**DATASTRUCTURE**

**PROGRAMS:**

**1.Red Black Tree**

*#include <stdio.h>*

*#include <stdlib.h>*

*typedef enum { RED, BLACK } NodeColor;*

*typedef struct RBTreeNode {*

*int data;*

*NodeColor color;*

*struct RBTreeNode \*left, \*right, \*parent;*

*} RBTreeNode;*

*RBTreeNode\* createNode(int data) {*

*RBTreeNode\* newNode = (RBTreeNode\*)malloc(sizeof(RBTreeNode));*

*newNode->data = data;*

*newNode->color = RED;*

*newNode->left = newNode->right = newNode->parent = NULL;*

*return newNode;*

*}*

*void rotateLeft(RBTreeNode\*\* root, RBTreeNode\* x);*

*void rotateRight(RBTreeNode\*\* root, RBTreeNode\* y);*

*void fixViolation(RBTreeNode\*\* root, RBTreeNode\* z);*

*void insert(RBTreeNode\*\* root, int data);*

*void inorderTraversal(RBTreeNode\* root);*

*void rotateLeft(RBTreeNode\*\* root, RBTreeNode\* x) {*

*RBTreeNode\* 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(RBTreeNode\*\* root, RBTreeNode\* y) {*

*RBTreeNode\* 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(RBTreeNode\*\* root, RBTreeNode\* z) {*

*while (z != \*root && z->parent->color == RED) {*

*RBTreeNode\* y;*

*if (z->parent == z->parent->parent->left) {*

*y = z->parent->parent->right;*

*if (y && 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 {*

*y = z->parent->parent->left;*

*if (y && 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;*

*}*

*void insert(RBTreeNode\*\* root, int data) {*

*RBTreeNode\* z = createNode(data);*

*RBTreeNode\* y = NULL;*

*RBTreeNode\* 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;*

*fixViolation(root, z);*

*}*

*void inorderTraversal(RBTreeNode\* root) {*

*if (root == NULL)*

*return;*

*inorderTraversal(root->left);*

*printf("%d ", root->data);*

*inorderTraversal(root->right);*

*}*

*int main() {*

*RBTreeNode\* root = NULL;*

*insert(&root, 10);*

*insert(&root, 20);*

*insert(&root, 30);*

*insert(&root, 15);*

*insert(&root, 25);*

*insert(&root, 5);*

*printf("In-order Traversal of Created Tree:\n");*

*inorderTraversal(root);*

*return 0;*

*}*

**OUTPUT:**

*In-order Traversal of Created Tree:*

*5 10 15 20 25 30*

**2.Splay Tree**

*#include <stdio.h>*

*#include <stdlib.h>*

*struct Node {*

*int key;*

*struct Node\* left;*

*struct Node\* right;*

*};*

*struct Node\* newNode(int key) {*

*struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));*

*node->key = key;*

*node->left = node->right = NULL;*

*return node;*

*}*

*struct Node\* rightRotate(struct Node\* x) {*

*struct Node\* y = x->left;*

*x->left = y->right;*

*y->right = x;*

*return y;*

*}*

*struct Node\* leftRotate(struct Node\* x) {*

*struct Node\* y = x->right;*

*x->right = y->left;*

*y->left = x;*

*return y;*

*}*

*struct Node\* splay(struct Node\* root, int key) {*

*if (root == NULL || root->key == key)*

*return root;*

*if (root->key > key) {*

*if (root->left == NULL) return root;*

*if (root->left->key > key) {*

*root->left->left = splay(root->left->left, key);*

*root = rightRotate(root);*

*}*

*else if (root->left->key < key) {*

*root->left->right = splay(root->left->right, key);*

*if (root->left->right != NULL)*

*root->left = leftRotate(root->left);*

*}*

*return (root->left == NULL) ? root : rightRotate(root);*

*} else*

*if (root->right == NULL) return root;*

*if (root->right->key > key) {*

*root->right->left = splay(root->right->left, key);*

*if (root->right->left != NULL)*

*root->right = rightRotate(root->right);*

*}*

*else if (root->right->key < key) {*

*root->right->right = splay(root->right->right, key);*

*root = leftRotate(root);*

*}*

*return (root->right == NULL) ? root : leftRotate(root);*

*}*

*}*

*struct Node\* insert(struct Node\* root, int key) {*

*if (root == NULL) return newNode(key);*

*root = splay(root, key);*

*if (root->key == key) return root;*

*struct Node\* newnode = newNode(key);*

*if (root->key > key) {*

*newnode->right = root;*

*newnode->left = root->left;*

*root->left = NULL;*

*} else {*

*newnode->left = root;*

*newnode->right = root->right;*

*root->right = NULL;*

*}*

*return newnode;*

*}*

*void preOrder(struct Node\* root) {*

*if (root != NULL) {*

*printf("%d ", root->key);*

*preOrder(root->left);*

*preOrder(root->right);*

*}*

*}*

*int main() {*

*struct Node\* root = NULL;*

*root = insert(root, 100);*

*root = insert(root, 50);*

*root = insert(root, 200);*

*root = insert(root, 40);*

*root = insert(root, 30);*

*root = insert(root, 20);*

*printf("Preorder traversal of the splay tree is \n");*

*preOrder(root);*

*return 0;*

*}*

**OUTPUT:**

*Preorder traversal of the splay tree is*

*20 30 40 50 100 200*