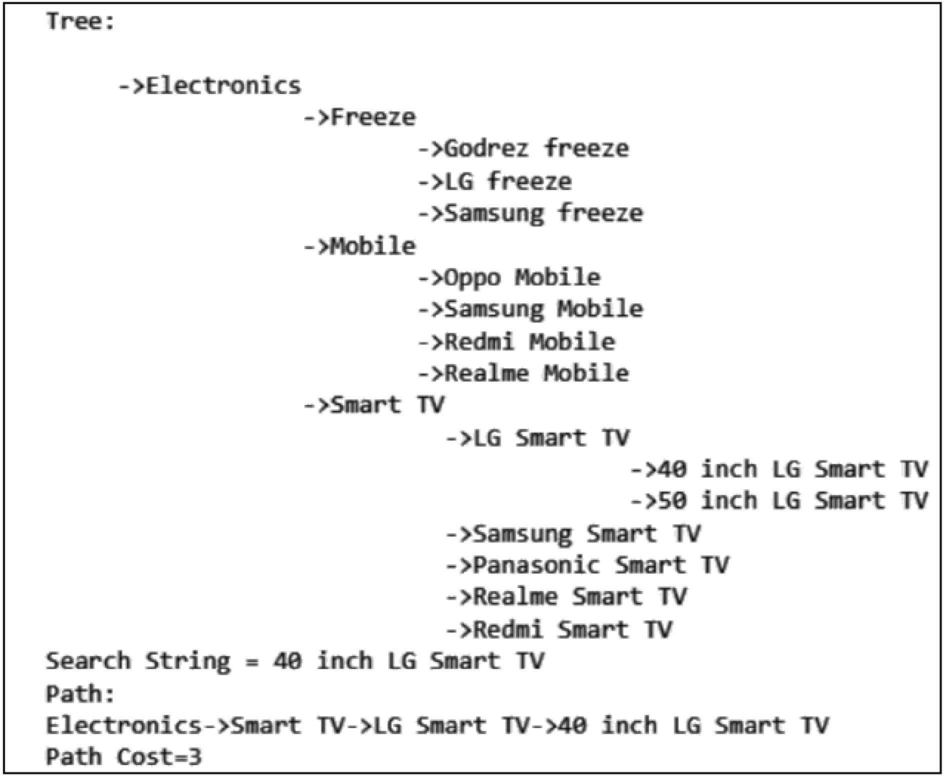
Aim: Write a program to implement Breadth first search Traversal on Tree using Python (without using any libraries or packages of python).

- **Use class concept of python (Tree Class, Node Class).**
- **Use class to implement Data structure to be used in program.**
- **Tree & Output should look like below:**

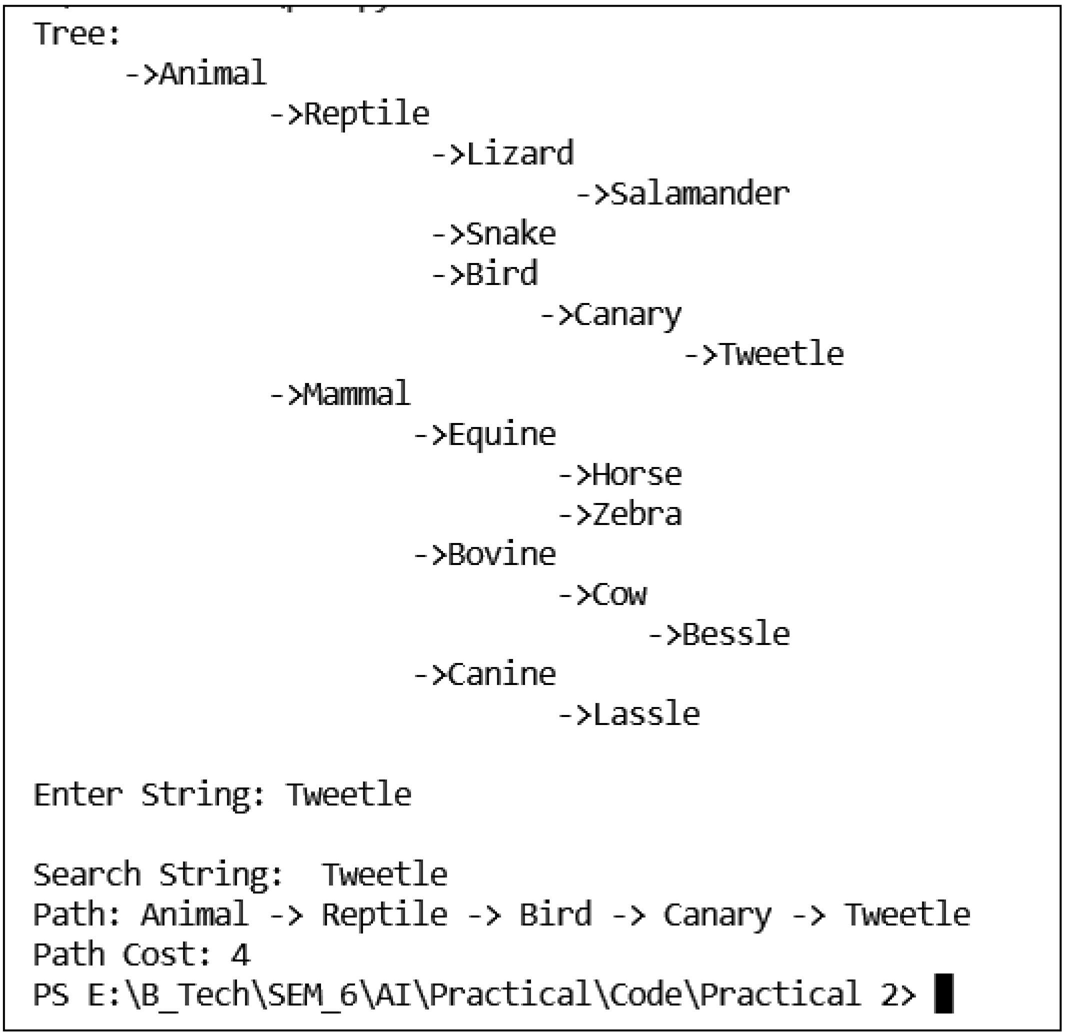


```
Tree:

     ->Electronics
                  ->Freeze
                          ->Godrez freeze
                          ->LG freeze
                          ->Samsung freeze
                  ->Mobile
                          ->Oppo Mobile
                          ->Samsung Mobile
                          ->Redmi Mobile
                          ->Realme Mobile
                  ->Smart TV
                            ->LG Smart TV
                                         ->40 inch LG Smart TV
                                         ->50 inch LG Smart TV
                            ->Samsung Smart TV
                            ->Panasonic Smart TV
                            ->Realme Smart TV
                            ->Redmi Smart TV
Search String = 40 inch LG Smart TV
Path:
Electronics->Smart TV->LG Smart TV->40 inch LG Smart TV
Path Cost=3
```

Program:

```
# Contains the node attributes
# @parent(Object of node class),
# @nodeValue(String),
# @childNodeList(List of child Node Object)
# @nodeLevel(Integer)
class node:
    def __init__(self, parent, value, child):
        self.parent = parent
        self.value = value
        self.child = child
        self.level = 0
        if parent != None:
            self.level = self.parent.level + 1

    def addChildNode(self, childNode):
        self.child.append(childNode)
    def spaceCount(self):
        strParent = ""
        if self.parent == None:
            strParent = str(self.parent)
        else:
            strParent = "None"
            tempNode = self.parent
            while tempNode != None:
                strParent += "->" + tempNode.value
                tempNode = tempNode.parent
        return len(strParent) + 1

    def __repr__(self):
        strReturn=""
        if(self.parent==None):
            strReturn = "Tree: "
        strReturn += "\n"
        strReturn += " " * self.spaceCount()

        # map() function returns a map object(which is an iterator) of the results
        # after applying the given function to each item of a given iterable (list, tuple etc.).
        # Syntax:- map(fun, iter)
        # @fun : It is a function to which map passes each element of given iterable.
        # @iter : It is a iterable which is to be mapped.

        #join(): The join() method takes all items in an iterable and joins them into one string.
        #@return : String
        strReturn += "->" + str(self.value) + "  ".join(map(str, self.child))
        return strReturn

# Contains root node data also use to add node in tree
# @rootNode (Node class object)
class tree:
    def __init__(self, rootnode):
        self.root = rootnode

    def addNode(self, value, parentNode):
        nd = node(parentNode, value, [])
        parentNode.addChildNode(nd)
        return nd
    def __repr__(self):
        # call built in __repr__ method of node class
        return str(self.root)

# Create tree with root node
t1 = tree(node(None, "Animal", []))

# Add node in tree
# @nodeValue (String)
# @parentNode (Object of node class)
node1 = t1.addNode("Reptile",t1.root)

node1_1 = t1.addNode("Lizard",node1)
node1_1_1 = t1.addNode("Salamander",node1_1)

node1_2 = t1.addNode("Snake",node1)

node1_3 = t1.addNode("Bird",node1)
node1_3_1 = t1.addNode("Canary",node1_3)
node1_3_1_1 = t1.addNode("Tweetle",node1_3_1)

node2 = t1.addNode("Mammal",t1.root)
node2_1 = t1.addNode("Equine",node2)
node2_1_1 = t1.addNode("Horse",node2_1)
node2_1_2 = t1.addNode("Zebra",node2_1)

node2_2 = t1.addNode("Bovine",node2)
node2_2_1 = t1.addNode("Cow",node2_2)
node2_2_1_1 = t1.addNode("Bessle",node2_2_1)

node2_3 = t1.addNode("Canine",node2)
node2_3_1 = t1.addNode("Lassle",node2_3)

# Print the tree t1
print(t1)

# Function use for search node using Breadth First Search Algo
def bfs(searchString, rootNode):
    node = [rootNode]
    while node:
        tempNode = node.pop(0)
        if tempNode.value == searchString:
            return tempNode
        else:
            # extend(): The extend() method will extend
            # the iterable by appending all the elements from the iterable to the end of the list.
            # Example:
            # lis=[1,2]
            # lis.extend([3,4,5])
            # output: lis[1,2,3,4,5]
            node.extend(tempNode.child)
    return None

# Function use to find cost of searching node
def findCost(self):
    stringDisp = []
    ndd = self.parent
    while ndd != None:
        # append(): The append() method will add an item to the end of the list.
        # Example :
        # lis=[1,2]
        # lis.append([3,4,5])
        # output: lis[1,2,[3,4,5]]
        stringDisp.append(ndd.value)
        ndd = ndd.parent
    stringDisp.reverse()
    return stringDisp

searchSTR = input("\nEnter String: ")
print("\nSearch String: ",searchSTR)

# Perform Breadth First Search. Searching Start from rootNode
breadthF = bfs(searchSTR, t1.root)
if breadthF == None:
    print("Sorry, we can't find this string.")
else:
    # Find Cost of searching node from searchNode to rootNode
    res = findCost(breadthF)
    strPath = res[0]
    for i in range(1,len(res)):
        strPath += " -> " + res[i]
    strPath += " -> " + breadthF.value
    print("Path: " + strPath)
    print("Path Cost: " + str(len(res)))
```

Output:



```
Tree:
     ->Animal
             ->Reptile
                      ->Lizard
                              ->Salamander
                      ->Snake
                      ->Bird
                            ->Canary
                                    ->Tweetle
             ->Mammal
                     ->Equine
                             ->Horse
                             ->Zebra
                     ->Bovine
                             ->Cow
                                  ->Bessle
                     ->Canine
                             ->Lassle

Enter String: Tweetle

Search String:  Tweetle
Path: Animal -> Reptile -> Bird -> Canary -> Tweetle
Path Cost: 4
PS E:\B_Tech\SEM_6\AI\Practical\Code\Practical 2> █
```