Python 3 Binary Tree Implementation

Eddie Guo

September 2019

1 Introduction to Binary Trees

1.1 Topics Covered

(i) Recursive representation

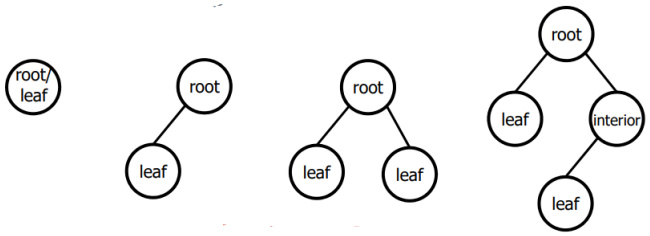
(ii) Tree leaf class

(iii) Tree branch class

(iv) Binary tree traversal

1.2 Defining a Binary Tree

- Binary tree made up of 1/more nodes
- Each node has 0/1/2 children
- All nodes have 1 parent, except root node (which has no parent)
- All leaf nodes store a value (byte to be compressed)
- All interior nodes are root node of subtree



2 How to Implement Binary Trees in Python?

- Need way to rep tree nodes & rels btw them
- Option 1: lists of lists
  - Each subtree is list that contains root, left subtree, right subtree
  - Gets complicated quickly; hard to keep track of nested subtrees
- Option 2: custom classes
  - Tree branch class capable of containing left & right subtree
  - Tree leaf class to rep indiv leaf nodes

2.1 Tree Leaf Class

- Tree leaf properties: has value (uncompressed byte) that it's storing
- Tree leaf behavs: n/a

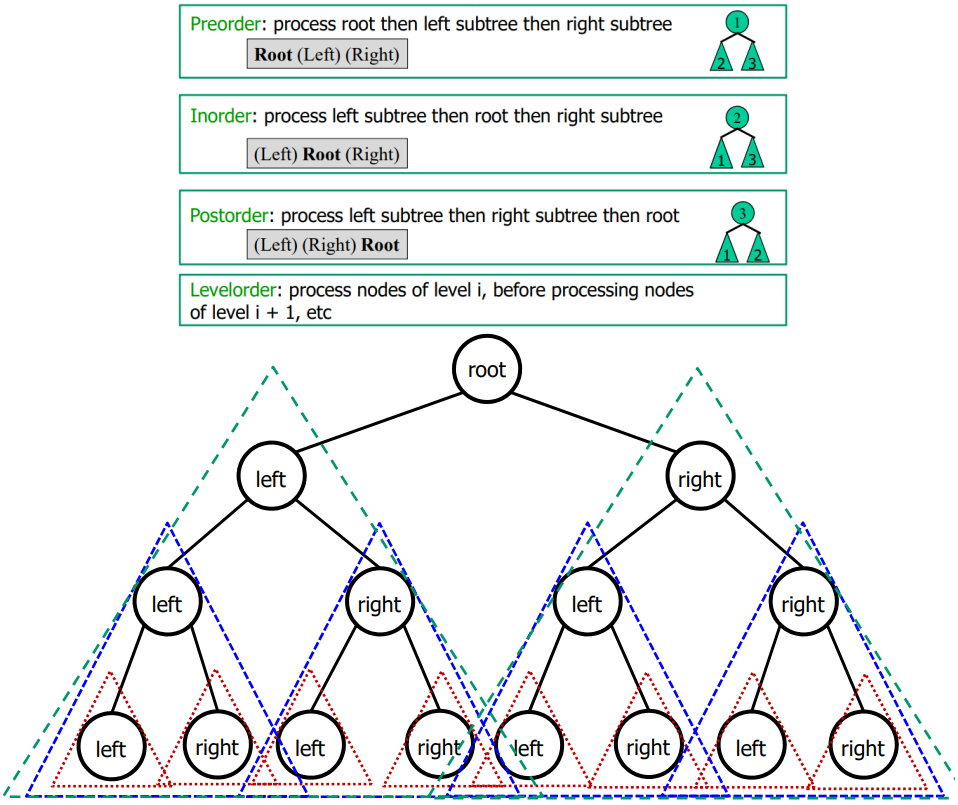
2.2 Tree Branch (Subtree) Class

- Tree branch properties: left child, right child
- Tree branch behavs: n/a

```python
class TreeLeaf:
    def __init__(self, uncompressed_byte):
        self.value = uncompressed_byte

    def __str__(self):
        return 'Leaf storing: ' + self.value

class TreeBranch:
    def __init__(self, lchild, rchild):
        self.left = lchild
        self.right = rchild

    def __str__(self):
        return f'({self.left} <- branch root -> {self.right})'

if __name__ == "__main__":
    leafA = TreeLeaf('a')
    leafB = TreeLeaf('b')
    leafC = TreeLeaf('c')
    print(leafA, leafB)
    branch = TreeBranch(leafA, leafB)
    branch2 = TreeBranch(branch, leafC)
    print(branch)
    print(branch2)

# Output
a b
(a <- branch root -> b)
((a <- branch root -> b) <- branch root -> c)
```

2.3 Binary Tree Traversals

- Base case is leaf
- Processing of left & right subtrees done recursively
- There are 4 common binary tree traversals:



2.4 Binary Tree Traversals: Example

- Preorder: 1 2 3 4 5 6 7 8 9
- Inorder: 4 3 5 2 6 8 7 9 1
- Postorder: 4 5 3 6 2 8 9 7 1
- Levelorder: 1 2 7 3 6 8 9 4 5