# The Red-Black Restoration

locked



Once upon a time, in the mystical kingdom of Algoria, there existed a Tree of Balance 🌲. This was no ordinary tree; it was the Red-Black Tree, an enchanted data structure that ensured balance in the kingdom's magical records. The Grand Wizard, Merlin the Coder, used this tree to maintain order in the kingdom's vast archives, preventing chaos and inefficiency. However, a mischievous imp named Bugzor tried to tamper with the tree by inserting unbalanced data, causing the kingdom's spells to take longer to cast. To restore harmony, the young apprentice (you!) must help Merlin by performing the following tasks on the enchanted Red-Black Tree:

1.Insertion Spell – Insert magical numbers into the tree while maintaining its balance. 2.Searching Charm – Search for specific magical numbers to verify their existence.

Only by mastering these operations can you defeat Bugzor and restore order to Algoria!

Objectives:
1.Implement a Red-Black Tree with insertion, search, and display functions.
2.**Ensure the tree follows Red-Black Tree properties:**
1. Every node is either Red or Black.
2. The root is always Black.
3. No two consecutive Red nodes exist (No Red-Red violations).
4. Every path from root to leaf has the same number of Black nodes.
5. The tree remains balanced after insertions.

BoilerPlate Code:

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

// Node structure for Red-Black Tree
typedef struct Node {
   int data;
   char color; // 'R' for Red, 'B' for Black
   struct Node *left, *right, *parent;
} Node;

// Root of the Red-Black Tree
typedef struct {
   Node* root;
} RedBlackTree;

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->color = 'R'; // New node must be Red
   newNode->left = newNode->right = newNode->parent = NULL;
   return newNode;
}
// Function to rotate left
void rotateLeft(RedBlackTree* tree, 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)
        tree->root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}

// Function to rotate right
void rotateRight(RedBlackTree* tree, Node* x) {
    Node* y = x->left;
    x->left = y->right;
    if (y->right != NULL)
        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;
}

// Function to insert a node into the Red-Black Tree
void insert(RedBlackTree* tree, int data) {
    // Implement
}

// Function to search for a value in the tree
int search(Node* root, int data) {
    // Implement
}

// Function to store nodes in an array
Node** displayTree(RedBlackTree* tree, int* size) {
    *size = 0; // To be calculated while traversing
    Node** result = (Node**)malloc(100 * sizeof(Node*));
    Node* stack[100];
    int top = -1;
    Node* current = tree->root;
    while (current != NULL || top != -1) {
        while (current != NULL) {
            stack[++top] = current;
            current = current->left;
        }
        current = stack[top--];
        result[(*size)++] = current;
        current = current->right;
    }
    return result;
}

int main() {
    int n;
    scanf("%d", &n);
    RedBlackTree tree;
    tree.root = NULL;
    for (int i = 0; i < n; i++) {
        char command[10];
        int value;
        scanf("%s", command);
        if (strcmp(command, "insert") == 0) {
            scanf("%d", &value);
            insert(&tree, value);
        } else if (strcmp(command, "search") == 0) {
            scanf("%d", &value);
```

```
            printf("%s\n", search(tree.root, value) ? "YES" : "NO");
        } else if (strcmp(command, "display") == 0) {
            int size;
            Node** arr = displayTree(&tree, &size);
            for (int j = 0; j < size; j++) {
                printf("%d(%c) ", arr[j]->data, arr[j]->color);
            }
            printf("\n");
            free(arr);
        }
    }
    return 0;
}
```

### Input Format

First line contains the total number of operations n. Next n lines contain n operations on the tree. (eg- insert , display, search )

### Constraints

1 <= n <= 100

### Output Format

If the operation is “search 10”, print “YES” if value 10 exists in the tree else “NO”. If the operation is “display”, return node value and node colour in “In-order traversal” format.

### Sample Input 0

```
5
insert 10
insert 20
insert 30
insert 15
display
```

### Sample Output 0

```
10(B) 15(R) 20(B) 30(B)
```

### Sample Input 1

```
4
insert 5
insert 10
insert 15
search 15
```

### Sample Output 1

```
YES
```

Submissions: 54
Max Score: 10
Difficulty: Medium

Rate This Challenge:

More

C

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int main() {

    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}
```

Line: 1 Col: 1

Upload Code as File | Test against custom input | Run Code | Submit Code