第二次上机实验报告

一、 问题介绍

编程实现红黑树的以下功能：遍历、取最大、取最小、求后续、求前驱、插入结点、删除结点、查找具有给定秩的元素，并用多组数据分析性能。

二、 算法思想介绍

- 数据结构：
  结点和树均用结构体表示，红黑树结点间的关系用链表表示。
- void inorder_rbtree_walk(Tree* T,Node* x)：
  中序遍历红黑树，先遍历左子树，然后输出此结点的 key 值，最后遍历右子树，递归调用此函数。
- Node* rbtree_minimun(Tree* T)：
  采用循环结构，输出红黑树中最左的元素即最小值
- Node* rbtree_maximun(Tree* T)：
  采用循环结构，输出红黑树中最右的元素即最大值。
- Node* rbtree_succesor(Tree* T,Node* x)：
  分类讨论，若结点有右子树，则对右子树调用 Node* rbtree_minimun(Tree* T)求得的最小值为后继；若结点无右子树，则采用循环结构向上搜寻，第一个比此结点大的元素为后继。
- Node* rbtree_predecessor(Tree* T,Node* x)：
  分类讨论，若结点有左子树，则对左子树调用 Node* rbtree_maximun(Tree* T)求得的最大值为前驱；若结点无左子树，则采用循环结构向上搜寻，第一个比此结点小的元素为前驱。
- int rb_insert(Tree* T, Node* z)：
  采用循环结构，向下搜寻找到插入结点的位置。将结点设为红色，再向上通过左旋和右旋对结点进行调整，使之满足红黑树的性质，分为以下三种情况（若 z 的父结点为左孩子，否则相反）：（1）z 的叔结点为红色，调整颜色后将 z 上移两层；（2）z 的叔结点是黑色且 z 是右孩子，通过左旋使其变为左孩子；（3）z 的叔结点是黑色且 z 是左孩子，调整颜色并进行右旋
- int rb_delete(Tree* T, Node* z)：
  分类讨论，若 z 有一侧的子树为空，则将另一侧的子树代替 z 即可删除 z；若 z 既有左子树又有右子树，则用 z 的后继代替 z，并分后继是否为 z 的孩子进行讨论。最后调整红黑树的颜色，分为以下四种情况（若 z 为左孩子，否则相反）：（1）x 的兄弟结点为红色，则调整颜色做左旋；（2）x 的兄弟结点为黑色且兄弟结点的子结点是黑色，则调整颜色并将 x 上移一层；（3）x 的兄弟结点为黑色且兄弟结点的左孩子是红色、右孩子是黑色，交换兄弟结点及其左孩子的颜色，并进行右旋；（4）x 的兄弟结点为黑色且兄弟结点的右孩子是红色，调整颜色并进行左旋。
- Node* rb_select(Node* x, int i)：

比较 i 与结点的秩的大小，递归调用此函数。

三、 实验中问题、理解与解决办法

- 为了便于后续操作，红黑树中用 NIL 代替 NULL，且树 T 的根节点的父结点为 NIL。不方便构造全局变量 NIL，于是用结构体表示树并在主函数中创建结点 T->NIL，调用 T 时即可调用 T->NIL。
- 由于要实现 selection 操作，结点的属性需要加上 size，所有的操作需维护 size。在遍历、取最大、取最小、求后续、求前驱操作中，未改变树，不需维护 size，而在插入和删除操作中，待插入（删除）结点向上路径的所有结点的 size 均需+(-)1，且左旋和右旋需改变结点的 size。
- 在分析性能时，需统计关键操作的次数，于是将 void 型的插入和删除函数改为 int 型，返回其中循环的次数。
- 插入和删除的情况很复杂，需仔细考虑各种情况
- 用多次插入结点来构造红黑树
- 在主函数中采用循环结构，便于对红黑树重复操作。

四、 测试结果

- 依次插入 3,9,2,15,10,8，创建红黑树：

```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
6
输入要插入的关键字value
3
3
```

```
6
输入要插入的关键字value
9
3 9
```

```
6
输入要插入的关键字value
2
2 3 9
```

```
6
输入要插入的关键字value
15
2 3 9 15
```

```
6
输入要插入的关键字value
10
2 3 9 10 15
```

```
6
输入要插入的关键字value
8
2 3 8 9 10 15
```

- 求最小值：

```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
2
最小值为:2
```

- 求最大值

```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
3
最大值为:15
```

- 求 9 的后继

```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
4
输入value，求value的后继
9
10
```

- 求 9 的前驱

```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
5
输入value，求value的前驱
9
8
```

- 删除 key 值为 9 的结点

```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
7
输入要删除的关键字value
9
2 3 8 10 15
```

- 查找秩为 3 的结点

```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
8
输入i值，查找第i小的元素
3
8
```

五、 性能分析

由于插入、删除操作复杂，且是红黑树区别于普通二叉搜索树的地方，所以重点分析创建、插入、删除操作的时间复杂度。随机生成长度为 n 的数组，将其创建为红黑树，并插入、删除某个结点，统计函数里关键部分的执行次数来检验理论复杂度。

实验结果如下所示：

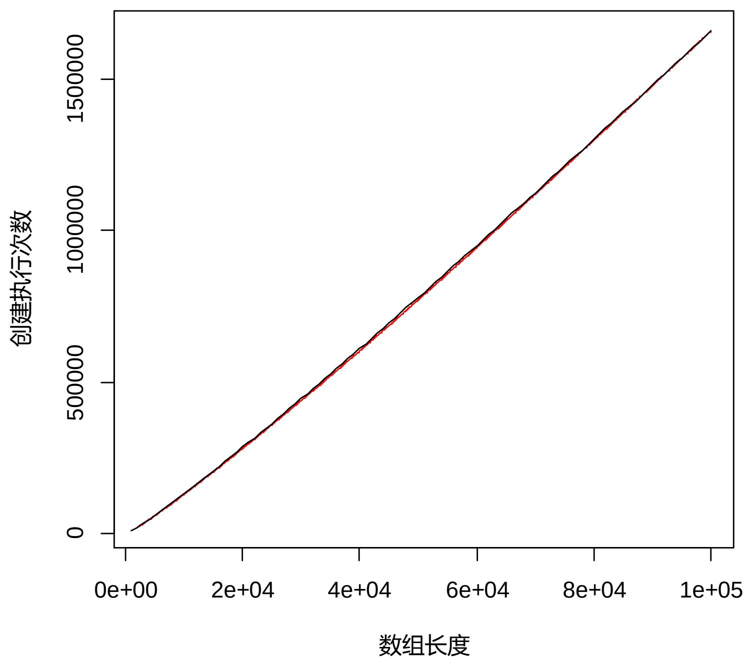
```
选择对红黑树进行的操作：
1.遍历
2.求最小值
3.求最大值
4.求后继
5.求前驱
6.插入结点
7.删除结点
8.查找具有给定秩的元素
9.随机产生数组，分析性能
10.退出
9
输入数组长度
2000
创建红黑二叉树循环次数:21441
插入一个结点循环次数13
删除一个结点循环次数10
```

作执行次数关于 n 的表格（其中插入和删除为取十次的平均值）：

数组长度 n	2000	5000	10000	20000	50000	100000
创建红黑树的执行次数	21441	59917	130664	282003	772894	1659750
插入结点的执行次数	12.4	14.0	13.9	15.5	17.7	18.4
删除结点的执行次数	10.7	12.0	13.3	14.4	15.6	16.7

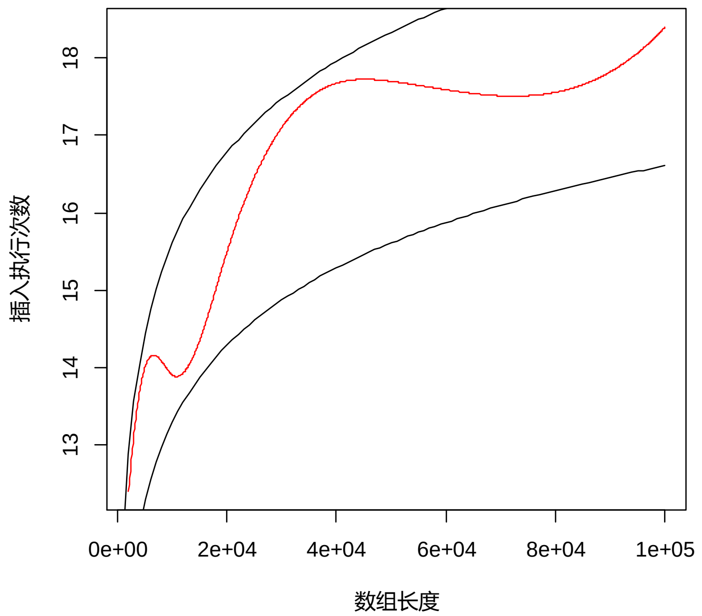
作出曲线图：

（1） 创建红黑树执行次数-数组长度（红线）



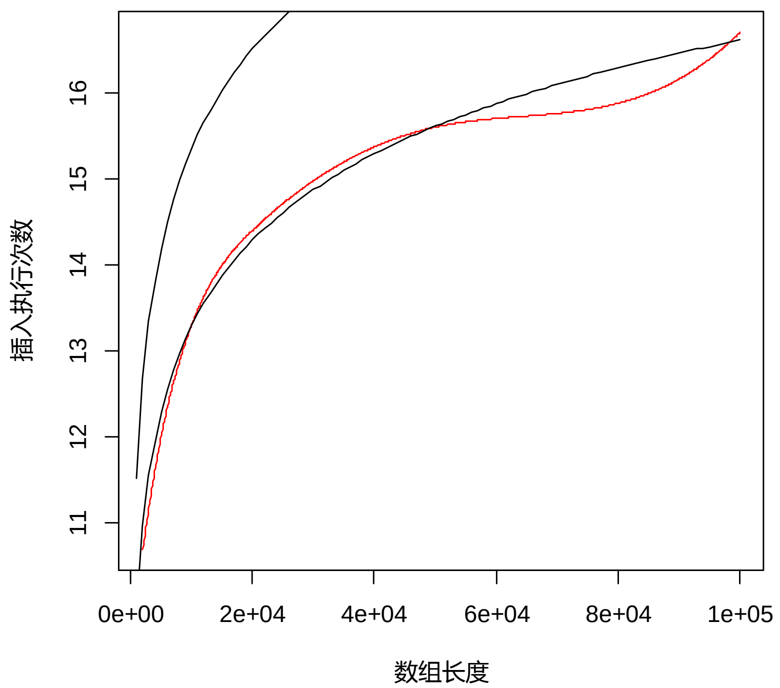
曲线图与 y=n*logn（黑线）几乎重合。

（2） 插入结点执行次数-数组长度（红线）



下面的黑线为 y=logn，上面的黑线为 y=1.175logn，红线的趋势为对数函数，且红线在 y=1.175logn 之下。

（3） 删除结点执行次数-数组长度（红线）



下面的黑线为 y=logn，上面的黑线为 y=1.155logn，红线的趋势为对数函数，且红线在 y=1.155logn 之下。

六、 实验结论

在红黑树的结点个数 n 非常大(n>1000)时，创建红黑树的复杂度近似为 nlogn，创建和删除结点的复杂度均满足 O(logn)。