



AMRITA
VISHWA VIDYAPEETHAM

Second Year B.Tech

(Computer Science and Engineering)

Design and Analysis of Algorithms

TASK – 4

Name: Shaik Kolimi Dada Hussain

College Name: Amrita vishwa Vidyapeetham

Roll No : ch.sc.u4cse24144

Department: CSE-B

Academic Year : 2024-2028

AVL TREE :

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int key;
    struct Node *left, *right;
    int height;
};
int max(int a, int b) {
    return (a > b) ? a : b;
}
int getHeight(struct Node *n) {
    return (n == NULL) ? 0 : n->height;
}
struct Node* newNode(int key) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = 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;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left), getHeight(x->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(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    return y;
}

struct Node* balanceTree(struct Node* node) {
    if (node == NULL)
        return NULL;
    node->left = balanceTree(node->left);
    node->right = balanceTree(node->right);

    node->height = 1 + max(getHeight(node->left), getHeight(node->right));

    int balance = getHeight(node->left) - getHeight(node->right);

    if (balance > 1 && getHeight(node->left->left) >= getHeight(node->left->right))
        return rightRotate(node);

    if (balance > 1 && getHeight(node->left->left) < getHeight(node->left->right)) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && getHeight(node->right->right) >= getHeight(node->right->left))
        return leftRotate(node);

    if (balance < -1 && getHeight(node->right->right) < getHeight(node->right->left)) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

void printGivenLevel(struct Node* root, int level) {
    if (root == NULL)
        return;

    if (level == 1)
        printf("%d ", root->key);
    else {
        printGivenLevel(root->left, level - 1);
        printGivenLevel(root->right, level - 1);
    }
}
```

```

void printLevelOrder(struct Node* root) {
    int h = getHeight(root);
    for (int i = 1; i <= h; i++)
        printGivenLevel(root, i);
}

int main() {
    printf("CH.SC.U4CSE24144\n");

    struct Node *root = newNode(157);
    root->left = newNode(110);
    root->left->right = newNode(147);
    root->left->right->left = newNode(122);
    root->left->right->right = newNode(149);
    root->left->right->right->right = newNode(151);
    root->left->right->left->left = newNode(111);
    root->left->right->left->right = newNode(141);
    root->left->right->left->right = newNode(112);
    root->left->right->left->right->left = newNode(123);
    root->left->right->left->right->left->right = newNode(133);
    root->left->right->left->right->left->left = newNode(117);

    root = balanceTree(root);

    printf("AVL Level Order: ");
    printLevelOrder(root);
    printf("\n");

    return 0;
}

```

```

dada@Ubuntu:~/Desktop$ ./a.out
CH.SC.U4CSE24144
AVL Level Order: 123 110 157 122 147 111 117 141 149 112 133 151
dada@Ubuntu:~$ 

```

TIME AND SPACE COMPLEXITY:

Time Complexity:

O(n)

Reason: Each node in the tree is visited once while balancing and printing.

Space Complexity:

O(n)

Reason: Memory is needed to store all n nodes in the tree (recursion uses only small extra space).

RED BLACK TREE:

```
//CH.SC.U4CSE24144
#include <stdio.h>
#include <stdlib.h>
enum Color { RED, BLACK };
struct Node {
    int data;
    enum Color color;
    struct Node *left, *right, *parent;
};
struct Node* newNode(int data) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = data;
    temp->left = temp->right = temp->parent = NULL;
    temp->color = RED;
    return temp;
}
void rotateLeft(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 rotateRight(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 fixViolation(struct Node** root, struct Node* z) {
    while (z != *root && 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;
                    rotateLeft(root, z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                rotateRight(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;
                    rotateRight(root, z);
                }
                z->parent->color = BLACK;
                z->parent->parent->color = RED;
                rotateLeft(root, z->parent->parent);
            }
        }
    }
}
```

```

        }
        (*root)->color = BLACK;
    }

    int height(struct Node* node) {
        if (node == NULL)
            return 0;
        int l = height(node->left);
        int r = height(node->right);
        return (l > r ? l : r) + 1;
    }

    void printGivenLevel(struct Node* root, int level) {
        if (root == NULL)
            return;

        if (level == 1)
            printf("%d ", root->data);
        else {
            printGivenLevel(root->left, level - 1);
            printGivenLevel(root->right, level - 1);
        }
    }

    void printLevelOrder(struct Node* root) {
        int h = height(root);
        for (int i = 1; i <= h; i++) {
            printGivenLevel(root, i);
            printf(" | ");
        }
    }

    int main() {
        struct Node* root = newNode(157);
        root->color = BLACK;

        struct Node* n110 = newNode(110);
        root->left = n110; n110->parent = root;

        struct Node* n147 = newNode(147);
        n110->right = n147; n147->parent = n110;

        struct Node* n122 = newNode(122);

```

```

        struct Node* n122 = newNode(122);
        n147->left = n122; n122->parent = n147;

        struct Node* n149 = newNode(149);
        n147->right = n149; n149->parent = n147;

        struct Node* n111 = newNode(111);
        n122->left = n111; n111->parent = n122;
        struct Node* n141 = newNode(141);
        n122->right = n141; n141->parent = n122;
        struct Node* n151 = newNode(151);
        n149->right = n151; n151->parent = n149;
        struct Node* n112 = newNode(112);
        n111->right = n112; n112->parent = n111;
        struct Node* n123 = newNode(123);
        n141->left = n123; n123->parent = n141;
        struct Node* n117 = newNode(117);
        n112->left = n117; n117->parent = n112;
        struct Node* n133 = newNode(133);
        n123->right = n133; n133->parent = n123;
        fixViolation(&root, n147);
        fixViolation(&root, n111);
        fixViolation(&root, n141);
        fixViolation(&root, n151);
        fixViolation(&root, n117);
        fixViolation(&root, n133);

        printf("Red-Black Level Order: ");
        printLevelOrder(root);
        printf("\n");
        printf("CH.SC.U4CSE24144\n");
        return 0;
    }
}

```

```
Red-Black Level Order: 122 | 111 147 | 110 112 133 151 | 117 123 141 149 157 |
```

```
CH. SEC.U4CSE24144
```

```
dada@Ubuntu:~$
```

TIME AND SPACE COMPLEXITY:

Time Complexity:

O(n)

Reason: All nodes are processed while fixing violations and printing the tree.

Space Complexity:

O(n)

Reason: Memory is required to store all n nodes (recursion uses only small extra space).