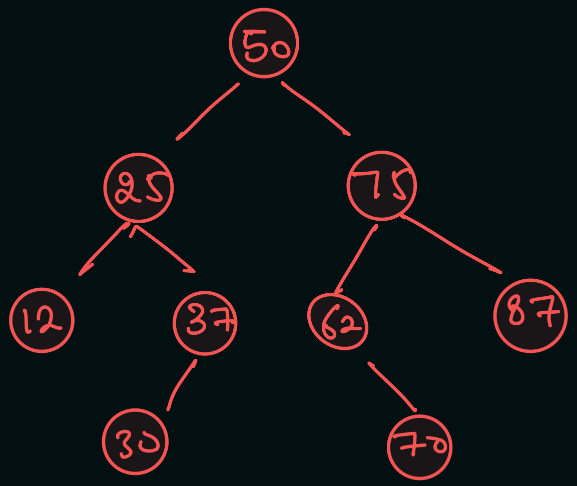
# Construct BST from Pre Order:→

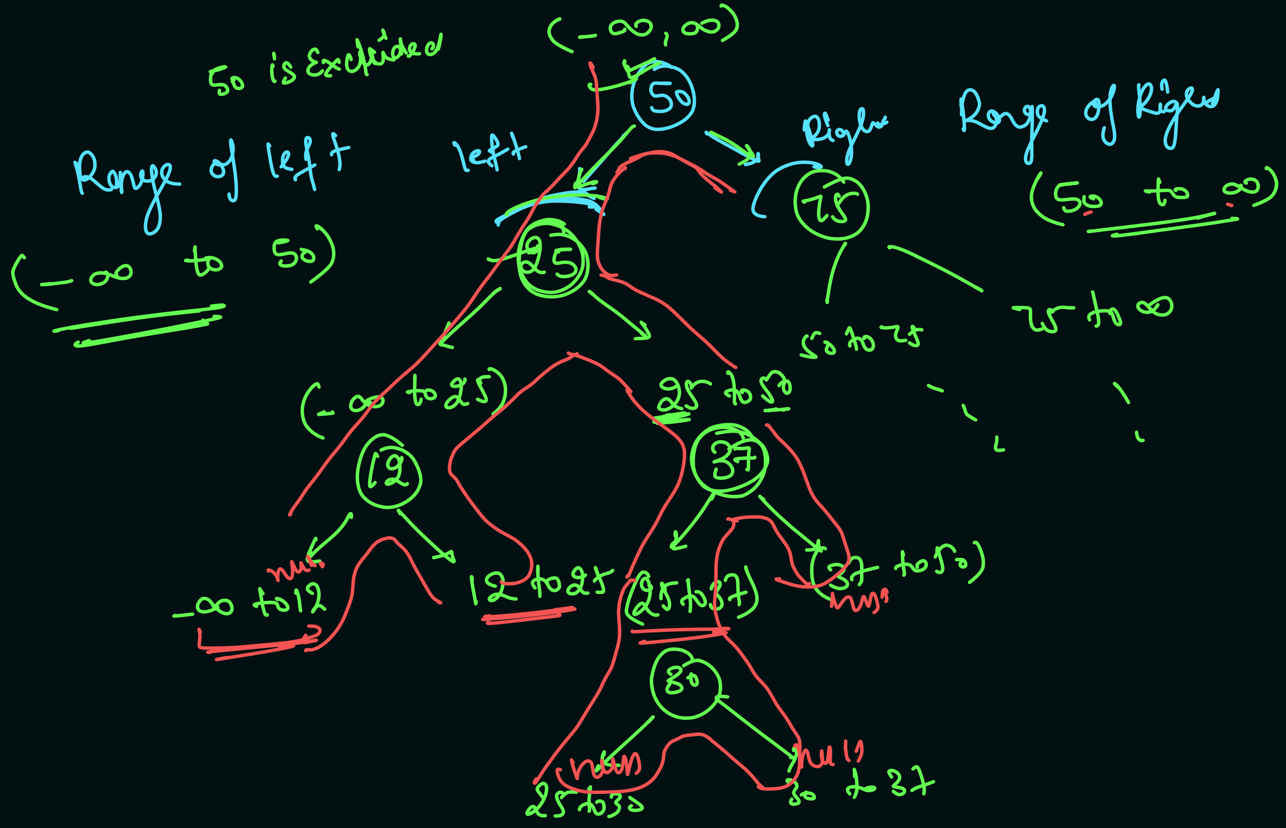
Root Left Right

PreOrder→ ~~50~~ 25 12 37 30 75 62 70 87

0 1 2 3 4 5 6 7 8

Indx

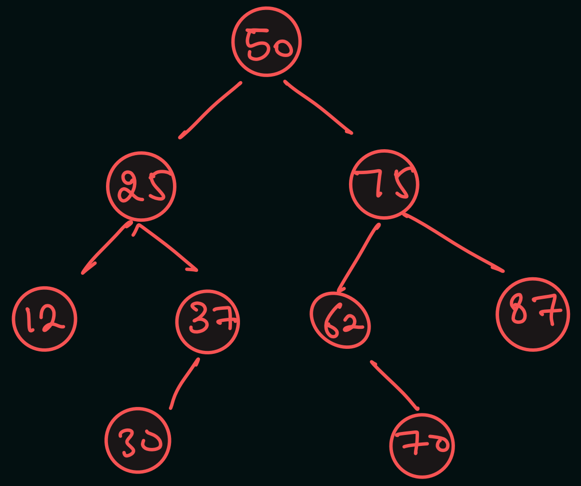


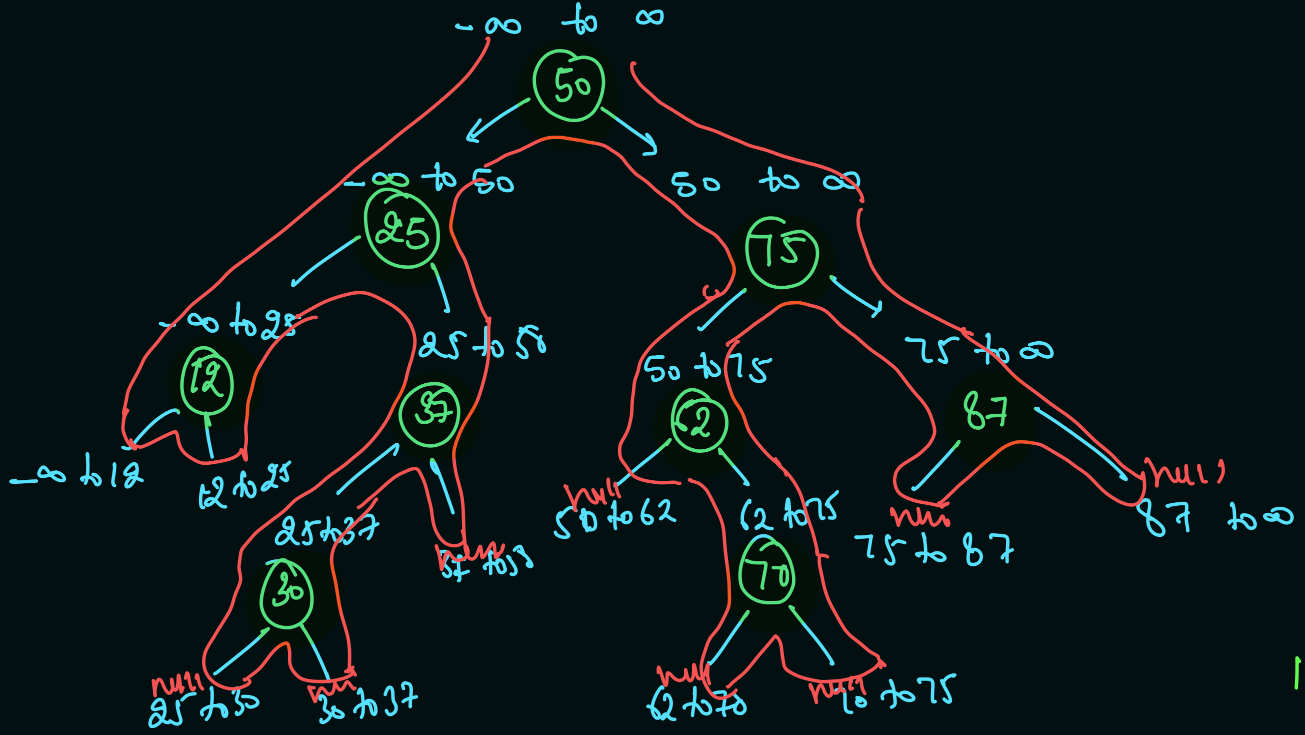


# Construct BST from Post Orders

Left Right Root

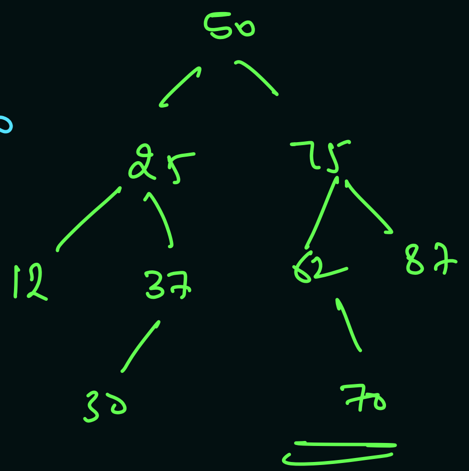
| Post Order → | 12 | 30 | 37 | 25 | 70 | 62 | 87 | 75 | 50 |
|---|---|---|---|---|---|---|---|---|---|
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| indx ⟸ | indx | indx | indx | indx | indx | indx | indx | indx | |





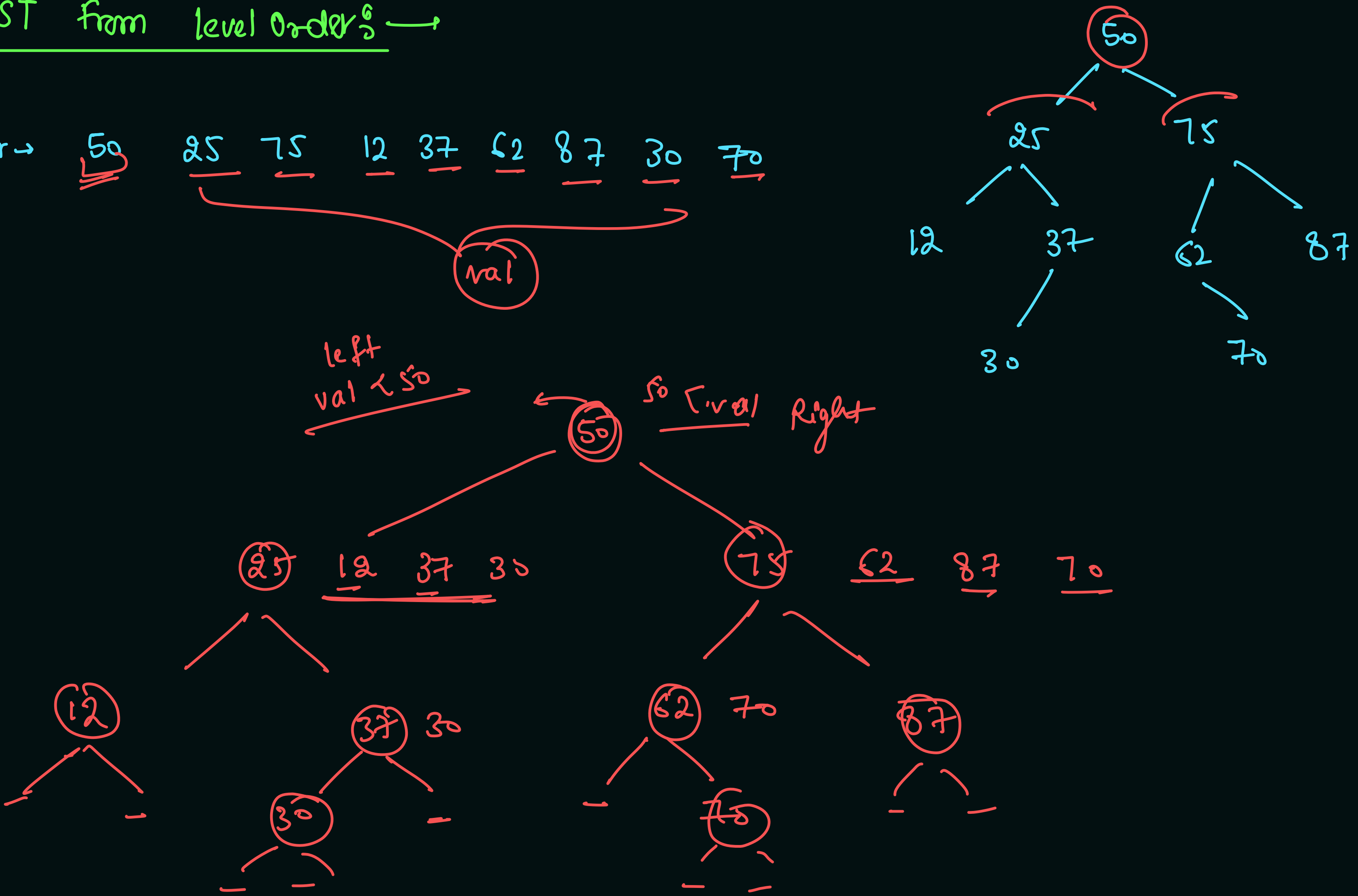
① Right Set

② left set



# Construct BST from level Order:→

level Order → 50 25 75 12 37 62 87 30 70



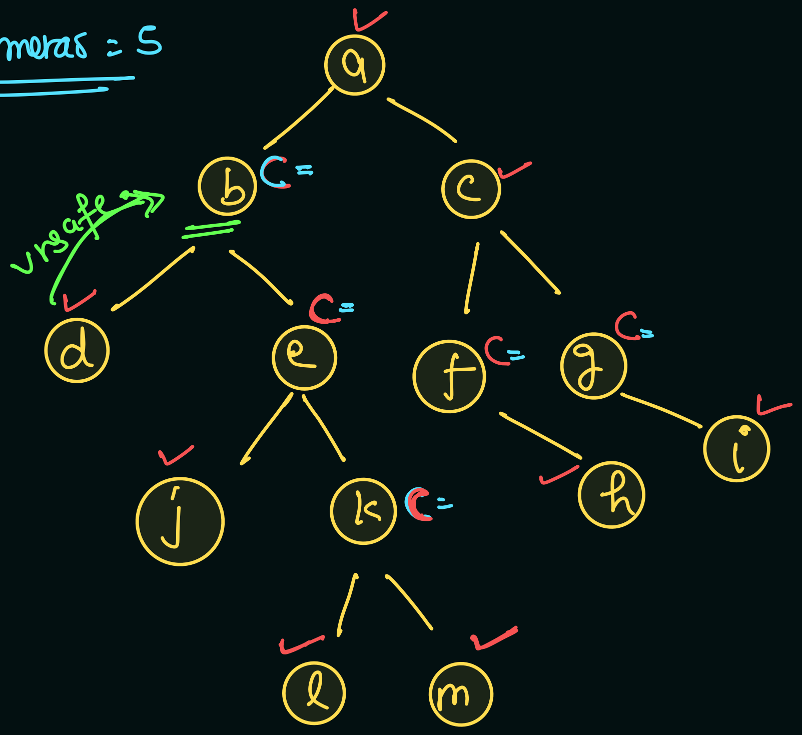
# Camera's In Binary Tree :

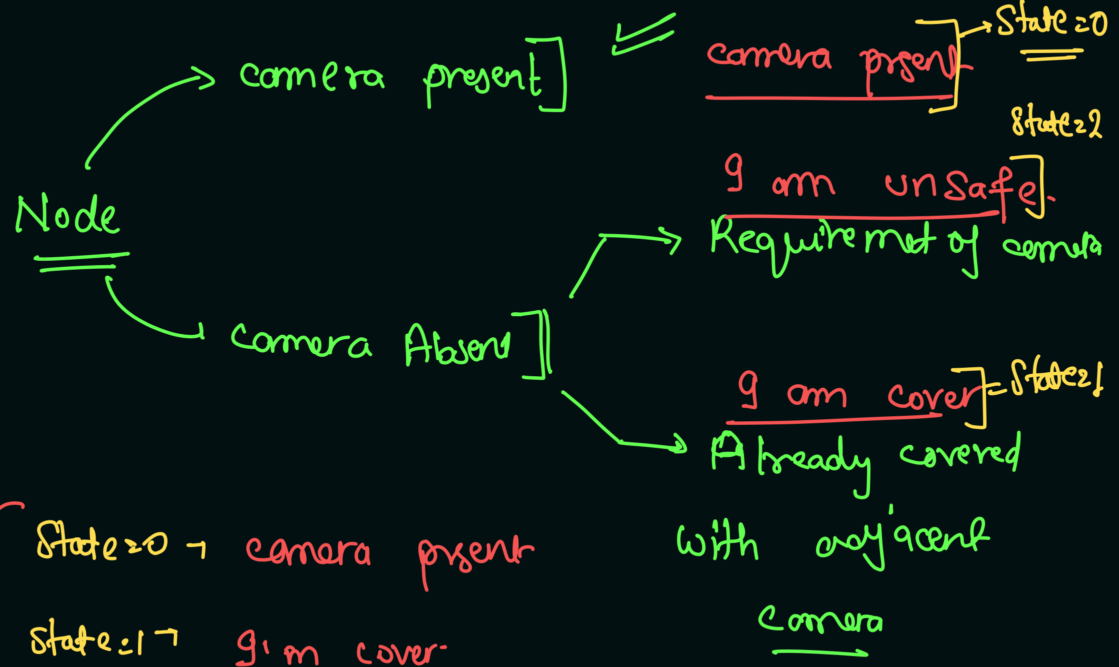
# Place cameras in binary tree ] → Min. Requirement

# camera can mouse eyes on children as well as on parent

# Cover all nodes of binary Tree

Min Cameras = 5



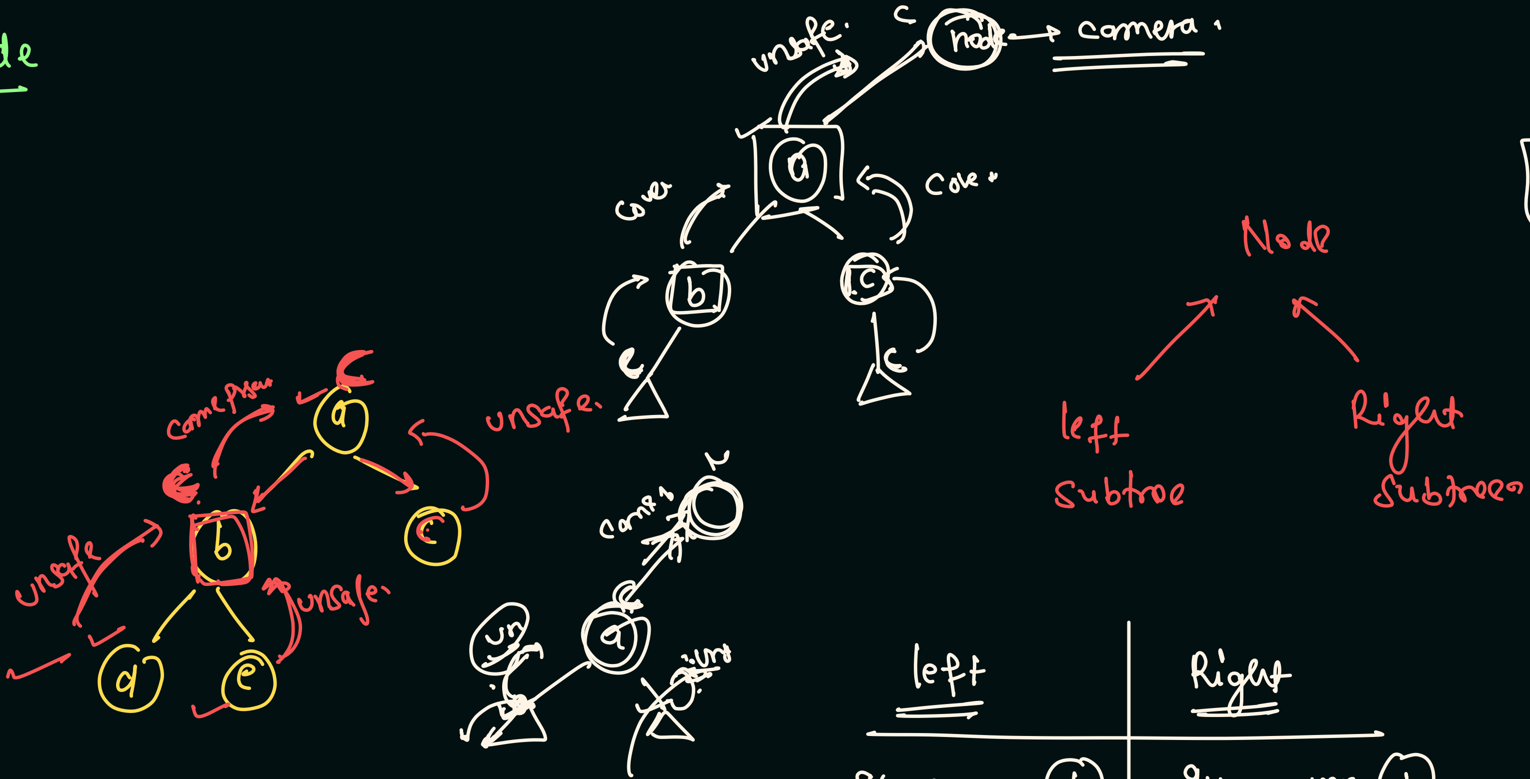


State = 0 → camera present

State = 1 → I'm cover

State = 2 → I'm unsafe

## Leaf Node



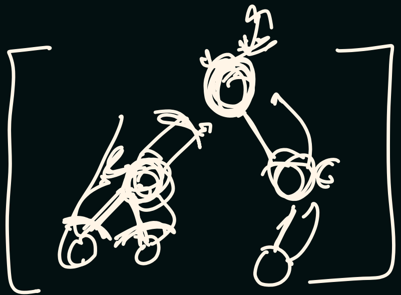
[Post Area :]

2 camera.

State=0 → camera present

State=1 → I'm cover.

State=2 → I'm unsafe



| left | Right |
|---|---|
| I'm cover 1 | I'm cover 1 |
| I'm unsafe 2 | I'm unsafe. 2 |

node return to its parent

→ Parent unsafe

↳ left == 1 && right == 1

└ return unsafe → 2

left == 2 || right == 2

left = t

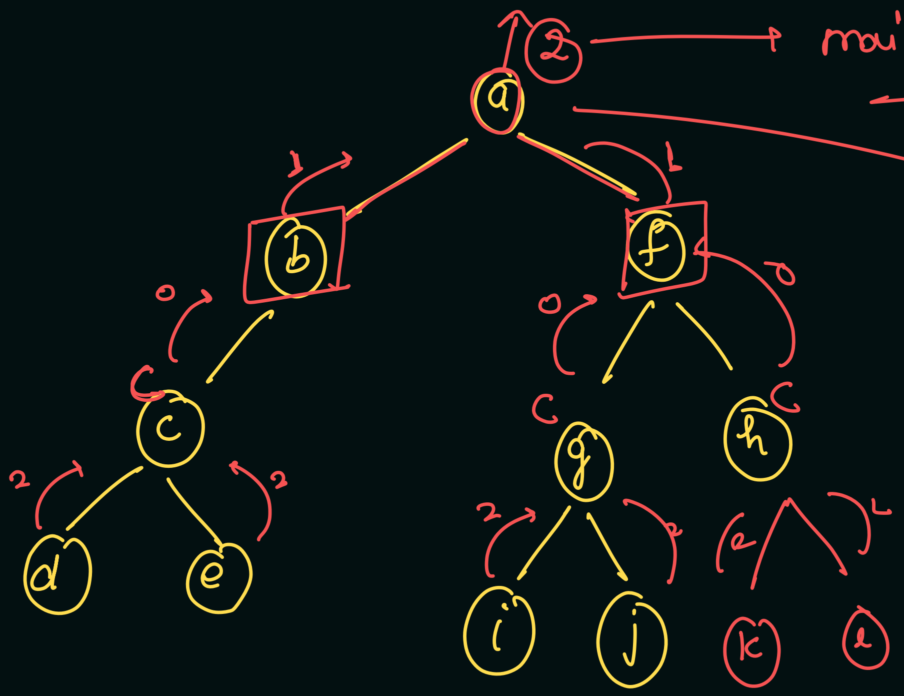
↳ camera fit!

→ return 0 Camera

left

Otherwise → If camera in any side.

→ return → 1 I'm covered!



→ main function ⊣ (2)

→ need to place a camera

level (1) → level (1)

[Successor and predeccor] state] → [Iterative DFS traversal]

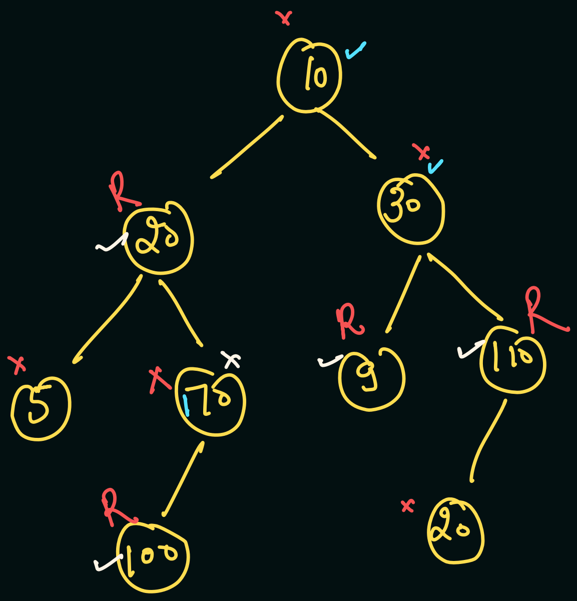
State of Child

# House Robber-III

data = money

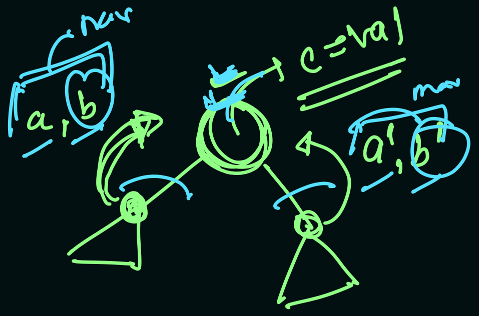
✓ ⊛ Robbery Such that max. money loot.

✓ ⊛ Adjacent House selection is not allowed



Max Amount = 230+9 = 239

discarded Solution ① Adjacent level select for Robbery ② select 'max of order] ×



Requirement of a node to make perfect Result

a: left (Robbery), b: left (without Robbery)

Right (Robbery), Right (without Robbery)

a' b'

Node → Robbery → c + b + b'

left & Right (No Robbery)

└ without Robbery ⇒

left a, b

Right a', b'

$(a + a')$ vs. $(a + b')$ vs $(b + a')$ vs $(b + b')$

max

conclusion ⇒ max(a, b) + max(a', b')