

AMRITA VISHWA VIDYAPEETHAM  
CHENNAI CAMPUS

Second Year. B.TECH

(Computer Science and Engineering)

(DAA Lab Work)

Name: M CYNTHIA SHREE

RollNo.: CH.SC.U4CSE24110

Department: CSE-B

Academic Year: 2025-26

# 1. AVL tree

## CODE:

```
//CH.SC.U4CSE24110
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int key;
    int height;
    struct Node *right;
    struct Node *left;
};

int height(struct Node *n) {
    if (n==NULL)
        return 0;
    return n->height;
}

int max(int a, int b) {
    return (a>b) ? a:b;
}

struct Node* newNode(int key) {
    struct Node* node= (struct Node*)malloc(sizeof(struct Node));
    node->key=key;
    node->left=NULL;
    node->right=NULL;
    node->height=1;
    return(node);
}

struct Node *rightRotate(struct Node *y) {
    struct Node *x= y->left;
    struct Node *T2= x->right;
    x->right=y;
    y->left=T2;
    x->height= max(height(x->left),height(x->right))+1;
    y->height= max(height(y->left),height(y->right))+1;
    return x;
}
```

```

struct Node *leftRotate(struct Node *x) {
    struct Node *y= x->right;
    struct Node *T2= y->left;
    y->left=x;
    x->right=T2;
    x->height= max(height(x->left),height(x->right))+1;
    y->height= max(height(y->left),height(y->right))+1;
    return y;
}

int getBalance(struct Node *n) {
    if (n == NULL)
        return 0;
    return height(n->left) - height(n->right);
}

struct Node* insertNode(struct Node* node, int key) {
    if (node == NULL)
        return(newNode(key));
    if (key < node->key)
        node->left = insertNode(node->left, key);
    else if (key > node->key)
        node->right = insertNode(node->right, key);
    else
        return node;

    node->height = 1 + max(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);
    }
    return node;
}

```

```

struct Node * minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}

struct Node* deleteNode(struct Node* root, int key) {
    if (root == NULL)
        return root;
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);

    else {
        if((root->left == NULL) || (root->right == NULL)) {
            struct Node *temp = root->left ? root->left : root->right;
            if (temp == NULL) {
                temp = root;
                root = NULL;
            } else
                *root = *temp;
            free(temp);
        } else {
            struct Node* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }

    if (root == NULL)
        return root;
    root->height = 1 + max(height(root->left), height(root->right));
    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)
        return rightRotate(root);
    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}

```

```

void inOrder(struct Node *root) {
    if(root != NULL) {
        inOrder(root->left);
        printf("%d ", root->key);
        inOrder(root->right);
    }
}

int main() {
    struct Node *root = NULL;
    root = insertNode(root, 157);
    root = insertNode(root, 110);
    root = insertNode(root, 147);
    root = insertNode(root, 122);
    root = insertNode(root, 111);
    root = insertNode(root, 149);
    root = insertNode(root, 151);
    root = insertNode(root, 141);
    root = insertNode(root, 123);
    root = insertNode(root, 112);
    root = insertNode(root, 117);
    root = insertNode(root, 133);
    printf("Inorder traversal of the constructed AVL tree is:\n");
    inOrder(root);
    printf("\n");
    return 0;
}

```

## OUTPUT:

```

C:\Users\mrgns\Desktop\daa\task4>gcc avltree.c -o avltree

C:\Users\mrgns\Desktop\daa\task4>avltree
Inorder traversal of the constructed AVL tree is:
110 111 112 117 122 123 133 141 147 149 151 157

```

TIME COMPLEXITY:  $O(\log(n))$  because the tree is strictly balanced and it's height always remains  $\log n$ .

SPACE COMPLEXITY: There's no matrix or array so it is just  $O(n)$ .

## 2. Red Black tree

CODE:

```
//CH.SC.U4CSE24110
#include <stdio.h>
#include <stdlib.h>

enum Color { RED, BLACK };

struct Node {
    int data;
    enum Color color;
    struct Node *left, *right, *parent;
};

struct Node *root = NULL;

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->color = RED;
    node->left = node->right = node->parent = NULL;
    return node;
}
```

```

void leftRotate(struct Node **root, struct Node *x) {
    struct Node *y = x->right;
    x->right = y->left;
    if (y->left != NULL)
        y->left->parent = x;
    y->parent = x->parent;
    if (x->parent == NULL)
        *root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}

void rightRotate(struct Node **root, struct Node *y) {
    struct Node *x = y->left;
    y->left = x->right;
    if (x->right != NULL)
        x->right->parent = y;
    x->parent = y->parent;
    if (y->parent == NULL)
        *root = x;
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    x->right = y;
    y->parent = x;
}

```

```

void fixInsert(struct Node **root, struct Node *z) {
    while (z->parent != NULL && z->parent->color == RED) {
        if (z->parent == z->parent->parent->left) {
            struct Node *y = z->parent->parent->right;
            if (y != NULL && y->color == RED) {
                z->parent->color = BLACK;
                y->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            } else {
                if (z == z->parent->right) {
                    z = z->parent;
                    leftRotate(root, z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                rightRotate(root, z->parent->parent);
            }
        } else {
            struct Node *y = z->parent->parent->left;
            if (y != NULL && y->color == RED) {
                z->parent->color = BLACK;
                y->color = BLACK;
                z->parent->parent->color = RED;
                z = z->parent->parent;
            } else {
                if (z == z->parent->left) {
                    z = z->parent;
                    rightRotate(root, z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                leftRotate(root, z->parent->parent);
            }
        }
    }
    (*root)->color = BLACK;
}

```

```

void insert(struct Node **root, int data) {
    struct Node *z = newNode(data);
    struct Node *y = NULL;
    struct Node *x = *root;

    while (x != NULL) {
        y = x;
        if (z->data < x->data)
            x = x->left;
        else
            x = x->right;
    }

    z->parent = y;

    if (y == NULL)
        *root = z;
    else if (z->data < y->data)
        y->left = z;
    else
        y->right = z;

    fixInsert(root, z);
}

void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

int main() {
    int values[] = {157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133};
    int n = sizeof(values) / sizeof(values[0]);

    for (int i = 0; i < n; i++)
        insert(&root, values[i]);

    inorder(root);
    return 0;
}

```

## OUTPUT:

```

C:\Users\mrgns\Desktop\daa\task4>gcc redblacktree.c -o redblacktree
C:\Users\mrgns\Desktop\daa\task4>redblacktree
110 111 112 117 122 123 133 141 147 149 151 157

```

TIME COMPLEXITY:  $O(\log(n))$  because the tree's height is at most  $2^{\log n}$ .

SPACE COMPLEXITY: There's no matrix or array so it is just  $O(n)$ .