//Assignment 6b
//Sarvesh suryavanshi
//123B1B065

Problem statement: 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.

Code:

```
#include <iostream>
using namespace std;

enum Color { RED, BLACK };

struct Node {
    string name;
    Color color;
    Node* left, *right, *parent;
};

Node* root = NULL;

Node* createNode(string name) {
    Node* node = new Node;
    node->name = name;
    node->color = RED;
    node->left = node->right = node->parent = NULL;
    return node;
}

void rotateLeft(Node*& root, Node* x) {
    Node* y = x->right;
    x->right = y->left;
    if (y->left) y->left->parent = x;
    y->parent = x->parent;
    if (!x->parent) 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* x) {
    Node* y = x->left;
    x->left = y->right;
    if (y->right) y->right->parent = x;
    y->parent = x->parent;
    if (!x->parent) root = y;
    else if (x == x->parent->left) x->parent->left = y;
    else x->parent->right = y;
    y->right = x;
    x->parent = y;
}

void fixInsert(Node*& root, Node* z) {
    while (z != root && z->parent->color == RED) {
        if (z->parent == z->parent->parent->left) {
            Node* 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 {
            Node* 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 insertDirectory(Node*& root, string name) {
  Node* z = createNode(name);
  Node* y = NULL;
  Node* x = root;

  while (x != NULL) {
    y = x;
    if (z->name < x->name) x = x->left;
    else if (z->name > x->name) x = x->right;
    else {
      cout << "Directory already exists.\n";
      return;
    }
  }

  z->parent = y;
  if (y == NULL) root = z;
  else if (z->name < y->name) y->left = z;
  else y->right = z;

  fixInsert(root, z);
}

void inorder(Node* root) {
  if (!root) return;
  inorder(root->left);
  cout << root->name << " (" << (root->color == RED ? "R" : "B") << ") ";
  inorder(root->right);
}

Node* minimum(Node* node) {
  while (node->left) node = node->left;
  return node;
}
```

```
void deleteDirectory(Node*& root, string name) {
    Node* z = root;
    while (z && z->name != name) {
        if (name < z->name) z = z->left;
        else z = z->right;
    }
    if (!z) {
        cout << "Directory not found.\n";
        return;
    }

    Node* y = z;
    Node* x;
    bool yOriginalColor = y->color;

    if (!z->left) {
        x = z->right;
        if (x) x->parent = z->parent;
        if (z == root) root = x;
        else if (z == z->parent->left) z->parent->left = x;
        else z->parent->right = x;
        delete z;
    } else if (!z->right) {
        x = z->left;
        if (x) x->parent = z->parent;
        if (z == root) root = x;
        else if (z == z->parent->left) z->parent->left = x;
        else z->parent->right = x;
        delete z;
    } else {
        y = minimum(z->right);
        yOriginalColor = y->color;
        x = y->right;
        if (y->parent == z) {
            if (x) x->parent = y;
        } else {
            if (x) x->parent = y->parent;
            if (y->parent) y->parent->left = x;
        }
        if (y == root) root = y;
        delete y;
    }
}
```

```
int main() {
    int choice;
    string name;

    while (1) {
        cout << "\n1. Insert Directory\n2. Delete Directory\n3. Display Directory Tree\n4.
Exit\nEnter choice: ";
        cin >> choice;

        if (choice == 1) {
            cout << "Enter directory name to insert: ";
            cin >> name;
            insertDirectory(root, name);
        } else if (choice == 2) {
            cout << "Enter directory name to delete: ";
            cin >> name;
            deleteDirectory(root, name);
        } else if (choice == 3) {
            cout << "Directory structure (inorder):\n";
            inorder(root);
            cout << endl;
        } else {
            break;
        }
    }

    return 0;
}
```

Output Clear

```
1. Insert Directory
2. Delete Directory
3. Display Directory Tree
4. Exit
Enter choice: 1
Enter directory name to insert: f

1. Insert Directory
2. Delete Directory
3. Display Directory Tree
4. Exit
Enter choice: 3
Directory structure (inorder):
f (B)

1. Insert Directory
2. Delete Directory
3. Display Directory Tree
4. Exit
Enter choice: 1
Enter directory name to insert: dere
1. Insert Directory
2. Delete Directory
3. Display Directory Tree
4. Exit
```

Output Clear

```
1. Insert Directory
2. Delete Directory
3. Display Directory Tree
4. Exit
Enter choice: 2
Enter directory name to delete: f

1. Insert Directory
2. Delete Directory
3. Display Directory Tree
4. Exit
Enter choice: 3
Directory structure (inorder):
dere (R)

1. Insert Directory
2. Delete Directory
3. Display Directory Tree
4. Exit
Enter choice: 4


=== Code Execution Successful ===
```