红黑树插入算法

姓名：陈鸿绪　学号：PB21000224　日期：10.23

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。(即实现教材 p178 页的 RB-INSERT, RB_INSERT_FIXUP 算法)，要求结点属性如下：

```
TNode =  {
            Color:  red / black,
            Key:  int,
            Left:  TNode*,
            Right:  TNode*,
            P:  TNode*
         }
```

实验目的：

熟悉红黑树基本插入操作，理解红黑树性质，加深算法理解。

算法设计思路：

1. 构造结点数据类型，从 txt 文件中读取数据。
2. 构建二叉排序树的插入算法，利用循环将插入结点插入正确位置。
3. 构建红黑树维护算法, 将 2 中插入的结点调整至正确位置。需要分情况讨论，如果需要调整结点的父结点是黑色，则直接结束不需要调整；如果调整结点的父节点是红色，则需要调整，分两种情况讨论：a. 调整结点的叔结点是红色，则将叔结点和父节点颜色调为黑色，调整结点的祖父结点调整红色。b. 调整结点的叔结点是黑色, 则要判断调整结点、和其父节点是左还是右孩子，如果全部都是左孩子或者全部为右孩子，则只需要一次右旋操作或者左旋操作即可，如果是一个左孩子一个右孩子，则需要对调整结点父节点右旋（左旋)，后再对调整结点原来的祖父结点左旋（右旋)。
4. 将读出来的结点一个一个插入红黑树、维护红黑树，即得到最后红黑树。再

进行先序、中序、层次遍历操作，输出最后红黑树的结构。

主体代码和注释：

1. 结点数据类型：

```
struct TNode{
	bool color; //0 红， 1 黑
	int key;
	struct TNode *L,*R,*p;
};
TNode *Head;
typedef struct TNode TNode;
```

2. 中序、先序、层次输出：

```
void LNR(TNode *p,FILE *fp){
	if(p==NULL) return;
	LNR(p->L,fp);
	if(p->color==0) fprintf(fp,"%d,红\n",p->key);
	else fprintf(fp,"%d,黑\n",p->key);
	LNR(p->R,fp);
}
void NLR(TNode *p,FILE *fp){
	if(p==NULL) return;
	if(p->color==0) fprintf(fp,"%d,红\n",p->key);
	else fprintf(fp,"%d,黑\n",p->key);
	LNR(p->L,fp);
	LNR(p->R,fp);
}
void LOT(int N,FILE *fp){
	TNode *q=(TNode *)malloc(sizeof(TNode)*N);
	int q_len=1;
	q[0]=*Head;
	do{
		if(q[0].color==0) fprintf(fp,"%d,红\n",q[0].key);
		else fprintf(fp,"%d,黑\n",q[0].key);
		printf("\n");
		TNode s=q[0];
		for(int i=0;i<=q_len-2;i++){
			q[i]=q[i+1];
		}
		q_len--;
		if(s.L!=NULL) q[q_len++]=*(s.L);
```

```
            if(s.R!=NULL) q[q_len++]=*(s.R);
        }while(q_len);
}
```

3. 左旋、右旋

```
void LEFT_ro(TNode *p){
        if(p->p!=NULL){
            if(p->p->L==p) p->p->L=p->R;
            else p->p->R=p->R;
        }
        else{
            Head=p->R;
        }
        TNode *temp=p->R->L;
        p->R->p=p->p;
        p->R->L=p;
        p->p=p->R;
        p->R=temp;
        if(temp) temp->p=p;
}
void RIGHT_ro(TNode *p){
        if(p->p!=NULL){
            if(p->p->L==p) p->p->L=p->L;
            else p->p->R=p->L;
        }
        else{
            Head=p->L;
        }
        TNode *temp=p->L->R;
        p->L->R=p;
        p->L->p=p->p;
        p->p=p->L;
        p->L=temp;
        if(temp) temp->p=p;
}
```

4. 维护红黑树

```
void RB_insert_fixup(TNode *z){
        while(z->p->color==0){
            if(z->p->p->L==z->p){
                TNode *y=z->p->p->R;
                if(y!=NULL&&y->color==0){
```

```
                z->p->color=1;
                y->color=1;
                z->p->p->color=0;
                z=z->p->p;
            }//case 1
            else if(z==z->p->R){
                z=z->p;
                LEFT_ro(z);
            }//case 2
            else{
                z->p->color=1;
                z->p->p->color=0;
                RIGHT_ro(z->p->p);
            }//case 3

        }
        else{
            TNode *y=z->p->p->L;
            if(y!=NULL&&y->color==0){
                z->p->color=1;
                y->color=1;
                z->p->p->color=0;
                z=z->p->p;
            }//case 4
            else if(z==z->p->L){
                z=z->p;
                RIGHT_ro(z);
            }//case 5
            else{
                z->p->color=1;
                z->p->p->color=0;
                LEFT_ro(z->p->p);

            }//case 6
        }
        Head->color=1;
        if(z==Head) break;
    }

}
```

5. 二叉排序树插入算法

```
void RB_insert(TNode *z){
```

```
    TNode *p=Head,*pre;
    while(true){
        if(z->key<=p->key) pre=p,p=p->L;
        else pre=p,p=p->R;
        if(p==NULL){
            if((pre->key)>=(z->key)) {pre->L=z;}
            else {pre->R=z;}
            z->L=NULL,z->R=NULL,z->p=pre;
            break;
        }
    }
    RB_insert_fixup(z); //维护红黑树
}}
```

算法测试与正确性检验：

根据输出的三个文件：LOT.txt，NLR.txt，LNR.txt，再通过输出结果每个结点的颜色、父节点、子节点，画出红黑树，检验得到满足红黑树的定义。(输出见附件：LOT.txt，NLR.txt，LNR.txt)。再经过构造其他数据再进行测试，检验后均正确。

对于程序的输出内容如下：

实验过程中遇到的困难及收获：

困难在调试代码的时候发现最后生成的红黑树的指针有些结点并没有正确指示，在最后生成的红黑树中缺失了几个结点，最后发现是由于旋转操作指针没有正确变换导致（缺失了一条语句）。实验收获是我提高了代码能力，更深入地理解了红黑树的插入结点算法。