Traversal + View

30 March 2024 02:08 PM

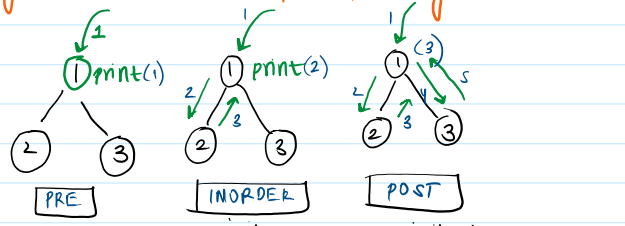
Tree Traversal Docs: https://takeuforward.org/binary-tree/binary-tree-traversal-inorder-preorder-postorder/

Preorder Inorder Postorder Traversals in One Traversal

TC → O(N)
SC → O(3N)

→ How many times you visit a node before printing it?

1. Preorder (1x)
   Print → L → R
2. INORDER (2x)
   L → Print → R
3. POST ORDER (3x)
   L → R → Print



INORDER
→ visit 1x
→ go left
→ visit 2x

POST
→ visit 1x
→ go left → visit 2x
→ go right → visit 3x
→ print

SOLN

```
Q = [(Root, 1)]
while(Q):
    cur = Q.pop()
    if cur[1] == 1:
        add preorder
        Q ← (curr, 2)
        Q ← (left, 1)
    elif cur[1] == 2:
        add inorder
        Q ← (curr, 3)
        Q ← (right, 1)
    elif cur[1] == 3:
        add postorder
```

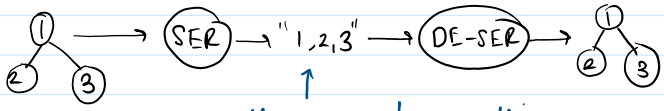
1 means you have pre → print → left subtree ✓

2 → INORDER → (L) → print → right tree

3 → ① you dont add
② L → R → print ✗

SERIALIZE + DESERIALIZE

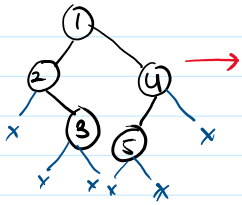
→ Given a tree convert it into any str rep (can use any delim or way)
→ convert it back to tree



You can choose this rep way

SERIALIZE

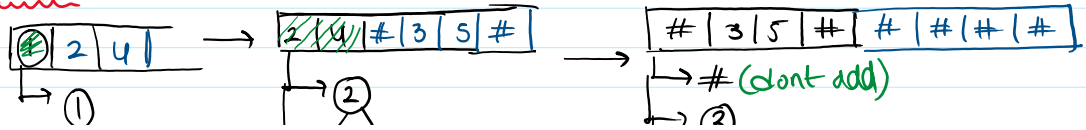
problem → Handle null/empty nodes
→ SOLUTION : Replace NULL → # in str + Level order traversal

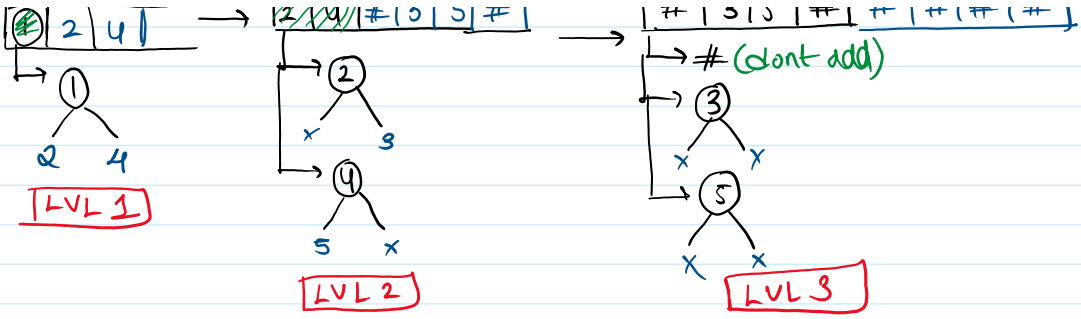


→ (1, 2, 4, #, 3, 5, #, #, #, #, #)

→ Handle commas added at end of str
→ return only val in case only root node

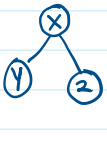
DRY RUN





DESERIALIZE

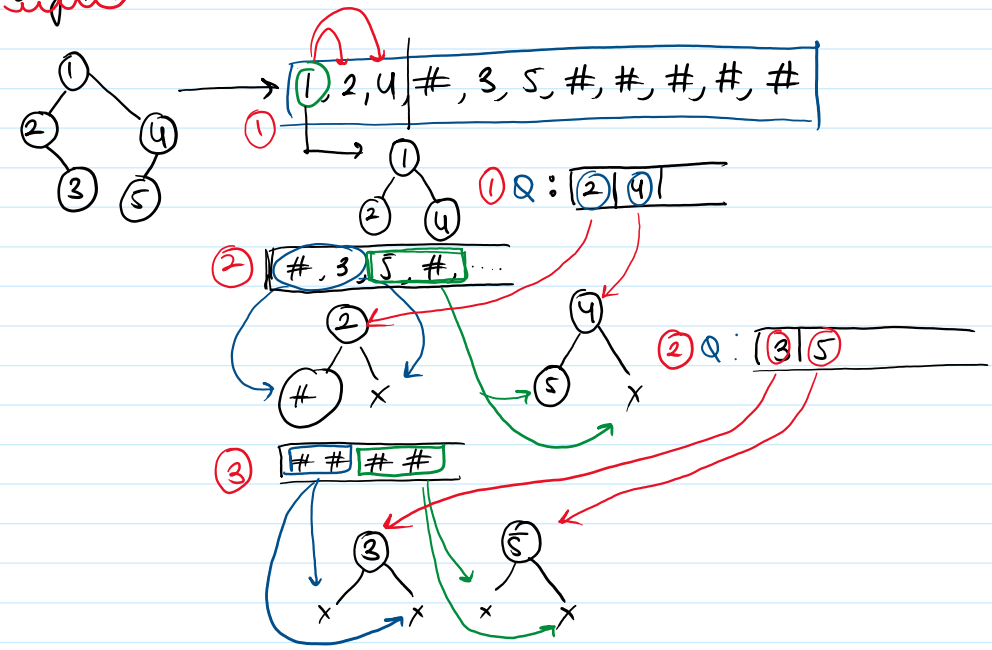
* In SER we made sure we added 2 nodes ∀ node
  * left
  * right (# → nulls)
* x | y | z | ...



s[0] = root
s[0].left = s[1]
s[0].right = s[2]
} NULL if = #

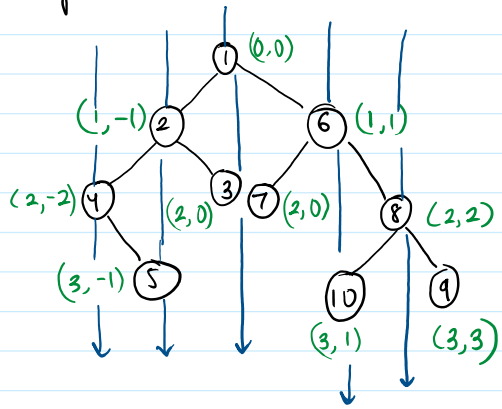
IMP
→ if # dont add to Q
else add to Q

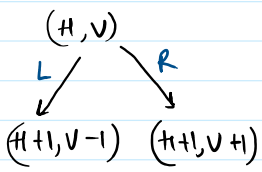
Dry Run



VERTICAL TRAVERSALS

→ Assign vertical and horizontal levels to tree nodes





Concept Based

30 March 2024 05:14 PM

IS BT BALANCED

→ BT is balanced if abs(depth(LT) − depth(RT)) ≤ 1 ∀ subtrees

BRUTE FORCE O(N×N)

→ DFS on nodes

→ dfs(node): depth()

→ if abs(LH, RH) ≥ 1 repeat calls

ret False

→ ret dfs(n.L) & dfs(n.R)

OPTIMAL – O(N)

→ use postorder traversal

→ is_bal(node) {

if not node:

ret 0

L { LH ← is_bal(n.L)

if LH == −1 → −1 LT Imb

R { RH ← is_bal(n.R)

if RH == −1 → −1 RT Imb

if diff(LH, RH) > 1 → −1 → cur tree is Imbalanced

ret 1 + max(LH, RH) → ret height

(MVP)

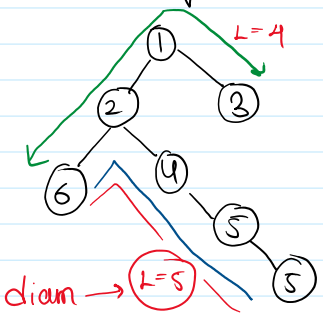
this returns

(1) max H : if tree is balanced

(2) −1 : if tree is imbalanced

DIAMETER OF BT ≡ WIDTH OF BINARY TREE

Diam : longest path in BT (might not pass via root)



Brute force → O(N²)

fn diam(N) {

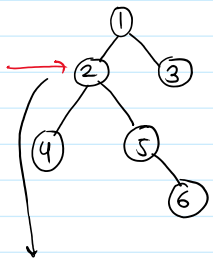
LH, RH ← Depth() repeated calls

dim = 1 + max(LH, RH)

}

OPTIMAL → CODE

→ ∀ node → u need Heights + Diam



max_d : max(max_d(L), max_d(R), LH + RH) → just pick a subtree; you can't join these or take max

max_h : 1 + max(LH, RH)

SOLVE(N) { → returns (max_d, max_h)

if N is None → (0, 0)

LD, LH ← solve(N.L)

RD, RH ← solve(N.R)

ret (max(LD, RD, LH + RH) → max diameter current

1 + MAX(LH, RH)) → max height current

}

* this (LH + RH) for (d) might change based on questions (+1, −2, −1)

Path - Sum - Count based

30 March 2024 05:14 PM

MAX PATH SUM IN BT