breadth_first_search

May 10, 2025

breadth first search, depends upon ; Binary Search Tree.

Or, this is usually a search method on binart tree

Breadth first == -> Hoizontal first

Row wise search method

Breadth First Search is a (method) of binary search this. This is gold, you have to understand

[22]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self):
        self.root = None

    def insert(self, value):
        new_node = Node(value)

        if self.root == None:
            self.root = new_node

        temp = self.root
        while True:
            if temp.value == new_node.value:
                return False
            if new_node.value < temp.value:
                if temp.left == None:
                    temp.left = new_node
                    return True
                temp =temp.left
            else:
                if temp.right == None:
                    temp.right = new_node
                    return True
                temp = temp.right
```

```
    def BFS(self):
        current_node = self.root
        queue = []
        results = []
        queue.append(current_node)

# It is important to put your value; append the queue; before you start to run␣
 ↪your while loop
# If you never append, while loop will never run

        while len(queue)>0:
            current_node = queue.pop(0)
            # result.append(current_node) # Remember why i highlight this

            results.append(current_node.value)

            if current_node.left != None:
                queue.append(current_node.left)
            if current_node.right != None:
                queue.append(current_node.right)
        return results
```

[23]:
```
myTree = BinaryTree()
myTree.insert(30)

print(myTree.root.value)

print(myTree.insert(40))
print(myTree.insert(20))

print(myTree.insert(60))

print(myTree.root.left.value)
print(myTree.root.right.value)

print(myTree.root.right.left)
print(myTree.root.right.right)

print(myTree.root.right.right.value)



myTree.BFS()
```

```
30
True
True
True
20
40
None
<__main__.Node object at 0x7fe5d40ab220>
60
```

[23]:
```
[30, 20, 40, 60]
```

[29]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self):
        self.root = None

    def insert(self,value):
        new_node = Node(value)
        if self.root == None:
            self.root = new_node
            return 50

        temp = self.root
        while True:
            if new_node.value == temp.value:
                return False
            if new_node.value < temp.value:
                if temp.left == None:
                    temp.left = new_node
                    return True
                temp = temp.left
            else:
                if temp.right == None:
                    temp.right = new_node
                    return True
                temp = temp.right

    def BFS(self):
        queue = []
        results = []
        current_node = self.root
```

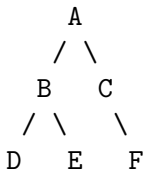
```
        queue.append(current_node)
        while len(queue) > 0 :
            current_node = queue.pop(0)
            results.append(current_node.value)
            if current_node.left is not None:
                queue.append(current_node.left)
            if current_node.right is not None:
                queue.append(current_node.right)
        return results

myTree = BinaryTree()
myTree.insert(50)
myTree.insert(60)
myTree.insert(70)
myTree.insert(5)
myTree.insert(15)
myTree.insert(2)

myTree.BFS()
```

[29]: [50, 5, 60, 2, 15, 70]

1 DFS



S.N	Traversal Type	Visit Order	Result
1	Preorder	Node → Left → Right	A B D E C F
2	Postorder	Left → Right → Node	D E B F C A
3	Inorder	Left → Node → Right	D B E A C F

2 DFS

2.1 1. Pre-order

[38]:
```
class Node:
    def __init__(self, value ):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree:
```

```
    def __init__(self):
        self.root = None

    def insert(self, value):
        new_node = Node(value)
        if self.root == None:
            self.root = new_node
            return True

        temp = self.root

        while (True):
            if temp.value == new_node.value:
                return False

            if new_node.value < temp.value:
                if temp.left == None:
                    temp.left = new_node
                    return True
                temp = temp.left
            else:
                if temp.right == None:
                    temp.right = new_node
                    return True
                temp = temp.right

    def dfs_pre_order(self):
        results = []

        # recursive function
        def traverse(current_node):
            # results.append(current_node)
            results.append(current_node.value)
            if current_node.left != None:
                traverse(current_node.left)
            if current_node.right != None:
                traverse(current_node.right)

        # yaha katai function cull ni bhako xaina
        traverse(self.root)

        return results


myTree = BinaryTree()
myTree.insert(50)
myTree.insert(60)
```

```
myTree.insert(70)
myTree.insert(5)
myTree.insert(15)
myTree.insert(2)

print(myTree.root.value)
print(f'--------')

myTree.dfs_pre_order()
```

```
50
--------
```

[38]: [50, 5, 2, 15, 60, 70]

3 DFS

3.1 2. Post-order

[40]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree():
    def __init__(self):
        self.root = None

    def insert(self, value):
        new_node = Node(value)
        if self.root == None:
            self.root = new_node

        temp = self.root

        while (True):
            if temp.value == new_node.value:
                return False

            if new_node.value < temp.value:
                if temp.left == None:
                    temp.left = new_node
                    return True
                temp = temp.left
            else:
                if temp.right == None:
```

```
                    temp.right = new_node
                    return True
                temp = temp.right

    def DFS_post_order(self):
        results = []

        current_node = self.root
        def traverse(current_node):
            if current_node.left != None:
                traverse(current_node.left)
            if current_node.right != None:
                traverse(current_node.right)
            # results.append(current_node) # mistake
            results.append(current_node.value)

        traverse(self.root)

        return results

myTree = BinaryTree()
myTree.insert(47)
myTree.insert(21)
myTree.insert(76)
myTree.insert(18)
myTree.insert(27)
myTree.insert(52)
myTree.insert(82)

print(myTree.root.value)
print(f'--------')

myTree.DFS_post_order()
```

```
47
--------
```

[40]: `[18, 27, 21, 52, 82, 76, 47]`

4 DFS

4.1 3. In-order

[44]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
```

```
        self.right = None

class BinaryTree:
    def __init__(self):
        self.root = None

    def insert(self,value):
        new_node = Node(value)

        if self.root == None:
            self.root = new_node
            return True

        temp = self.root

        while (True):
            if temp.value == new_node.value:
                return False

            if new_node.value < temp.value:
                if temp.left == None:
                    temp.left = new_node
                    return None
                temp = temp.left
            else:
                if temp.right == None:
                    temp.right = new_node
                    return True
                temp = temp.right

    def DFS_in_order(self):

        results = []

        current_node = self.root

        def traverse(current_node):
            if current_node.left != None:
                traverse(current_node.left)
            # results.append(current_node)
            results.append(current_node.value)
            if current_node.right != None:
                traverse(current_node.right)

        traverse(self.root)

        return results
```

```
myTree = BinaryTree()
myTree.insert(47)
myTree.insert(21)
myTree.insert(76)
myTree.insert(18)
myTree.insert(27)
myTree.insert(52)
myTree.insert(82)

print(myTree.root.value)
print(f'--------')

myTree.DFS_in_order()
```

```
47
--------
```

[44]: [18, 21, 27, 47, 52, 76, 82]