Kartheeka.R

192372289

CSE_AI

03/08/2024

RED-BLACK TREE:

```
#include <stdio.h>
#include <stdlib.h>

typedef enum { RED, BLACK } Color;

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

typedef struct RedBlackTree {
    Node* root;
    Node* TNULL;
} RedBlackTree;

Node* createNode(int data, Node* TNULL) {
    Node* node = (Node*)malloc(sizeof(Node));
```

```
    node->data = data;
    node->parent = NULL;
    node->left = TNULL;
    node->right = TNULL;
    node->color = RED;
    return node;
}

void leftRotate(RedBlackTree* tree, Node* x) {
    Node* y = x->right;
    x->right = y->left;
    if (y->left != tree->TNULL) {
        y->left->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == NULL) {
        tree->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(RedBlackTree* tree, Node* x) {
```

```
    Node* y = x->left;
    x->left = y->right;
    if (y->right != tree->TNULL) {
        y->right->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == NULL) {
        tree->root = y;
    } else if (x == x->parent->right) {
        x->parent->right = y;
    } else {
        x->parent->left = y;
    }
    y->right = x;
    x->parent = y;
}

void fixInsert(RedBlackTree* tree, Node* k) {
    Node* u;
    while (k->parent->color == RED) {
        if (k->parent == k->parent->parent->right) {
            u = k->parent->parent->left;
            if (u->color == RED) {
                u->color = BLACK;
                k->parent->color = BLACK;
                k->parent->parent->color = RED;
                k = k->parent->parent;
```

```
    } else {
        if (k == k->parent->left) {
            k = k->parent;
            rightRotate(tree, k);
        }
        k->parent->color = BLACK;
        k->parent->parent->color = RED;
        leftRotate(tree, k->parent->parent);
    }
} else {
    u = k->parent->parent->right;
    if (u->color == RED) {
        u->color = BLACK;
        k->parent->color = BLACK;
        k->parent->parent->color = RED;
        k = k->parent->parent;
    } else {
        if (k == k->parent->right) {
            k = k->parent;
            leftRotate(tree, k);
        }
        k->parent->color = BLACK;
        k->parent->parent->color = RED;
        rightRotate(tree, k->parent->parent);
    }
}
if (k == tree->root) {
```

```
            break;
        }
    }
    tree->root->color = BLACK;
}

void insert(RedBlackTree* tree, int key) {
    Node* node = createNode(key, tree->TNULL);
    Node* y = NULL;
    Node* x = tree->root;

    while (x != tree->TNULL) {
        y = x;
        if (node->data < x->data) {
            x = x->left;
        } else {
            x = x->right;
        }
    }

    node->parent = y;
    if (y == NULL) {
        tree->root = node;
    } else if (node->data < y->data) {
        y->left = node;
    } else {
        y->right = node;
```

```
    }

    if (node->parent == NULL) {
        node->color = BLACK;
        return;
    }

    if (node->parent->parent == NULL) {
        return;
    }

    fixInsert(tree, node);
}

void inorderHelper(Node* node, Node* TNULL) {
    if (node != TNULL) {
        inorderHelper(node->left, TNULL);
        printf("%d ", node->data);
        inorderHelper(node->right, TNULL);
    }
}

void inorder(RedBlackTree* tree) {
    inorderHelper(tree->root, tree->TNULL);
}

RedBlackTree* initializeTree() {
```

```
    RedBlackTree* tree = (RedBlackTree*)malloc(sizeof(RedBlackTree));
    tree->TNULL = (Node*)malloc(sizeof(Node));
    tree->TNULL->color = BLACK;
    tree->TNULL->left = NULL;
    tree->TNULL->right = NULL;
    tree->root = tree->TNULL;
    return tree;
}

int main() {
    RedBlackTree* tree = initializeTree();

    insert(tree, 55);
    insert(tree, 40);
    insert(tree, 65);
    insert(tree, 60);
    insert(tree, 75);
    insert(tree, 57);

    printf("Inorder traversal of the tree:\n");
    inorder(tree);
    printf("\n");

    return 0;
}
```

OUTPUT:

AVL TREE order is:

40 55 57 60 65 75

SLAY TREE:

```
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a Splay Tree node
typedef struct Node {
    int key;
    struct Node* left;
    struct Node* right;
} Node;

// Function to create a new node
Node* createNode(int key) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    newNode->key = key;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to right rotate
void rightRotate(Node** root, Node* node) {
    Node* pivot = node->left;
    node->left = pivot->right;
    if (pivot->right != NULL) {
```

```
        pivot->right->left = node;
    }
    pivot->right = node;
    node->left = NULL;
    if ((*root) == node) {
        (*root) = pivot;
    }
}

// Function to left rotate
void leftRotate(Node** root, Node* node) {
    Node* pivot = node->right;
    node->right = pivot->left;
    if (pivot->left != NULL) {
        pivot->left->right = node;
    }
    pivot->left = node;
    node->right = NULL;
    if ((*root) == node) {
        (*root) = pivot;
    }
}

// Function to splay a node
void splay(Node** root, Node* node) {
    while (node->left != NULL || node->right != NULL) {
        if (node->left == NULL) {
```

```
            leftRotate(root, node);
        } else if (node->right == NULL) {
            rightRotate(root, node);
        } else if (node->left->left == NULL) {
            rightRotate(root, node);
            rightRotate(root, node);
        } else if (node->right->right == NULL) {
            leftRotate(root, node);
            leftRotate(root, node);
        } else if (node->left->right == NULL) {
            leftRotate(root, node->left);
            rightRotate(root, node);
        } else {
            rightRotate(root, node->right);
            leftRotate(root, node);
        }
    }
}

// Function to insert a node into the tree
void insertNode(Node** root, int key) {
    if (*root == NULL) {
        *root = createNode(key);
    } else {
        Node* currentNode = *root;
        while (1) {
            if (key < currentNode->key) {
```

```
            if (currentNode->left == NULL) {
                currentNode->left = createNode(key);
                break;
            }
            currentNode = currentNode->left;
        } else {
            if (currentNode->right == NULL) {
                currentNode->right = createNode(key);
                break;
            }
            currentNode = currentNode->right;
        }
    }
    splay(root, currentNode);
  }
}

// Function to print the tree
void printTree(Node* node) {
  if (node == NULL) {
    return;
  }
  printTree(node->left);
  printf("%d ", node->key);
  printTree(node->right);
}
```

```
int main() {
    Node* root = NULL;
    insertNode(&root, 5);
    insertNode(&root, 3);
    insertNode(&root, 7);
    insertNode(&root, 2);
    insertNode(&root, 4);
    insertNode(&root, 6);
    insertNode(&root, 8);
    printTree(root);
    return 0;
}
```

Output:

Splay tree is:

2 3 4 5 6 7 8