Lab 3: 红黑树插入算法

PB21020718 曾健斌

一、实验内容

编码实现红黑树的插入算法，使得插入后依旧保持红黑性质。

- 输入为`insert.txt`文件
- 输出红黑树插入时的 case 序号，以及不同方式遍历得到的树存入相应文件

二、算法实现

本次实验内容较为简单，代码思路均借鉴课本，此处不再搬运伪代码，而是直接给出源码

此次实验并未使用类，而是选择使用结构体，这是为了更贴近课本的实现：

```
enum Color{
    red,
    black,
};

struct RBTnode                      //红黑树结点
{
    int key;
    Color color;
    RBTnode* left;
    RBTnode* right;
    RBTnode* parent;
};

typedef RBTnode* RBTnodelist;

typedef struct RBTree               //红黑树
{
    RBTnodelist root;
    RBTnodelist NIL;
}*RBTreeptr;
```

首先需要对新定义的红黑树进行初始化：

```
void
RBTinit(RBTreeptr T)
{
    T->NIL = new RBTnode;
    T->NIL->color = black;
    T->root = T->NIL;               //初始时，根结点与NIL等同
    T->root->parent = T->NIL;
}
```

为了实现插入，还应实现左旋右旋的操作：

```
void
```

```
LeftRotate(RBTreeptr T, RBTnodelist x)
{
    RBTnodelist y = new RBTnode;
    y = x->right;
    x->right = y->left;
    if (y->left != T->NIL)
        y->left->parent = x;
    y->parent = x->parent;
    if (x->parent == T->NIL)
        T->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(RBTreeptr T, RBTnodelist x)
{
    RBTnodelist y = new RBTnode;
    y = x->left;
    x->left = y->right;
    if (y->right != T->NIL)
        y->right->parent = x;
    y->parent = x->parent;
    if (x->parent == T->NIL)
        T->root = y;
    else if (x == x->parent->right)
        x->parent->right = y;
    else
        x->parent->left = y;
    y->right = x;
    x->parent = y;
}
```

然后是本次实验最重要的`RBInsert()`和`RBInsertFixup()`算法:

```
void
Insert(RBTreeptr T, RBTnodelist z)
{
    RBTnodelist y = T->NIL;
    RBTnodelist x = T->root;
    while (x != T->NIL){                //二叉树的插入
        y = x;
        if (z->key < x->key)
            x = x->left;
        else
            x = x->right;
    }
    z->parent = y;
    if (y == T->NIL)
        T->root = z;
    else if (z->key < y->key)
```

```
        y->left = z;
    else
        y->right = z;
    z->left = T->NIL;
    z->right = T->NIL;
    z->color = red;                         //将结点涂红并调用InsertFixup()进行调整
    InsertFixup(T, z);
    return;
}

void
InsertFixup(RBTreeptr T, RBTnodelist z)
{
    RBTnodelist y;
    while(z->parent->color == red){
        if(z->parent == z->parent->parent->left){
            y = z->parent->parent->right;
            if(y->color == red){
                z->parent->color = black;
                y->color = black;
                z->parent->parent->color = red;
                z = z->parent->parent;        //case 1
                cout << "1 ";
            }
            else{
                if(z == z->parent->right){
                    z = z->parent;
                    LeftRotate(T, z);
                    cout << "2 ";              //case 2
                }
                else
                    cout << "3 ";        //case 3（此处为与case 2相区分，在原实现基础上增加了else
                z->parent->color = black;
                z->parent->parent->color = red;
                RightRotate(T, z->parent->parent);
            }
        }
        else{
            y = z->parent->parent->left;
            if(y->color == red){
                z->parent->color = black;
                y->color = black;
                z->parent->parent->color = red;
                z = z->parent->parent;
                cout << "4 ";                  //case 4-6同case 1-3
            }
            else{
                if (z == z->parent->left){
                    z = z->parent;
                    RightRotate(T, z);
                    cout << "5 ";
                }
                else
                    cout << "6 ";
                z->parent->color = black;
```

```
                z->parent->parent->color = red;
                LeftRotate(T, z->parent->parent);
            }
        }
    }
    T->root->color = black;
}
```

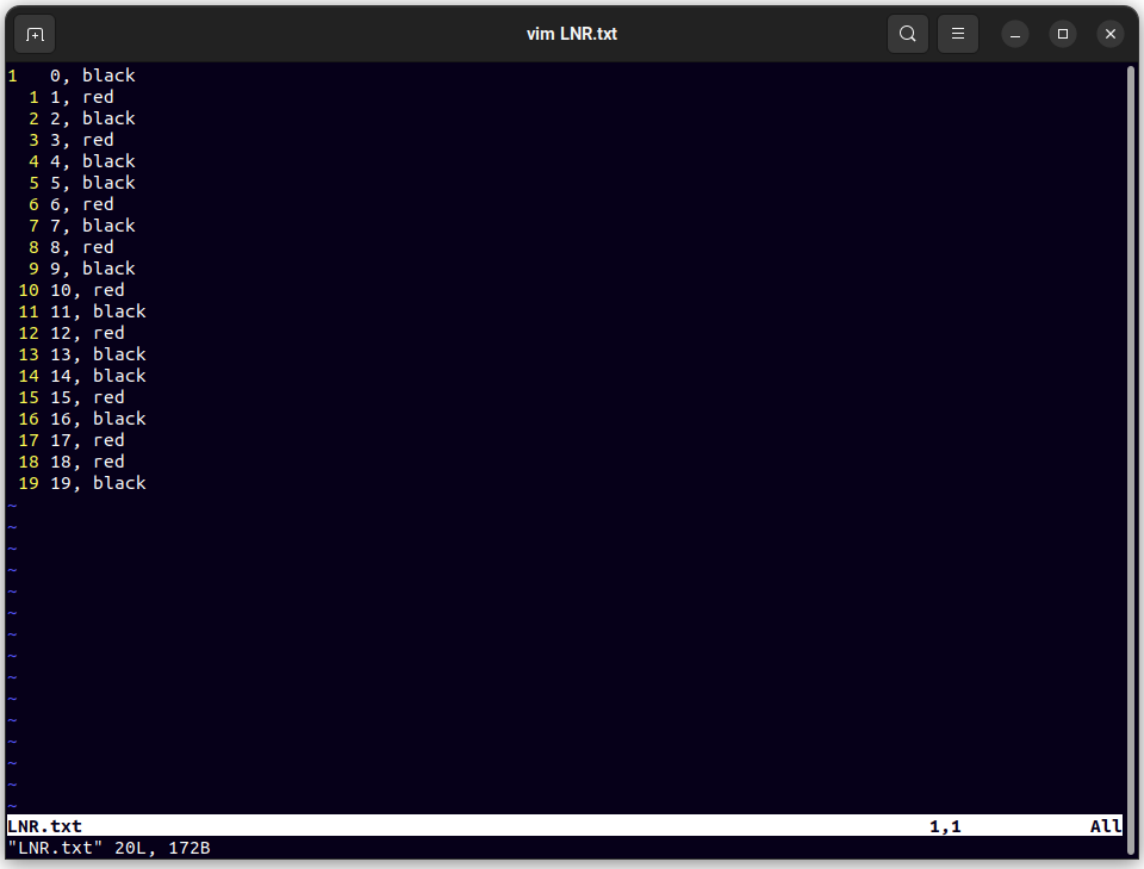
遍历与文件读写算法详见`Traverse.cpp`、`rw.cpp`，由于非本次实验重点，此处不再赘述。

三、实验结果

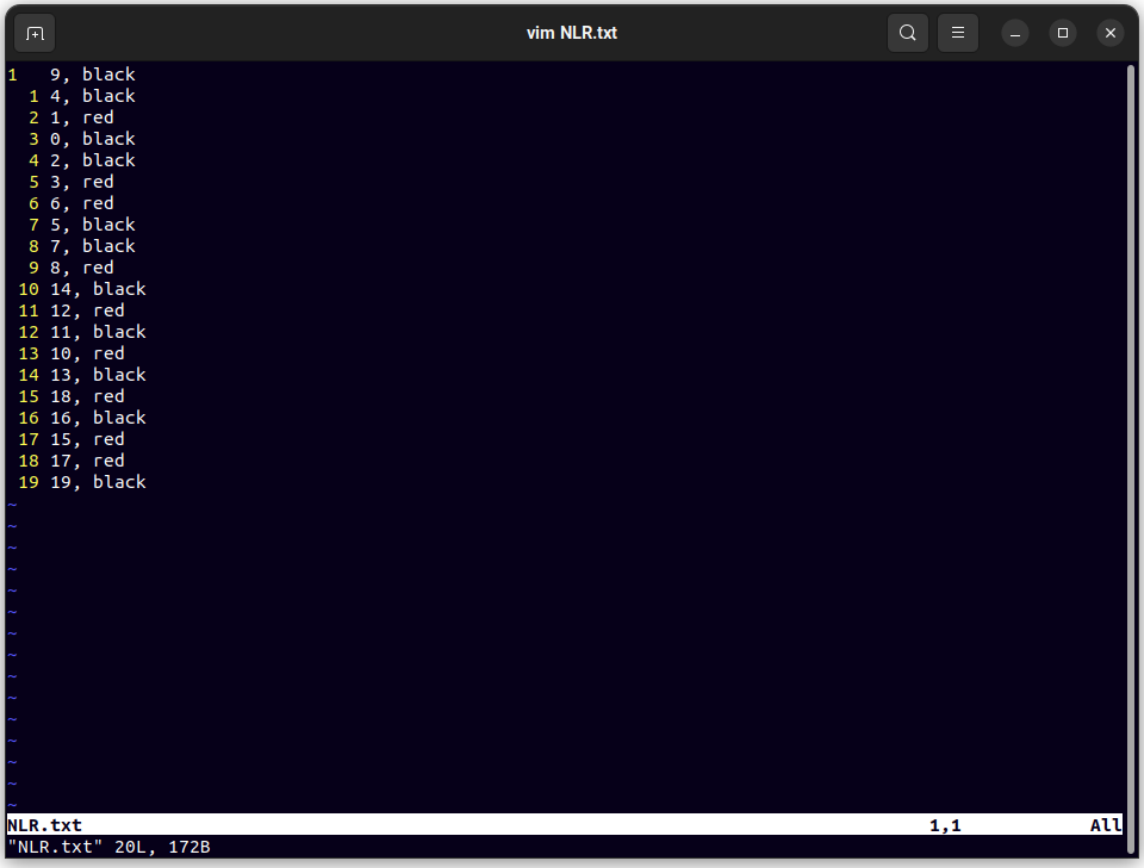
以下是`InsertFixup()`打印出的每一次插入时的`case`，可以看到`case`数小于 20，这是因为有些结点插入时或是成为根结点，或是父结点为黑。

```
22:05 zeng@ocoubuntu /home/zeng/Documents/alg-lab/lab333/source/cpp_source
% ./main
2 1 4 5 4 2 4 1 1 3 4 5 2 4 6 6
```

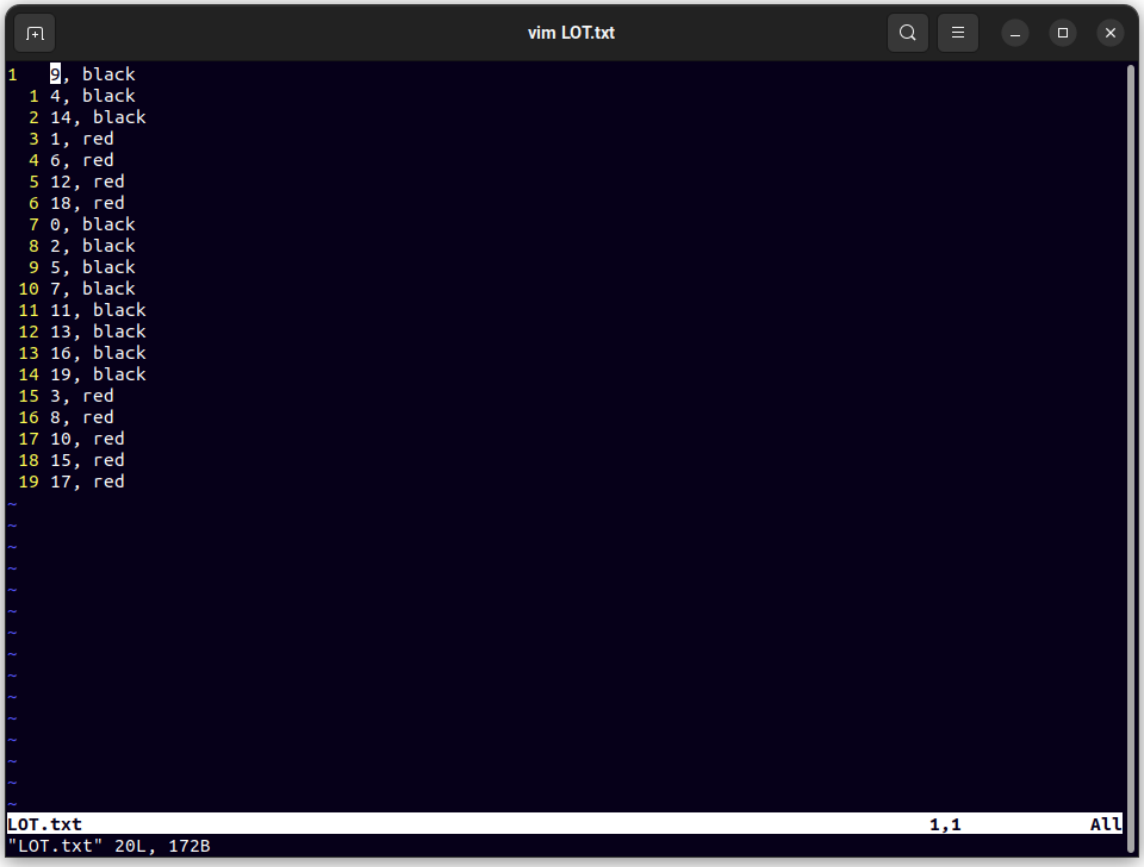
先序遍历结果：



中序遍历结果：


vim NLR.txt
1 9, black
1 4, black
2 1, red
3 0, black
4 2, black
5 3, red
6 6, red
7 5, black
8 7, black
9 8, red
10 14, black
11 12, red
12 11, black
13 10, red
14 13, black
15 18, red
16 16, black
17 15, red
18 17, red
19 19, black
NLR.txt 1,1 All
"NLR.txt" 20L, 172B

层序遍历结果：


vim LOT.txt
1 9, black
1 4, black
2 14, black
3 1, red
4 6, red
5 12, red
6 18, red
7 0, black
8 2, black
9 5, black
10 7, black
11 11, black
12 13, black
13 16, black
14 19, black
15 3, red
16 8, red
17 10, red
18 15, red
19 17, red
LOT.txt 1,1 All
"LOT.txt" 20L, 172B

四、实验总结

本次实验总体上较为简单，实验过程主要是对于课本算法伪代码的实现

- 本次实验让我得以温习二叉树的旋转、遍历等算法
- 在理解红黑树插入的过程中学习了2-3-4树等相关知识，对于红黑树的理解更进了一步