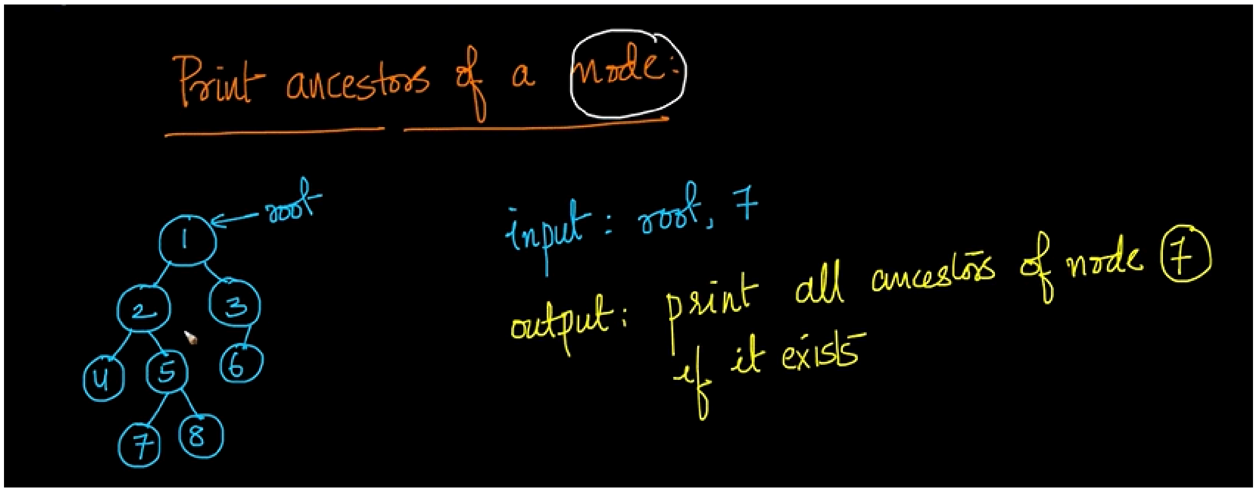
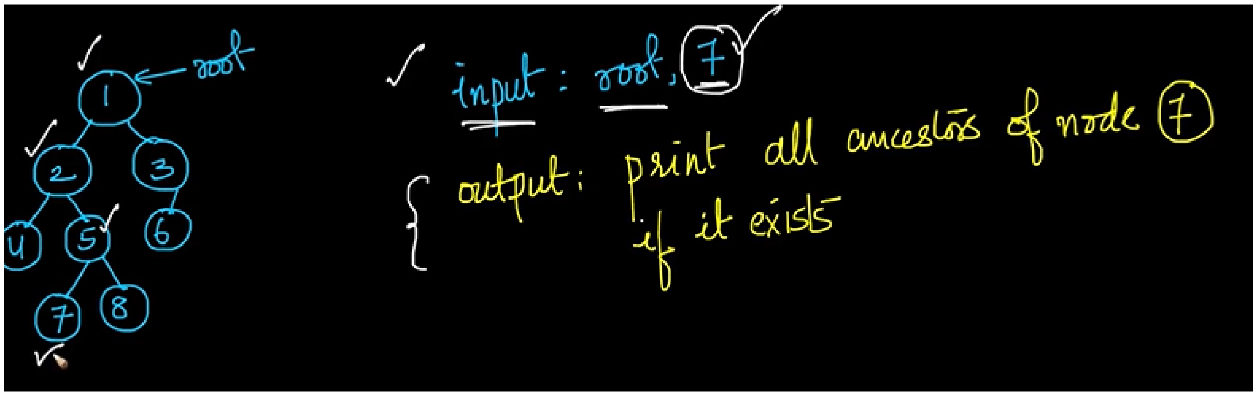
Print Ancestors of a node:

Problem Statement



Print ancestors of a node:
root
1
2
3
4
5
6
7
8
input : root, 7
output: print all ancestors of node 7 if it exists

1. Given Input as root and node…print all the ancestors of node if it exists
2. 

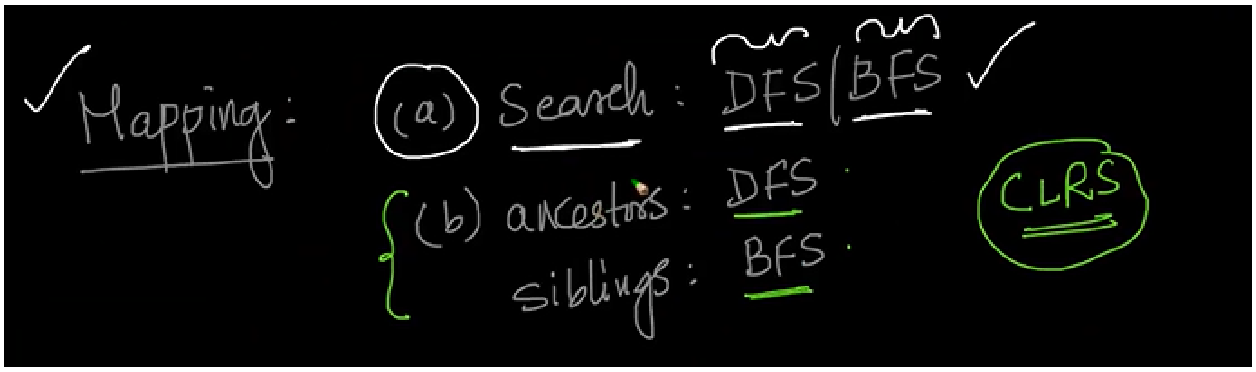
   So here ancestors of node 7 …are 5,2,

Approach

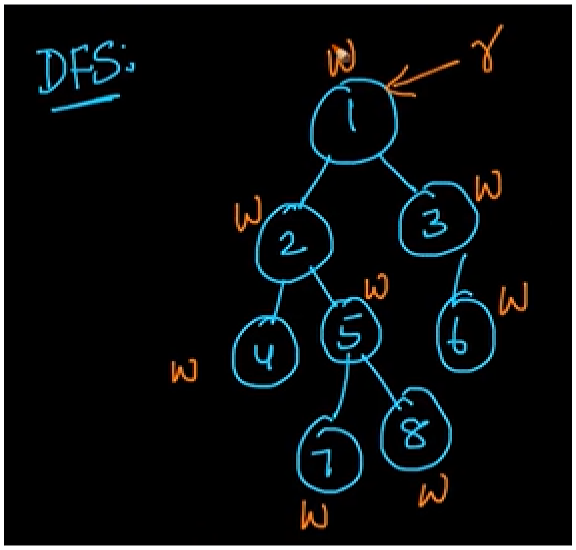
Tasks: Search
Print ancestors

1. Here we need to deal with 2 tasks
   FIrst we need to search for our node ..and then next print its ancestors
2. To solve any problem..first we have to understand the question and map to a technique which we have solved earlier

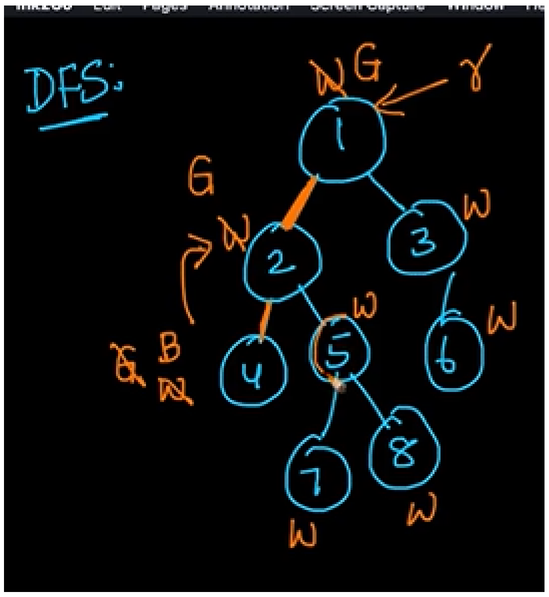
3. So here to search for any elements…we can use DFS/BFS
4. And to find the ancestors we can make use of DFS…and to find the siblings we can make use of BFS



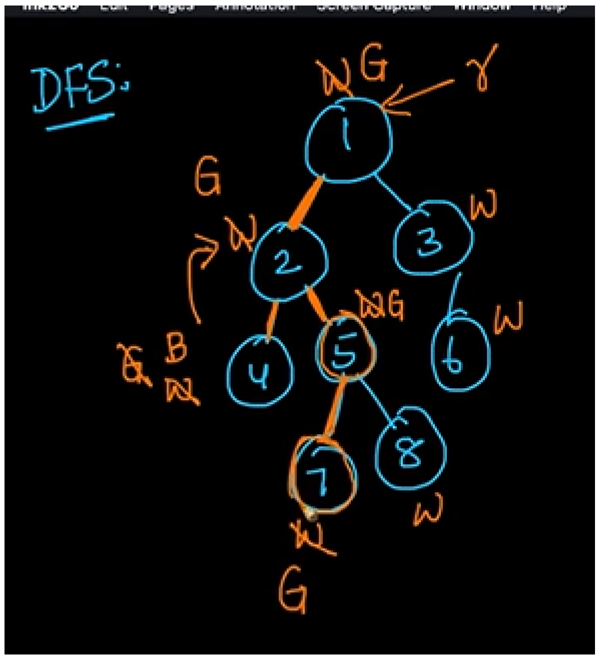
5. Now our first approach is to use DFS
6. Should Learn DFS and BFS in more depth
7. DFS Approach…here initially every node is colored with the white



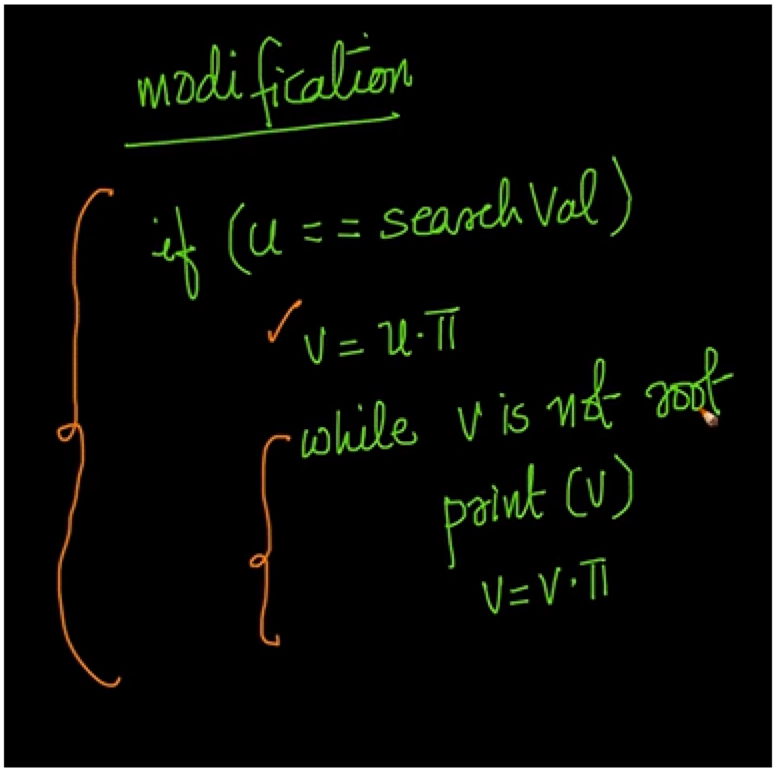
8. Now we start with the root and change its colour to Grey as we have visited it
9. From root node 1…we'll move to its left children node 2
10. Now node'2 predecessor is node1….similarly from node 2 ..we'll move to node 4
11. Node4's predecessor is node2

12. As node 4 has no children..we roll back to its predecessor(node2) and explore its right children 

13. Now we move to node 5 from node 2…and from node 5 to node 7



14. Here we have performed DFS to search for 7…and we have found our node
15. And now we'll print the predecessor's and this gives the ancestors of node 7

16. We need to slightly modify the DFS


modification

```
{
  if (u == searchVal)
      ✓ v = u.π
      while v is not root
      {
          print(v)
          v = v.π
      }
}
```

here ..if we found our element…v is its predecessor……
17. Now while V is not root…we go to V’s predecessor until we reach the root

Python code for iterative approach

```python
def iterativeAncestors(root, ele):
        if not root:
            return False
        if root.left == ele or root.right == ele: # if the given node is a child of root node
            print(root.data)
            return
        stack = []
        visited = set()
        stack.append(root)
        while stack:
            cur = stack[-1]
            if cur.data == ele: # if we find the node break the loop
                break
            if not cur.left and not cur.right: # if the current node is a leaf node
                temp = stack.pop(-1)
                visited.add(temp)
                continue
            if cur.left and cur.left not in visited:
                stack.append(cur.left)
            elif cur.right:
                if cur.right in visited:
                    temp = stack.pop(-1)
                    visited.add(temp)
                else:
                    stack.append(cur.right)
        if not stack: # all the nodes are searched but no node is equal to the given node
            return False
        while stack:
            cur = stack.pop(-1)
            print(cur.data)
        return
```

1.