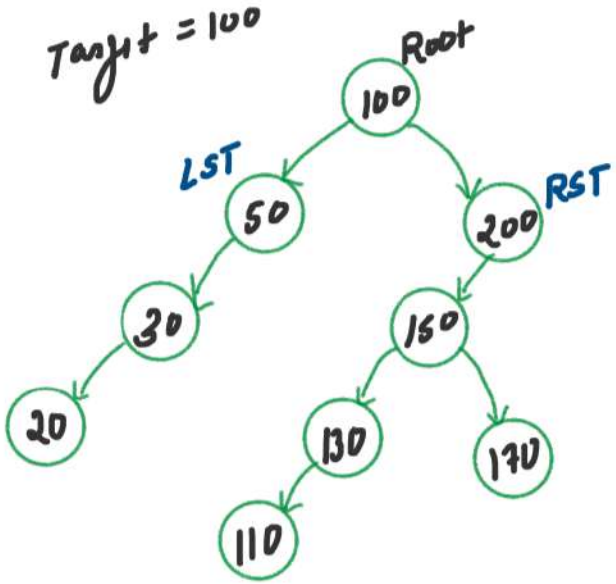
6. Delete Node from BST (Leetcode-450)

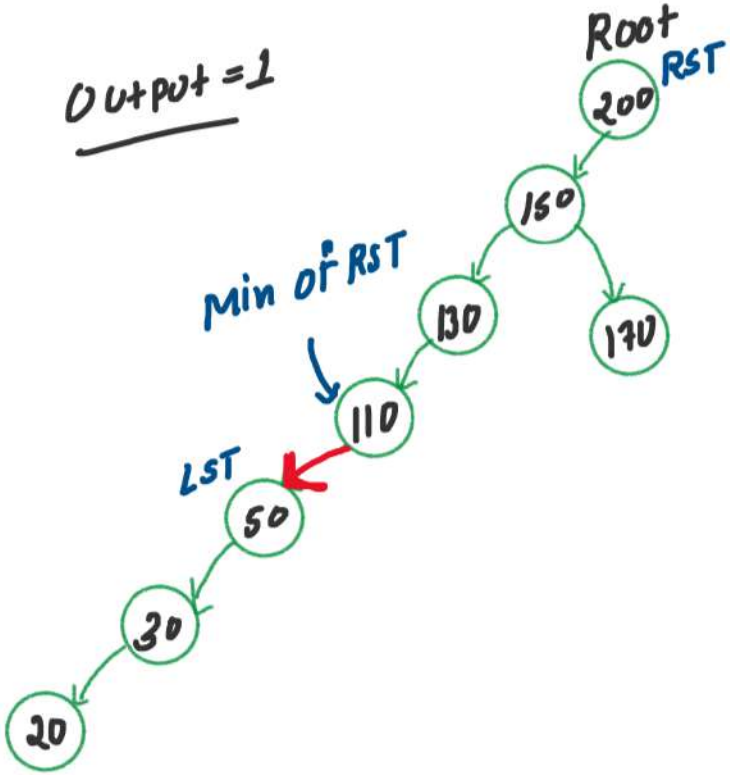
Ex Target = 100



[BST ⇒ LST < Root < RST]

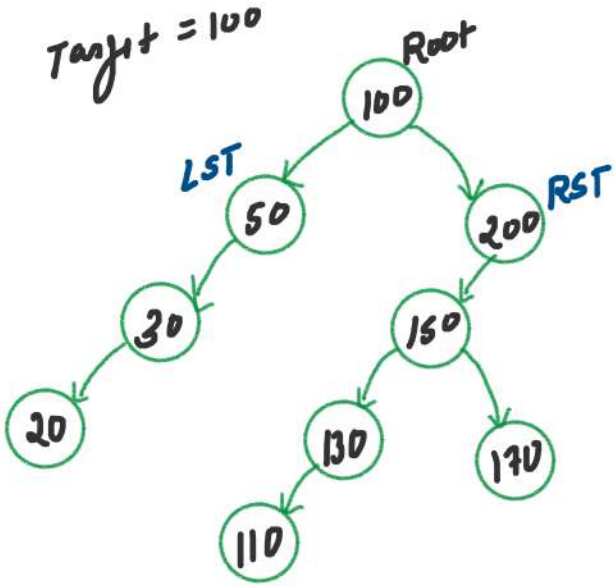
After deleting 100 ⇒ BST Nature maintain Rahe

Output = 1



EX

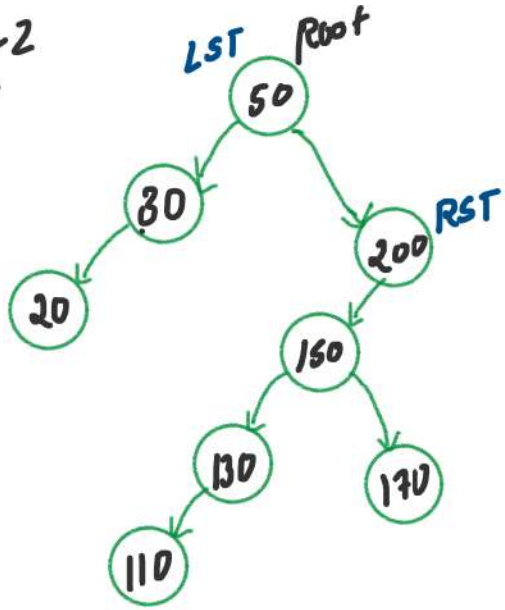
Target = 100



[BST ⇒ LST < Root < RST]

After deleting 100 ⇒ BST Nature maintain Rahe

Output-2



CASE1 Leaf Node

return Null

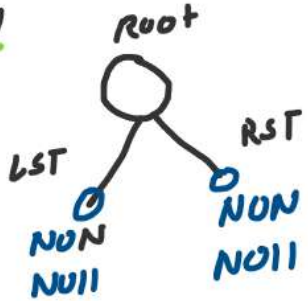
CASE2

return A

CASE3

return B

CASE4

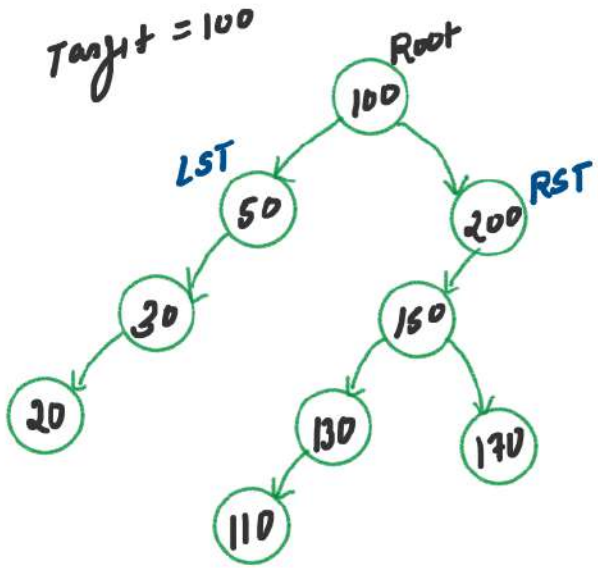


LST => Max Node

RST => MIN Node

DRY RUN

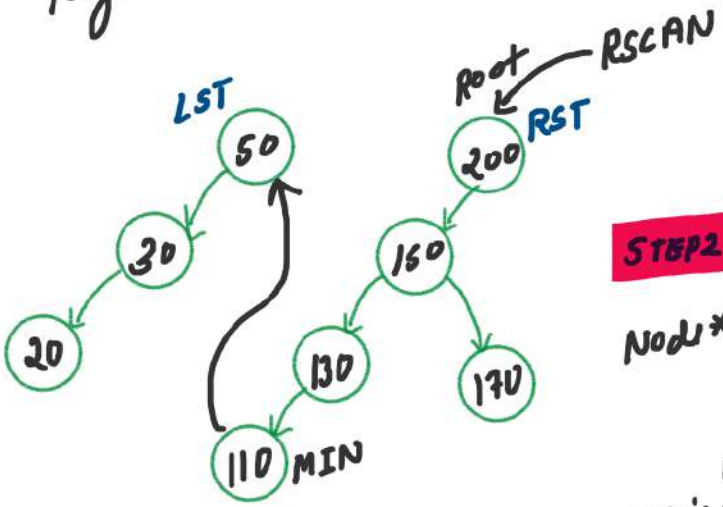
EX1 Target = 100



STEP 1

Find Target ⇒ **case 4**

Target = 100



STEP 2

Saved RST

```
Node* RSCAN = Root->right;
```

Find MIN OF RST

```
while (RSCAN->Left){
    RSCAN = RSCAN->Left;
}
```

Attached LST with MIN OF RST ka Left

```
RSCAN->Left = Root->Left;
```

Update the Root with RST

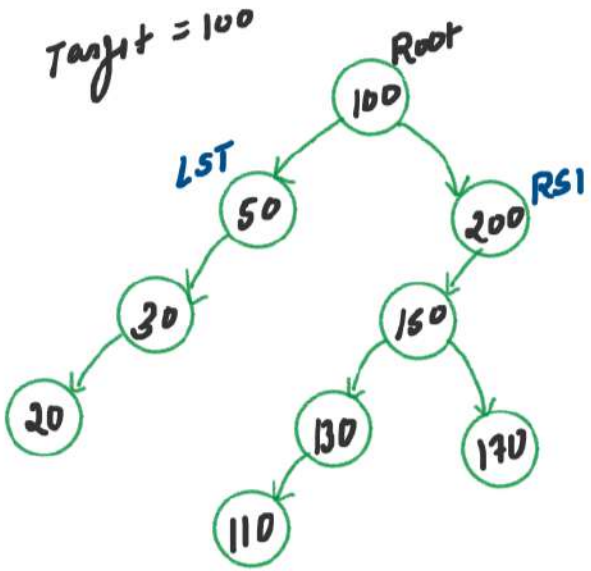
```
Node* Temp = Root;
root = Temp->right;
```

Delete the Target Node

```
delete Root;
return Root;
```

DRY RUN

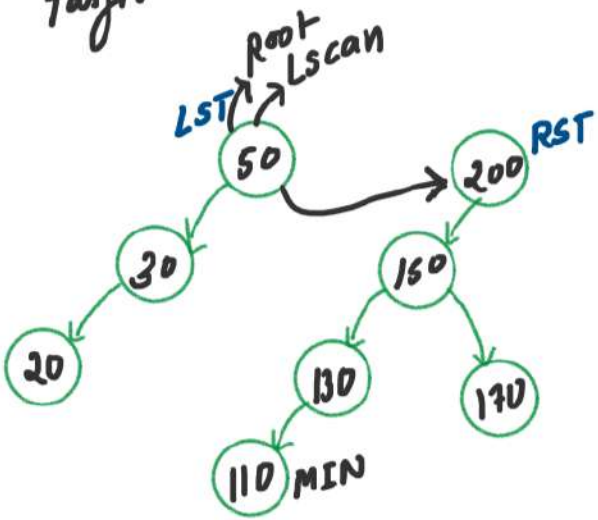
EX1 Target = 100



STEP1

Find Target ⇒ **Case 4**

Target = 100



STEP2

Saved LST

Node* LCAN = Root→ left;

Find MAX OF LST

```
while ( LCAN→Right ){
    LSCAN = LSCAN→Right;
}
```

Attached RST with MAX OF LST KA Right

LSCAN→right = Root→right;

Update the Root with LST

Node* Temp = Root;

root = Temp→Left;

Delete the Target Node

delete Root;

return Root;

```cpp
// 6. Delete Node From BST (Leetcode-450) with defferent approach

class Solution {
public:
    TreeNode* deleteNode(TreeNode* root, int key) {
        // Base case
        if(root == NULL) return NULL;

        // 1 case hum solve kar lenge
        if(root->val == key)
        {
          ...
        }
        // Ab recursion solve kar lega
        else if(root->val < key)
        {
            // Root ke right me chle jaoo
            root->right = deleteNode(root->right, key);
        }
        else
        {
            // Root ke left me chle jaoo
            root->left = deleteNode(root->left, key);
        }
        return root;
    }
};
```

```cpp
        // 1 case hum solve kar lenge
        if(root->val == key)
        {
            // Target root par v ho skta hai
            // Case 1:
            if(root->left == NULL && root->right == NULL){
                delete root;
                return NULL;
            }
            // Case 2:
            else if(root->left == NULL && root->right != NULL){
                auto temp = root;
                root = temp->right;
                delete temp;
                return root;
            }
            // Case 3:
            else if(root->left != NULL && root->right == NULL){
                auto temp = root;
                root = temp->left;
                delete temp;
                return root;
            }
            // Case 4: root->left != NULL && root->right != NULL
            else{

                // RST MIN NODE FIND KRLO
                ...

                // OR

                // LST MAX NODE FIND KRLO
                ...

            }
        }
```

```
// Case 4:
else(root->left != NULL && root->right != NULL)
{
    // LST MAX NODE FIND KRLO
    auto lscan = root->left;
    while(lscan->right){
        lscan = lscan->right;
    }
    lscan->right = root->right;
    auto temp = root;
    root = temp->left;
    delete temp;
    return root;
}
```

```
// Case 4:
else(root->left != NULL && root->right != NULL)
{
    // RST MIN NODE FIND KRLO
    auto rscan = root->right;
    while(rscan->left){
        rscan = rscan->left;
    }
    rscan->left = root->left;
    auto temp = root;
    root = temp->right;
    delete temp;
    return root;
}
```