Binary Trees

Friday, 8 April 2022 13:55

Binary Trees are also node - based data structure
⇒ Have ≥ 1 pointers for each node, but with hierarchical (below don't point up)
⇒ Made up of Parents and Childrens
✗ Each Node can have a maximum of 2 children

Binary Trees

- Instead of .next (points to 1), we have left and .right
- First node is called the root node
- Any node without children is called a leaf node
- The path between the root node and a leaf node is called a branch

1. TreeNode Class

```
class TreeNode:

    """A class that implements a binary tree."""

    def __init__(self, cargo=None, left=None, right=None):
        """
        (self) -> NoneType
        Create a Node with cargo and left and right subtrees.
        """
        self.cargo = cargo
        self.left = left
        self.right = right

    def __str__(self):
        return '(' + str(self.cargo) + ')'
```

#cleancode

```
tree = TreeNode(0, TreeNode(2), TreeNode(3))
print(tree.cargo)
print(tree.left.cargo)
print(tree.right.cargo)
```

2. Binary Tree Class

```
class BinaryTree:

    """A Node class used by a binary tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node, " ", end = "")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

Traversing through binary trees

→ Can be replaced with cargo_sum to find sum of cargos in tree

~~Falsy Statements. Identity Test~~

Saturday, 9 April 2022 15:05

1. Identity & Equality Test

`is` checks whether the RHS and LHS are very same object
`==` checks whether the RHS and LHS are equal objects (can be different objects)

+) a = "hello world"
b = "hello world"
a is b
→ False

+) a = None
b = None
a is b
→ True
(None is a Python object)

+) a = 'hello world'
b = 'hello world'
a == b
→ True

2. Falsy Values

+) Sequences and Collections
- Empty list `[]`
- Empty tuples `()`
- Empty dictionaries `{}`
- Empty Sets `set()`
- Empty strings `""`
- Empty ranges `range(0)`

+) Numbers
- Zero of any numeric type `0` , `0.0` , `0j`

+) Constants
- None
- False

☆ Any new class will evaluate to True
☆ Any variable created will evaluate to True

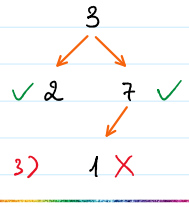
Binary Search Tree

Saturday, 9 April 2022 15:04

A special case of Binary Trees :

Rule node.left.cargo < node.cargo < node.right.cargo

and This rule must be true for the entire tree



- **Binary Search Tree Class**

1. Create class, similar to BinaryTree

```
class BinarySearchTree:

    """A Node class used by a binary sreach tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self):
        """
        (self) -> NoneType
        Prints tree level by level.
        """
        level = [self.root]

        while len(level) > 0:

            level_next = []

            for node in level:

                print(node, " ", end = "")

                if node.left is not None:
                    level_next.append(node.left)
                if node.right is not None:
                    level_next.append(node.right)

            print('\n')
            level = level_next
```

2. Create is_valid method

```
    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:

                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

- **Traversing through binary search tree**

```
def find(self, cargo):
    ''' Return True if there is such cargo in tree
        Else return False '''
    on = self.root
    while on is not None:
        if on.cargo < cargo:
            on = on.right
        elif on.cargo > cargo:
            on = on.left
```

```
            on = on.right
        elif on.cargo > cargo:
            on = on.left
        else:
            return True
    return False
```

- Adding to binary search tree