
  1__           __5       9         _13
 /   \         /   \     / \       /   \
0     2       4     6   8   10    12    14
     / \     / \
    1   2   3   4

BST after inserting values to the right:

            __________7_______
           /                  \
    ______3______           ___11___
   /             \         /        \
  1__           __5       9         _13
 /   \         /   \     / \       /   \
0     2       4     6   8   10    12    14
     / \     / \                       \
    1   2   3   4                       28
                                          \
                                           32
                                             \
                                              44
                                                \
```

[13]:
```
# 11. Take an array and insert the node one by one to generate the BST and then␣
 ↪delete the given node from a BST
from binarytree import Node
# Function to insert the node into the BST
def insert_node(root, value):
  if root is None:
    return Node(value)
  if value < root.value:
    root.left = insert_node(root.left, value)
  else:
    root.right = insert_node(root.right, value)
  return root
# Take the array of elelements
values = [15,5,16,3,12,20,10,13,18,23,6,7]
bst = None
print('BST original tree before deletion')
for value in values:
  bst = insert_node(bst, value)
print(values)
print(bst)
# Function to update the tree after deleting the node
def delete_item_return_tree(array_new,value,i):
  array_new = [x for x in array_new if x != value]
  print(array_new)
  bst_new = None
  for value in array_new:
    bst_new = insert_node(bst_new,value)
  print(bst_new)
  print(i)
  return array_new
# deleting the item 1 from the array and updating the BST
node1 = 16
print('BST after deleting node1 = ',node1)
# values = delete_item_return_tree(values,node1,1)
# deleting the item 2 from the array and updating the BST
node2 = 5
print('BST after deleting node2 = ',node2)
values = delete_item_return_tree(values,node2,2)
#print(values)
```

```
BST original tree before deletion
[15, 5, 16, 3, 12, 20, 10, 13, 18, 23, 6, 7]

    _____________15
   /               \
```

```
  5_______            16___
 /        \               \
3          _12             _20
          /   \           /   \
       __10    13       18    23
      /
     6
      \
       7

BST after deleting node1 =  16
BST after deleting node2 =  5
[15, 16, 3, 12, 20, 10, 13, 18, 23, 6, 7]

   ______________15
  /                \
 3_______            16___
         \               \
          _12             _20
         /   \           /   \
      __10    13       18    23
     /
    6
     \
      7

2
```

[ ]: