DATA STRUCTURE

DAY 8 – 03/08/2024

1.red black tree

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define RED 0
#define BLACK 1
 typedef struct Node {
    int data;
    int color;
    struct Node *left, *right, *parent;
} Node;
 Node* createNode(int data) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->color = RED; // New nodes are initially red
    node->left = node->right = node->parent = NULL;
    return node;
}
 void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d(%s) ", root->data, root->color == RED ? "RED" : "BLACK");
        inorderTraversal(root->right);
    }
}
void leftRotate(Node** root, Node* x) {
    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(Node** root, Node* y) {
    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->right)
        y->parent->right = x;
    else
        y->parent->left = x;
    x->right = y;
    y->parent = x;
}
 void fixInsert(Node** root, Node* node) {
```

```
Node* parent = NULL;
Node* grandparent = NULL;
while (node != *root && node->parent->color == RED) {
    parent = node->parent;
    grandparent = parent->parent;
    if (parent == grandparent->left) {
        Node* uncle = grandparent->right;
        if (uncle != NULL && uncle->color == RED) {
            parent->color = BLACK;
            uncle->color = BLACK;
            grandparent->color = RED;
            node = grandparent;
        } else {
            if (node == parent->right) {
                node = parent;
                leftRotate(root, node);
            }
            parent->color = BLACK;
            grandparent->color = RED;
            rightRotate(root, grandparent);
        }
    } else {
        Node* uncle = grandparent->left;
        if (uncle != NULL && uncle->color == RED) {
            parent->color = BLACK;
            uncle->color = BLACK;
            grandparent->color = RED;
            node = grandparent;
        } else {
                if (node == parent->left) {
                    node = parent;
                    rightRotate(root, node);
                }
                parent->color = BLACK;
                grandparent->color = RED;
                leftRotate(root, grandparent);
            }
        }
    }
    (*root)->color = BLACK;
}
void insert(Node** root, int data) {
    Node* node = createNode(data);
    Node* parent = NULL;
    Node* current = *root;
    while (current != NULL) {
        parent = current;
        if (node->data < current->data)
            current = current->left;
        else
            current = current->right;
    }
    node->parent = parent;
    if (parent == NULL)
        *root = node;
    else if (node->data < parent->data)
        parent->left = node;
    else
```

```
        parent->right = node;
    fixInsert(root, node);
}

int main() {
    Node* root = NULL;
    insert(&root, 10);
    insert(&root, 20);
    insert(&root, 30);
    insert(&root, 15);
    insert(&root, 25);
    printf("In-order Traversal of Red-Black Tree:\n");
    inorderTraversal(root);
    return 0;
}
```

OUTPUT:

```
In-order Traversal of Red-Black Tree:
10(BLACK) 15(RED) 20(BLACK) 25(RED) 30(BLACK)
```

2.splay tree

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;
 Node* createNode(int data) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}
Node* rightRotate(Node* root) {
    Node* newRoot = root->left;
    root->left = newRoot->right;
    newRoot->right = root;
    return newRoot;
}
Node* leftRotate(Node* root) {
    Node* newRoot = root->right;
    root->right = newRoot->left;
    newRoot->left = root;
    return newRoot;
}
 Node* splay(Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (key < root->data) {
        if (root->left == NULL) return root;
        if (key < root->left->data) {
            root->left->left = splay(root->left->left, key);
            root = rightRotate(root);
        } else if (key > root->left->data) {
            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 (key > root->right->data) {
            root->right->right = splay(root->right->right, key);
            root = leftRotate(root);
        } else if (key < root->right->data) {
            root->right->left = splay(root->right->left, key);
            if (root->right->left != NULL)
                root->right = rightRotate(root->right);
        }
        return (root->right == NULL) ? root : leftRotate(root);
    }
}
Node* insert(Node* root, int key) {
    if (root == NULL) return createNode(key);
    root = splay(root, key);
    if (root->data == key) return root;
    Node* newNode = createNode(key);
    if (key < root->data) {
        newNode->right = root;
        newNode->left = root->left;
        root->left = NULL;
    } else {
        newNode->left = root;
        newNode->right = root->right;
        root->right = NULL;
    }
    return newNode;
}
 Node* search(Node* root, int key) {
    return splay(root, key);
}
 void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
int main() {
    Node* root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 15);
    root = insert(root, 25);
    printf("In-order Traversal of Splay Tree:\n");
    inorderTraversal(root);
    printf("\n");
    Node* result = search(root, 15);
    printf("Searched for 15, now the root is: %d\n", result->data);
    return 0;
}
```

OUTPUT:

```
In-order Traversal of Splay Tree:
```

```
10 15 20 25 30

Searched for 15, now the root is: 15
```