MICP Week 6 Homework

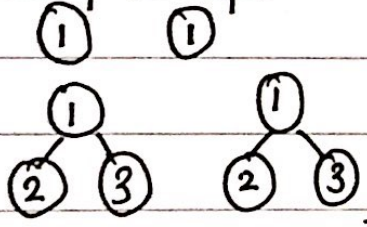
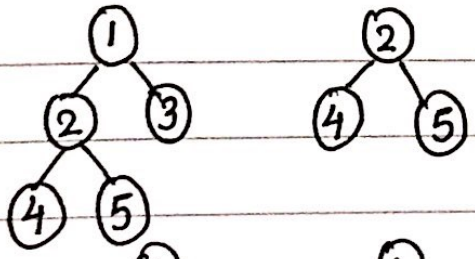
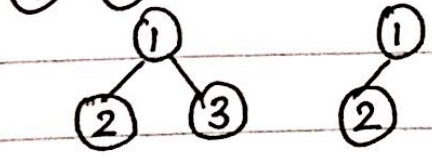
Subtree of another tree

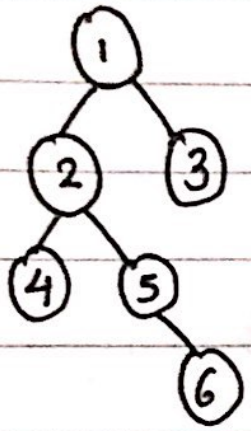
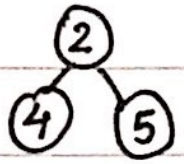
#TEBOW IT

T-Talk :

Can the trees be empty? No given in question - non empty

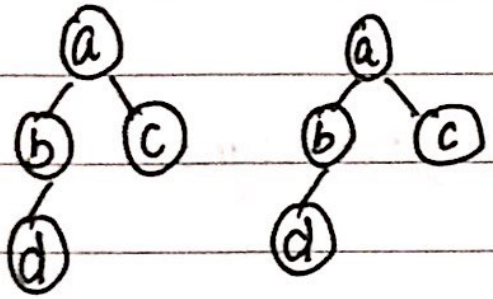
E-Examples :

Sample Input	Equivalence class	Output
	Trees with single node	True
	Fully same	True
	Successful subtree	True
	Simple fail	False

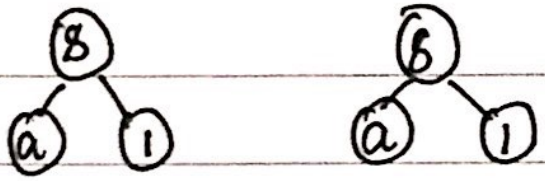
Fail because of 1 or more missing node

False



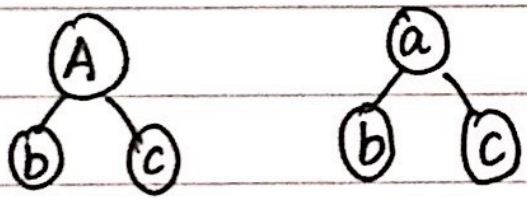
Alphabets

True



Other characters

True

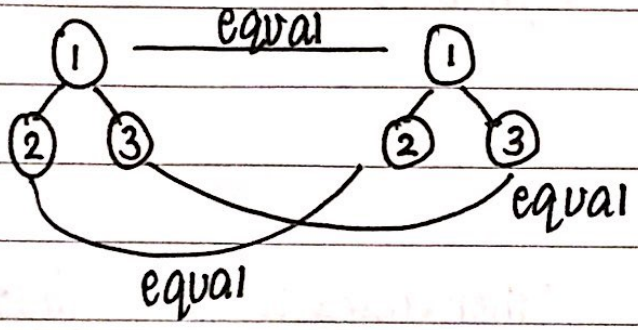


Case sensitivity

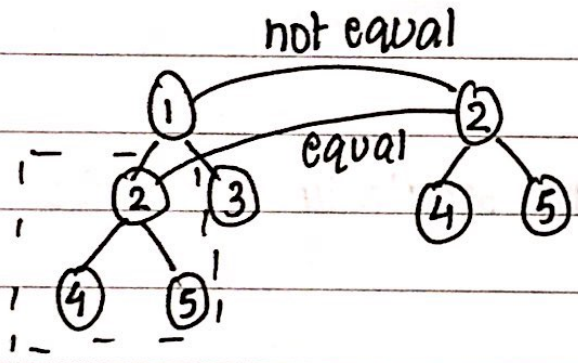
False

B-Brute Force :

- Check if the given trees are equal
- Check if one of the trees descendants - left or right is equal to the other tree



root, left, right



root.left

- check at root for equality
- Check for equality in left and right subtrees
- Recursively check subtrees

Time complexity - $O(n1) * O(n2)$

$= O(n1\,n2)$

n1 : # of nodes of tree1

n2 : # of nodes of tree2

O - optimize:

W - Walk Through:

① A function to check for equality
- at root - if values are equal
- recursively at left and right

② A function to
- call the above function at root as well as at left and right subtrees

equal(T,S)
equal(T.left, S)
equal(T.right, S)

- Keeping in mind to check if S becomes None.

I - Implement:

```
def equal(t1,t2):
    if t1 is None or t2 is None:
        return (t1 is None) and (t2 is None)
    else:
        return t1.value == t2.equal and
               equal(t1.left, t2.left) and
               equal(t1.right, t2.right)

def isSubtree(S,T):
    if T is None:
        return True
    if S is None:
        return False
    if equals(S,T):
        return True
    return isSubtree(S.left,T) or
           isSubtree(S.right,T)
```

T - Test