Homework → subtree of another tree

Week 6 - Trees - 2-08-2017

Given two non-empty binary trees s and t, check whether tree t has exactly the same structure and node values with a subtree of s. A subtree of s is a tree that consists of a node in s and all of that node's descendants.

T - TALK

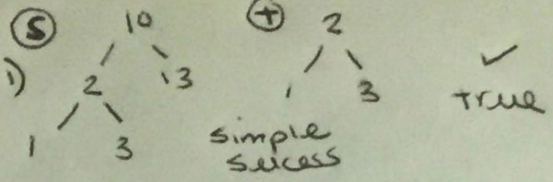
- Can the trees be empty? No
- Input? Two root nodes
- Output? Boolean
- [redacted]
- No need to create tree structure
- Binary tree can have duplicate values. (not BST)
- Binary tree which only contains numbers

B - BRUTE FORCE

- Traverse both trees and store the values in different vectors
- Find if the subsequence of t is in s.
- Need extra memory for storing values in the vecters.

E - EXAMPLES

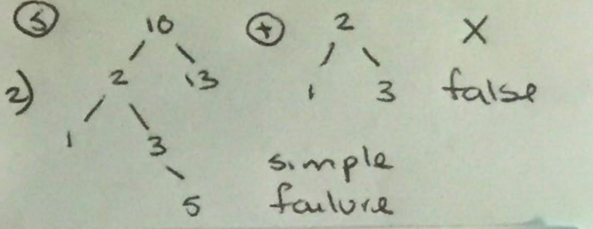
1)



✓ true

simple success

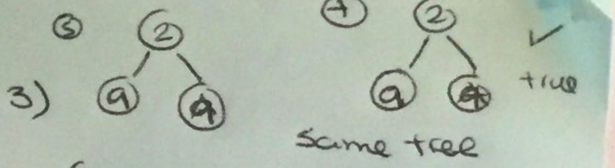
2)



X false

simple failure

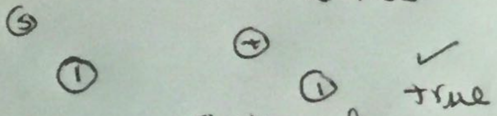
3)



✓ true

Same tree

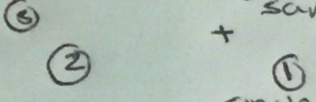
4)



✓ true

single node same

5)



X false

single node different

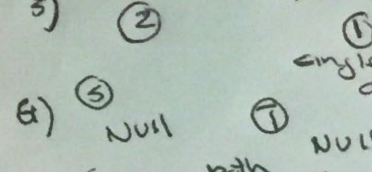
6) s NULL, t NULL ✓ true

both null

7) s NULL, t 1 2 only 1 null X false

O - OPTIMIZATION

- Recursively traverse the tree in preorder, comparing the nodes of t and s. and verifying if they are identical.



Time complexity (m * n)

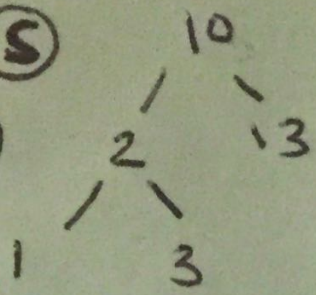
m: nodes in t

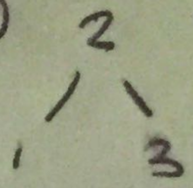
n: nodes in s

T-TALK

- Can the trees be empty? No
- Input? Two root nodes
- Output? Boolean
- [redacted]
- No need to create tree structure
- Binary tree can have duplicate values. (not BST)
- Binary tree which only contains numbers

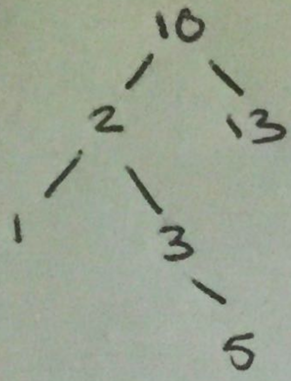
E - EXAMPLES

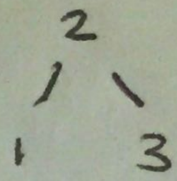
1) Ⓢ 

Ⓣ 

✓ true

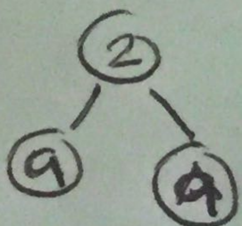
simple success

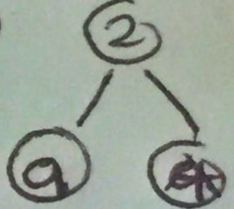
2) Ⓢ 

Ⓣ 

X false

simple failure

3) Ⓢ 

Ⓣ 

✓ true

same tree

4) Ⓢ ①

Ⓣ ①

✓ true

single node same

5) Ⓢ ②

Ⓣ ①

X false

single node diffrent

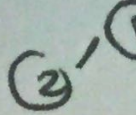
6) Ⓢ NUll

Ⓣ NUll

✓ true

both null

7) Ⓢ NUll



X false

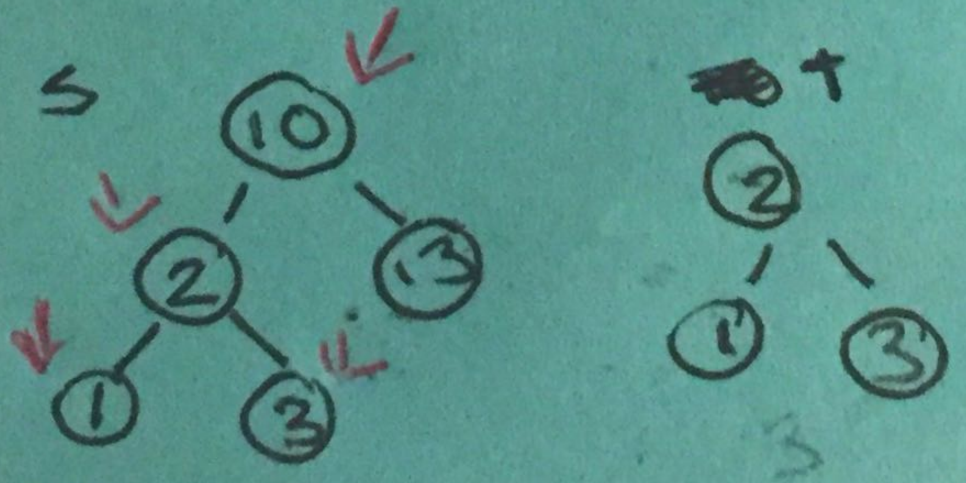
only 1 null

B- BRUTE FORCE

- Traverse both trees and store the values in different vectors.
- Find if the subsequence of t is in s.
- Need extra memory fo storing values in the vecters.

O-OPTIMIZATION

- Recursively traverse the tree in preorder, comparing the nodes of t and s. and verifying if they are identical.



Time complexity (m * n)

nodes in t

nodes in s

W-WALKTHROUGH

- Function to see is they are Equal Trees
    - → returns boolean
    - → If S && T arrive NULL
        - → then true
    - → If only 1 = NULL
        - → then false
    - → Check both trees structures recursively
- Function to traverse in preorder
    - → Return true is S != NULL and if not equal traverse S.
- Main function to call traverse()

I-IMPLE-MENTATION

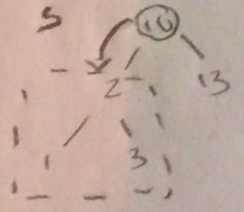
W-WALKTHROUGH

- Function to see is they are Equal Trees
  - → returns boolean
  - → If S && T arrive NULL
    - → then true
  - → If only 1 = NULL
    - → then false
  - → Check both trees structures recursively
- Function to traverse in preorder
  - → Return true is S != NULL and if not equal traverse S.
- Main function to call traverse

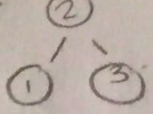
```
bool isEqual (node *s , node *t) {
    if (s == NULL && t == NULL) {
        return true;
    }
    if ( s== NULL || t == NULL) {
        return false;
    }
    return    s→val == t→val &&
              isEqual (s→left, t→left) &&
              isEqual (s→right, t→right);
}

bool traversePreorder (node *s, node *t) {
    return    s != NULL &&
          (is Equal(s, t) || traversePreorder (s→left, t) ||
                             traverse Preorder (s→right, t) );
}

bool isSubtree (node *s, node *t) {
    return    traverse Preorder (s, t);
}
```

T- TEST

s



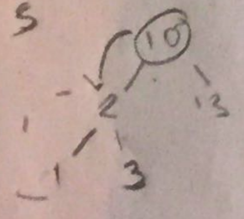
t



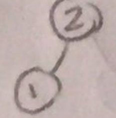
→ S != NULL → S = 10
isEqual (10, 2) → false
s→ left = 2

S != NULL → s = 2
is Equal (2, 2) → true
is Equal (1, 1) → true
is Equal (3, 3) → true
✓ SUCCESS

s



t



s != NULL → s = 10
is Equal (10, 2) → false
S → left = 2

S != NULL → s = 2
is Equal (2,2) → true
is Equal (1, 1) → true
is Equal(3, NULL) → False X

X Fail

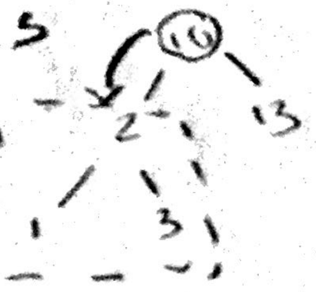
I-IMPLE-MENTATION

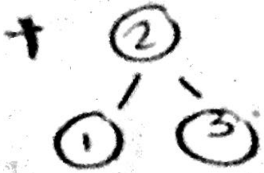
```
bool isEqual (node *s , node *t) {
    if (s == NULL && t == NULL) {
        return true;
    }
    if ( s== NULL || t == NULL) {
        return false;
    }
    return    s->val == t->val &&
              isEqual (s->left, t->left) &&
              isEqual (s->right, t->right);
}


bool traverse Preordor (node *s, node *t){
    return    s != NULL &&
          (is Equal(s, t) || traverse Preorder (s->left ,t) ||
                              traverse Preorder (s->right, t) );
}

bool isSubtree (node *s, node *t){
   return  traverse Preordor (s, t);
}
```

W-WALKTHRO

- Function to see i
  are Equal Trees
  - → returns boolean
  - → If S && T are
    - → then true
  - → If only 1 = N
    - → then false
  - → Check both tree
    structures recursi
- Function to traverse
  preorder
  - → Return true is S
  and if not equal trav
- Main function to call

T- TEST



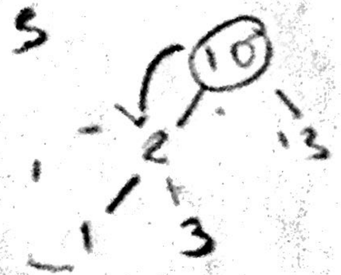


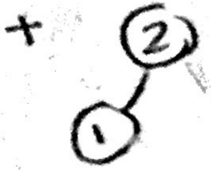
→

S != NUll → s = 10
isEqual (10, 2) → false
s → left = 2

S != NUll → s = 2
is Equal (2, 2) → true
is Equal (1, 1) → true
is Equal (3, 3) → true

✓ SUCCESS





S != NUll → s = 10
isEqual(10, 2) → false
S → left = 2

S != NULL → s = 2
is Equal (2, 2) → true
is Equal (1, 1) → true
is Equal(3, NUll) → False ✗

✗ Fail