

Tutorial: Binary Trees and Binary Search Trees

1. You are given a pre-order traversal and an inorder traversal of a binary tree. Draw the binary tree from the two traversal results.

pre-order: A, B, C, D, E, F, G

inorder: C, B, E, D, A, F, G

2. (**levelOrderTraversal**) Write an iterative function `levelOrderIterative()` that prints a level-by-level traversal of the binary search tree using a queue, starting at the root node level. The function uses **only** `enqueue()` or `dequeue()` operations to add or remove nodes from the queue. Ensure the queue is empty at the beginning of the traversal.

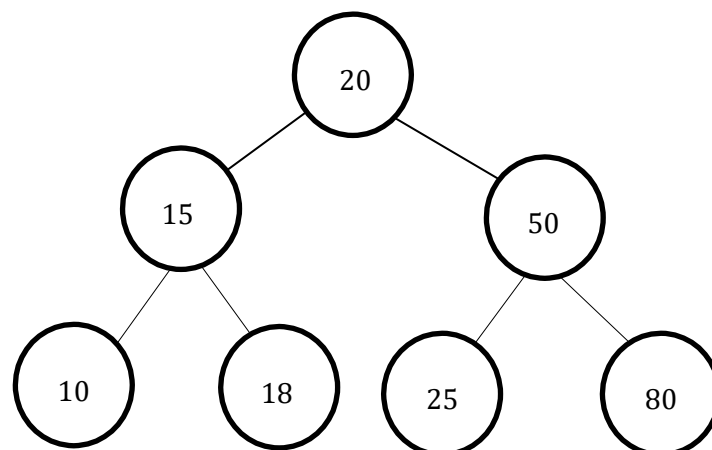
The function prototype is given as follows:

```
def levelOrderIterative(root):
```

Following is the detailed algorithm:

- 1) Create an empty queue `q`
- 2) If tree is not empty, then Enqueue root to the Queue
- 3) Repeat until Queue is empty
 - a) Dequeue node's data from the queue and print it
 - b) Enqueue node's left child to the `q`
 - c) Enqueue node's right child to the `q`

Let's consider the below tree for example.



Level-order Tree Traversal: **20 15 50 10 18 25 80**

3. (**preOrderIterative**) Write an iterative function `preOrderIterative()` that prints the pre-order traversal of a binary search tree using a **stack**. The function uses **only** `push()` or `pop()` operations to add or remove nodes from the stack. Ensure the stack is empty at the beginning of the traversal.

The function prototype is given as follows:

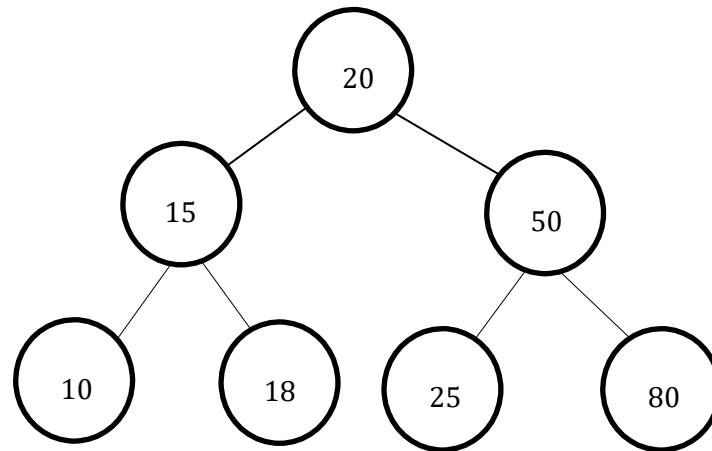
```
def preOrderIterative(root):
```

Following is the detailed algorithm:

- 1) Create an empty stack `nodeStack` and push root node to stack.
- 2) Do following while `nodeStack` is not empty.
 - a) Pop an item from stack and print it.
 - b) Push right child of popped item to stack
 - c) Push left child of popped item to stack

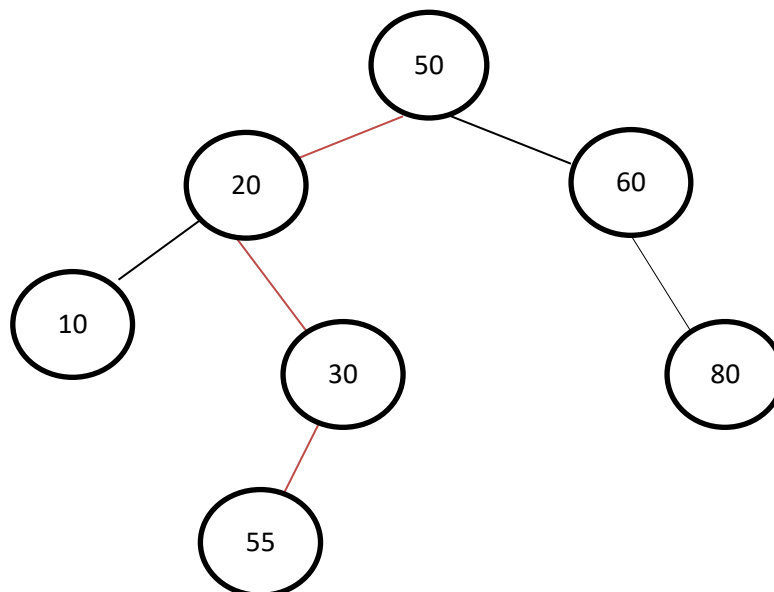
Right child is pushed before left child to make sure that left subtree is processed first.

Let's consider the below tree for example.



Preorder Tree Traversal: **20 15 10 18 50 25 80**

4. (**maxDepth**) Write a Python function to find the maximum depth of a binary tree.



The function prototype is given as follows:

```
def maxDepth(node) :
```

Some sample inputs and outputs sessions are given below:

1: Create a binary tree.

2: Find the maximum depth of the binary tree.

0: Quit;

Please input your choice(1/2/0): 1

Input an integer that you want to add to the binary tree. Any Alpha value will be treated as NULL.

Enter an integer value for the root: 50

Enter an integer value for the Left child of 50: 20

Enter an integer value for the Right child of 50: 60

Enter an integer value for the Left child of 20: 10

Enter an integer value for the Right child of 20: 30

Enter an integer value for the Left child of 10: a

Enter an integer value for the Right child of 10: a

Enter an integer value for the Left child of 30: 55

Enter an integer value for the Right child of 30: a

Enter an integer value for the Left child of 55: a

Enter an integer value for the Right child of 55: a

Enter an integer value for the Left child of 60: a

Enter an integer value for the Right child of 60: 80

Enter an integer value for the Left child of 80: a

Enter an integer value for the Right child of 80: a

The resulting binary tree is: 10 20 55 30 50 60 80

Please input your choice(1/2/0): 2

Find the maximum depth of the binary tree: 3

Please input your choice(1/2/0): 0