**Assignment 6(b)**

**Implement the scenario of a file system which maintains directory structure using the Red Black Tree. Each node in the tree represents a directory, and the tree is balanced to ensure efficient insertion, deletion, and display operations when navigating through the file system.**

#include <iostream>

using namespace std;

struct Node {

string dirName;

char color; // 'R' for Red, 'B' for Black

Node\* left;

Node\* right;

Node\* parent;

};

Node\* root = NULL;

Node\* createNode(string name) {

Node\* temp = new Node;

temp->dirName = name;

temp->color = 'R';

temp->left = temp->right = temp->parent = NULL;

return temp;

}

void rotateLeft(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 rotateRight(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\* z) {

while (z->parent != NULL && z->parent->color == 'R') {

Node\* gp = z->parent->parent;

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

Node\* y = gp->right;

if (y != NULL && y->color == 'R') {

z->parent->color = 'B';

y->color = 'B';

gp->color = 'R';

z = gp;

} else {

if (z == z->parent->right) {

z = z->parent;

rotateLeft(root, z);

}

z->parent->color = 'B';

gp->color = 'R';

rotateRight(root, gp);

}

} else {

Node\* y = gp->left;

if (y != NULL && y->color == 'R') {

z->parent->color = 'B';

y->color = 'B';

gp->color = 'R';

z = gp;

} else {

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

z = z->parent;

rotateRight(root, z);

}

z->parent->color = 'B';

gp->color = 'R';

rotateLeft(root, gp);

}

}

}

root->color = 'B';

}

void insertDirectory(string name) {

Node\* z = createNode(name);

Node\* y = NULL;

Node\* x = root;

while (x != NULL) {

y = x;

if (name < x->dirName) x = x->left;

else x = x->right;

}

z->parent = y;

if (y == NULL) root = z;

else if (name < y->dirName) y->left = z;

else y->right = z;

fixInsert(root, z);

}

void levelOrderDisplay(Node\* root) {

if (root == NULL) {

cout << "File system is empty\n";

return;

}

Node\* queue[100];

int front = 0, rear = 0;

queue[rear++] = root;

while (front < rear) {

int size = rear - front;

for (int i = 0; i < size; i++) {

Node\* temp = queue[front++];

cout << temp->dirName << "(" << temp->color << ") ";

if (temp->left != NULL) queue[rear++] = temp->left;

if (temp->right != NULL) queue[rear++] = temp->right;

}

cout << "\n";

}

}

Node\* search(Node\* root, string name) {

while (root != NULL) {

if (name == root->dirName) return root;

else if (name < root->dirName) root = root->left;

else root = root->right;

}

return NULL;

}

void deleteDirectory(string name) {

Node\* del = search(root, name);

if (del == NULL) {

cout << "Directory not found!\n";

return;

}

if (del->left != NULL && del->right != NULL) {

cout << "Deletion only supported for leaf or single-child directories.\n";

return;

}

Node\* child = (del->left != NULL) ? del->left : del->right;

if (del->parent == NULL) {

root = child;

if (child != NULL) child->parent = NULL;

} else {

if (del == del->parent->left) del->parent->left = child;

else del->parent->right = child;

if (child != NULL) child->parent = del->parent;

}

delete del;

cout << "Directory deleted.\n";

}

int main() {

int choice;

string dir;

do {

cout << "\n1. Insert Directory\n2. Delete Directory\n3. Display Structure\n4. Exit\n";

cout << "Enter choice: ";

cin >> choice;

if (choice == 1) {

cout << "Enter directory name: ";

cin >> dir;

insertDirectory(dir);

} else if (choice == 2) {

cout << "Enter directory name to delete: ";

cin >> dir;

deleteDirectory(dir);

} else if (choice == 3) {

levelOrderDisplay(root);

}

} while (choice != 4);

return 0;

}