# 作业 3 红黑树

## 1 红黑树代码实现

失衡恢复函数如下，其中补全了叔叔节点为黑色时的 LL、LR、RL、RR 四种情况。

```
void RedBlackTree::fixViolation(Node* pt) {
    // Fix violation logic
    Node* parent_pt = nullptr;
    Node* grand_parent_pt = nullptr;


    while ((pt != root) && (pt->color == RED) && (pt->parent->color ==
   RED)) {
        parent_pt = pt->parent;
        grand_parent_pt = pt->parent->parent;

        if (parent_pt == grand_parent_pt->left) {
            Node* uncle_pt = grand_parent_pt->right;

            // Uncle is red
            if (uncle_pt != nullptr && uncle_pt->color == RED) {
                grand_parent_pt->color = RED;
                parent_pt->color = BLACK;
                uncle_pt->color = BLACK;
                pt = grand_parent_pt;
            } else {// Uncle is black
                // LL-Case and LR-Case, write your code here
                // LR
                if (pt == parent_pt->right) {
                    rotateLeft(parent_pt);
                    pt = parent_pt;
                    parent_pt = pt->parent;
                }
                // LL
                grand_parent_pt->color = RED;
                parent_pt->color = BLACK;
                rotateRight(grand_parent_pt);
            }
        } else {
            Node* uncle_pt = grand_parent_pt->left;

            if ((uncle_pt != nullptr) && (uncle_pt->color == RED)) {
                grand_parent_pt->color = RED;
                parent_pt->color = BLACK;
                uncle_pt->color = BLACK;
                pt = grand_parent_pt;
            } else {
                // RR-Case and RL-Case, write your code here
                // RL
                if (pt == parent_pt->left) {
                    rotateRight(parent_pt);
                    pt = parent_pt;
                    parent_pt = pt->parent;
                }
                // RR
                grand_parent_pt->color = RED;
                parent_pt->color = BLACK;
                rotateLeft(grand_parent_pt);
            }
        }
    }

    root->color = BLACK;
}
```

## 2 代码测试

为测试，将中序遍历代码做了以下修改。

```
void RedBlackTree::inorderUtil(Node* root) {
    // Inorder traversal logic
    if (root == nullptr)
        return;
    inorderUtil(root->left);
    std::string color;
    if (root->color == RED) {
        color = "RED";
    } else {
        color = "BLACK";
    }
    std::cout << "Data: " << root->data << " Color: " << color << std::
   endl;
    inorderUtil(root->right);
}
```

在 main.cpp 中，插入九个数对红黑树进行测试。

正确的红黑树应为:

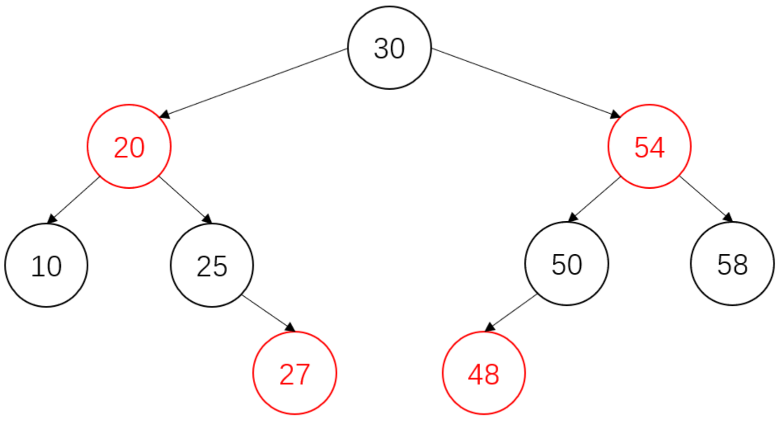
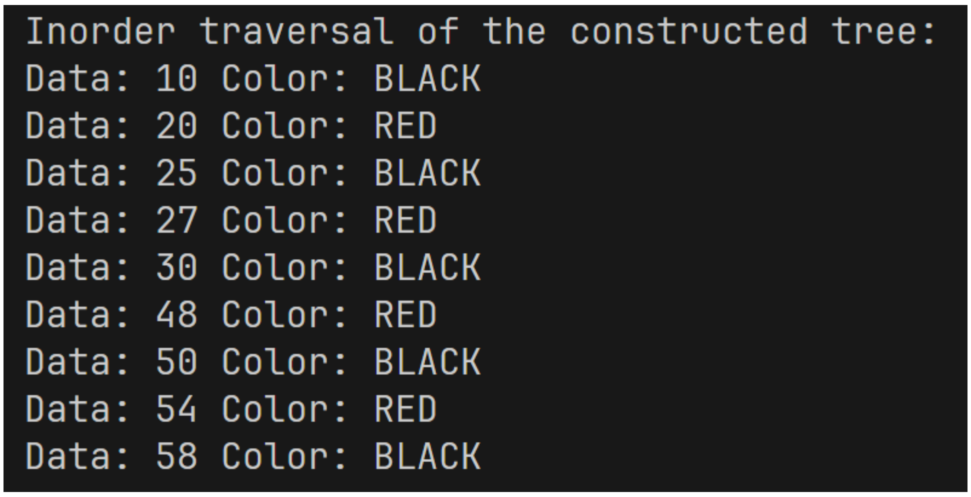


图 1: 正确情况

代码中序遍历得到的结果为:



```
Inorder traversal of the constructed tree:
Data: 10 Color: BLACK
Data: 20 Color: RED
Data: 25 Color: BLACK
Data: 27 Color: RED
Data: 30 Color: BLACK
Data: 48 Color: RED
Data: 50 Color: BLACK
Data: 54 Color: RED
Data: 58 Color: BLACK
```

图 2: 程序运行结果截图

符合要求，代码正确。