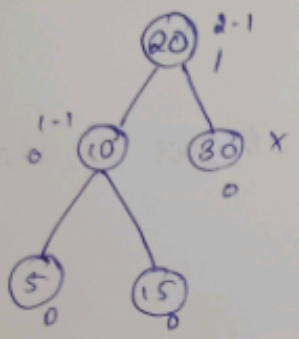
20/04/24

AVT - Tree.

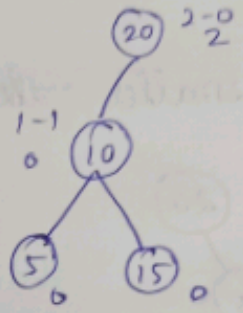
1- Deleting

Delete '30'

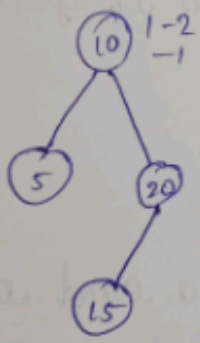


Balanced

⇒



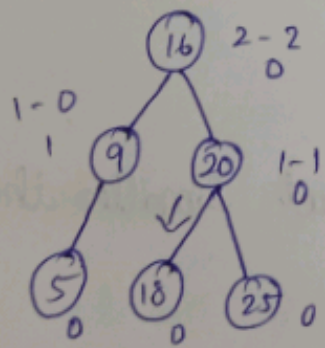
unBalancing.



Balanced.

Search:-

Search "18"



18 > 16 → Traverse right

18 < 20 → Traverse left.

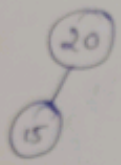
18 found.

1. Insert the following values in BST, 20, 25, 15, 12, 18, 45,

• Start with the root node as empty. Insert 20 as the root

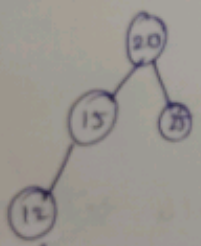
20

• 15 is smaller than 20, so it goes to the left of

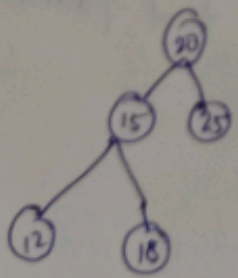


• 25 is larger than 20, So it goes to the right of

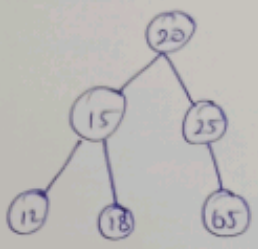
• 12 is smaller than 20 and also smaller than 15 So it goes to the left of 15.



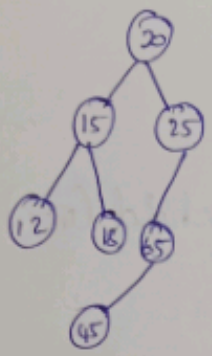
• 18 is larger than 15 but smaller than 20, So it goes to the right of 15.



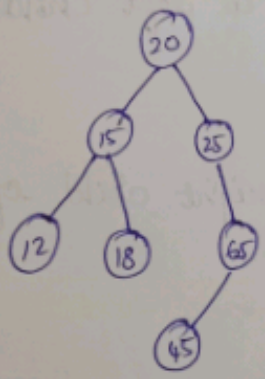
45 goes further to the right of 25



• 45 goes to the left of 65.



The final BST structure is :-



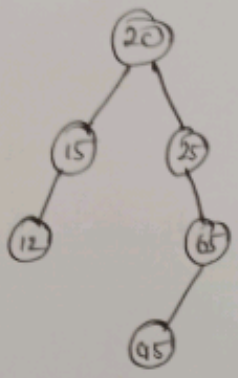
3 After deletion : delete the node no.

(i) 18 (ii) 65 (iii) 15.

Deleting Node : 18.

1. Node 18 is a leaf Node (has no children)
2. Delete Node 18.
3. Simply remove it from its Parents leaf child reference.

After deleting node 18,



Deleting Node: 65:-

1. Node 65 has a left child (45) but no right child.

2. Delete node 65.

3. Since node 65 has a left child (45), promote 45 to replace (65's) position.

4. Connect 45 as the right child of node 25 (65's Parent).

After deleting node: 65.

Deleting Node: 15 :-

1. Node 15 has two children (12 and 18).

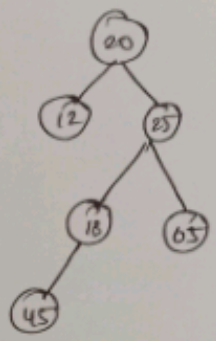
2. In this code, Find Largest node in the left

Subtree of 15 which is 12.

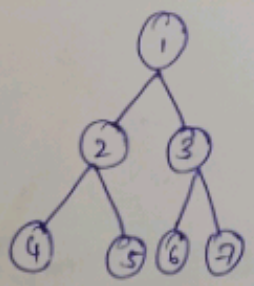
3. Replace node 15 with node 12.

4. Reconnect the subtree of node 15 its right child 18 to the left subtree of node 12.

After deleting node 15,



4 Binary Tree Traversal (In order, Pre order, Post order)

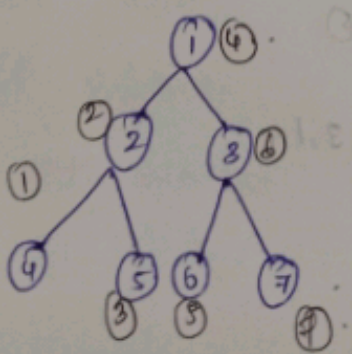


Inorder :-

Left → Root → Right.

- Visit the left subtree.
- Visit the Root subtree.
- Visit the Right subtree.

Inorder: 4, 2, 5, 1, 6, 3, 7.

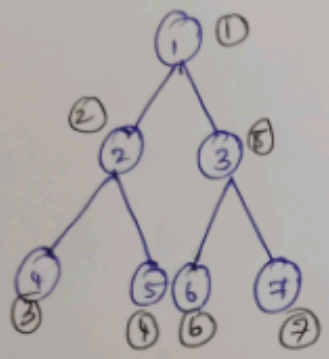


(ii)

Preorder:-

Root - left - Right .

- Visit the Root node.
- Visit the left subtree.
- Visit the right subtree.

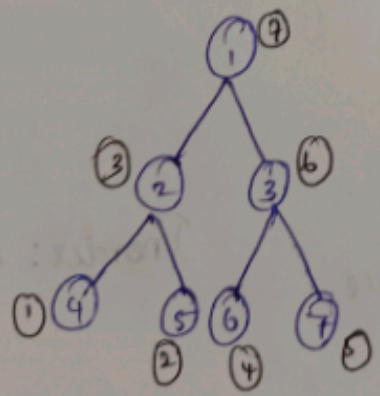


Preorder : 1, 2, 4, 5, 3, 6, 7.

Post order:-

left - Right - Root .

- Visit the left subtree.
- Visit the right subtree.
- Visit the root node.



Postorder : 4, 5, 2, 6, 7, 3, 1.

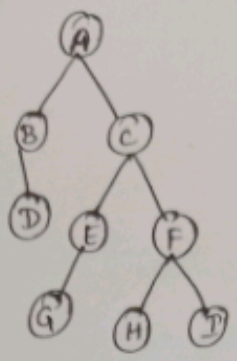
Inorder:-

- Vis
- Vi
- V

Preorde

- Vi
- V
- V

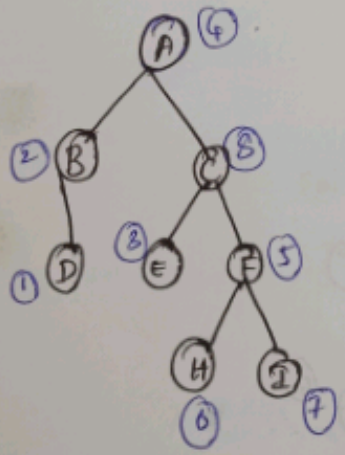
(ii)



Inorder:-

left → Root → Right.

- Visit the left subtree
- Visit the root node.
- Visit the right subtree.

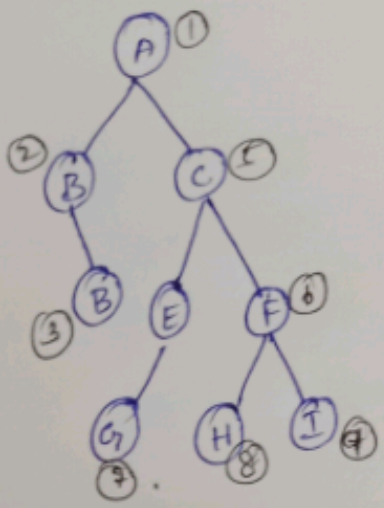


Inorder: D, B, E, A, F, H, I, C, G.

Preorder:-

Root → left → Right.

- Visit Root node.
- Visit left subtree.
- Visit Right subtree.

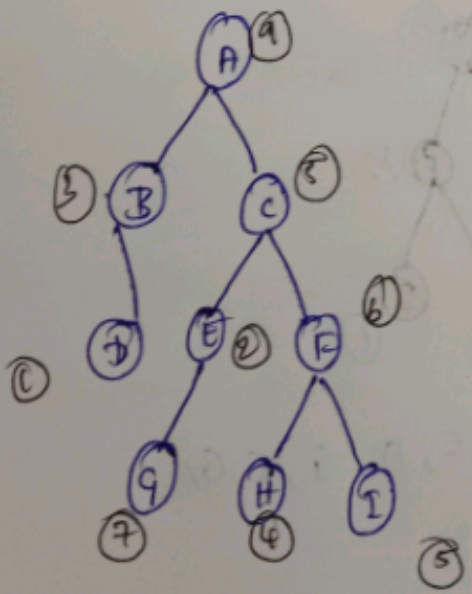


Preorder: A, B, D, E, C, F, G, H, I.

Postorder:-

Left - Right - Root.

- Visit left subtree.
- Visit Right subtree.
- Visit Rootnode.



Postorder: D, E, B, H, I, F, G, C, A.

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct AVLNode {
    int key;
    struct AVLNode* left;
    struct AVLNode* right;
    int height;
} AVLNode;
int height(AVLNode* node) {
    if (node == NULL) return 0;
    return node->height;
}
AVLNode* createNode(int key) {
    AVLNode* node = (AVLNode*)malloc(sizeof(AVLNode));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return node;
}
AVLNode* rightRotate(AVLNode* y) {
    AVLNode* x = y->left;
    AVLNode* T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = 1 + (height(y->left) > height(y->right) ? height(y->left) : height(y->right));
    x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) : height(x->right));

    return x;
}
AVLNode* leftRotate(AVLNode* x) {
    AVLNode* y = x->right;
    AVLNode* T2 = y->left;
    y->left = x;
    x->right = T2;
```

Output

Clear

```
/tmp/EyChDsR7yN.o
In-order traversal of the AVL tree is:
10 15 20 30

=== Code Execution Successful ===
```

main.c

Share | Run

```
    if (balance > 1 && key < node->left->key) {
        return rightRotate(node);
    }
    if (balance < -1 && key > node->right->key) {
        return leftRotate(node);
    }
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
void inOrder(AVLNode* root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d ", root->key);
        inOrder(root->right);
    }
}
int main() {
    AVLNode* root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 15);
    printf("In-order traversal of the AVL tree is:\n");
    inOrder(root);
    return 0;
}
```

Output | Clear

```
/tmp/EyChDsR7yN.o
In-order traversal of the AVL tree is:
10 15 20 30

=== Code Execution Successful ===
```

main.c

Share Run

```
    x->right = T2;
    x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) : height(x->right));
    y->height = 1 + (height(y->left) > height(y->right) ? height(y->left) : height(y->right));
    return y;
}
int getBalance(AVLNode* node) {
    if (node == NULL) return 0;
    return height(node->left) - height(node->right);
}
AVLNode* insert(AVLNode* node, int key) {
    if (node == NULL) return createNode(key);
    if (key < node->key) {
        node->left = insert(node->left, key);
    } else if (key > node->key) {
        node->right = insert(node->right, key);
    } else {
        return node;
    }
    node->height = 1 + (height(node->left) > height(node->right) ? height(node->left) : height(node
        ->right));
    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key) {
        return rightRotate(node);
    }
    if (balance < -1 && key > node->right->key) {
        return leftRotate(node);
    }
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
```

Output

Clear

```
/tmp/EyChDsR7yN.o
In-order traversal of the AVL tree is:
10 15 20 30

=== Code Execution Successful ===
```