**实验3-红黑树与顺序统计树**

1. **实验要求**

**1、输入输出格式:**

* **a)两个实验建立一个共同的project文件夹，每个文件夹分别包含3个文件夹：**
  + **input文件夹：存放输入数据**
  + **source文件夹：源程序**
  + **output文件夹：输出数据**
* **b)input：**
  + **实验一输入文件中每行一个随机数据，总行数大于等于60，分别读取12、24、36、48、60个正整数进行构建红黑树。**
  + **实验二不需要输入文件，直接对实验一构造的红黑树进行删除节点操作。**
* **c)output：**
  + **为每种数据规模建立一个子文件夹，分别为size12,size24,size36,size48,size60,其输出结果数据导出到其对应子文件下面**
    - **preoreder.txt ：输出构建好的红黑树的前序遍历序列**
    - **inorder.txt: 输出构建好的红黑树的中序遍历序列**
    - **postorder.txt: 输出构建好的红黑树的后序遍历序列**
    - **time1.txt：运行时间效率的数据。测试插入操作构建树的花费的时间，要求每插入10个节点测试一下花费的时间，并记录下构建完成所花的总时间**
  + **第二个实验输出结果同样是导入到相同的对应子文件夹下面**
    - **delete\_data.txt ：输出删除的两个数据**
    - **time2.txt: 测试删除掉实验要求删除掉的两个节点所花费的时间，每删除掉一个节点测试一次**

**2、算法实现:**

* **a)本次实验需要实现红黑树部分基本算法主要包括如下：**
* **1)实现红黑树左旋操作 LEFT-ROTATE(T, x) 实现右旋操作 RIGHT-ROTATE(T, x)**
* **2)实现红黑树插入节点的操作 RB-INSERT(T, z)以及插入之后修正为红黑书的的算法 RB-INSERT-FIXUP(T, x)（在函数实现过程中对于 case1 case2 case3 的三部分代码要注释清楚）**
* **3)实现红黑树删除节点的操作 RB-DELETE(T, z)以及插入之后修正为红黑书的的算法 RB-DELETE-FIXUP(T, x) （在函数实现过程中对于 case1 case2 case3 case4的四部分代码要注释清楚）**
* **4)实现按要求数据构建顺序统计树的操作**
* **5)实现遍历输出构建好的红黑树的操作**
* **6)实现查找顺序统计树的第i小关键字的操作OS-SELECT(T.root,i)**

**b) 为了验证第二个实验的正确性，要求编写一个检测程序，使用中位数一章的线性时间的选择算法Select(a,p,r,i)在输入数据找到找到第n/3小的节点和第n/4小节点，与OS-SELECT(T.root,i)的结果delete\_data进行对比检查**

**3、实验细节**

* **a)本次实验要求实现的是附加一个x.size信息的红黑即顺序统计树**
* **b)输入数据要求是互不相同的正整数**
* **c)自行设计一个较方便的函数，使得在检查时能够显示树的详细信息，以便能够观察到结点插入、删除前后树的结构变化(包括结点颜色，秩,父子节点关系等信息) ，每行输出一个key的信息**
* **d)第二个实验要求删除的第n/3小的节点和第n/4小的节点的n是动态变化的，而不是静态的，执行一次删除操作n的值减少1，例如n为12，首先删除12/3=4小节点 然后删除12/4=3小的节点**
* **e)由于删除节点等函数代码量比较大,要注意代码可读性和条理性,注释清楚实现过程**

1. **实验环境**

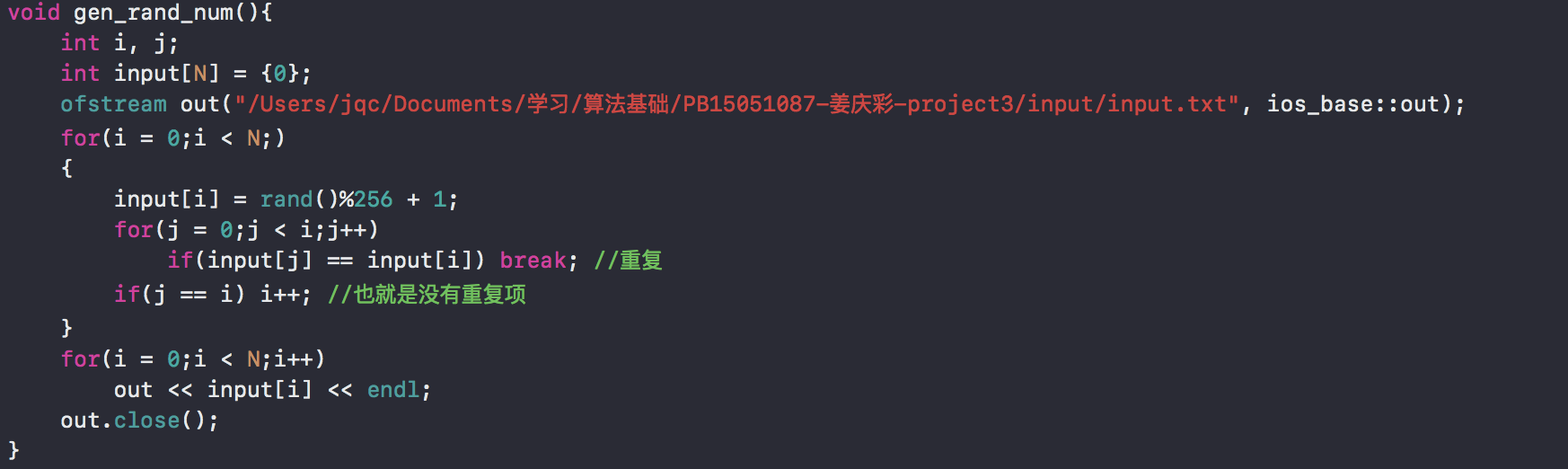
编译环境：xcode11

机器内存：8G

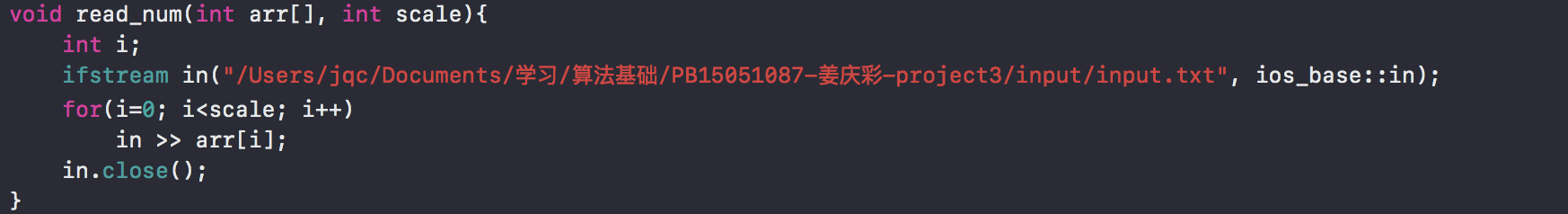
时钟主频：2.9GHz

1. **实验过程与实验关键代码截图（结合文字说明）**

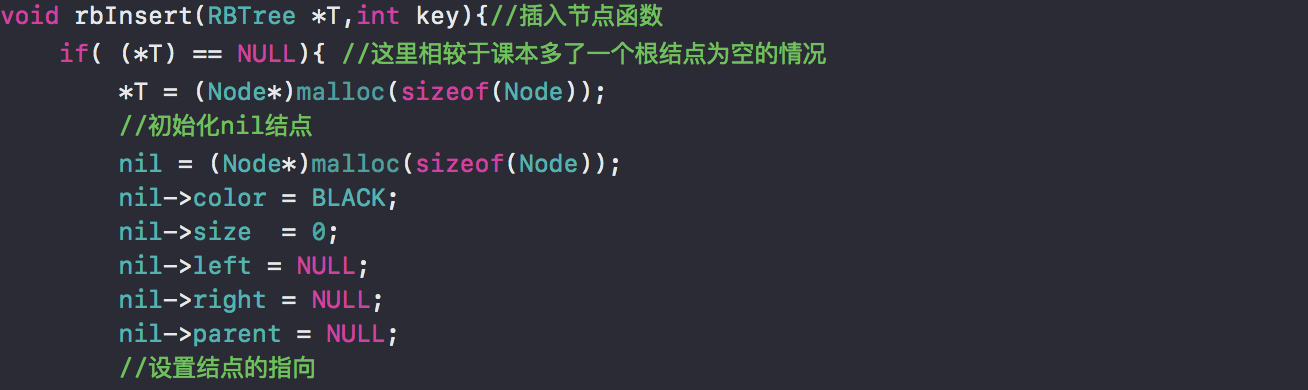
**实验中首先生成60个随机数字**

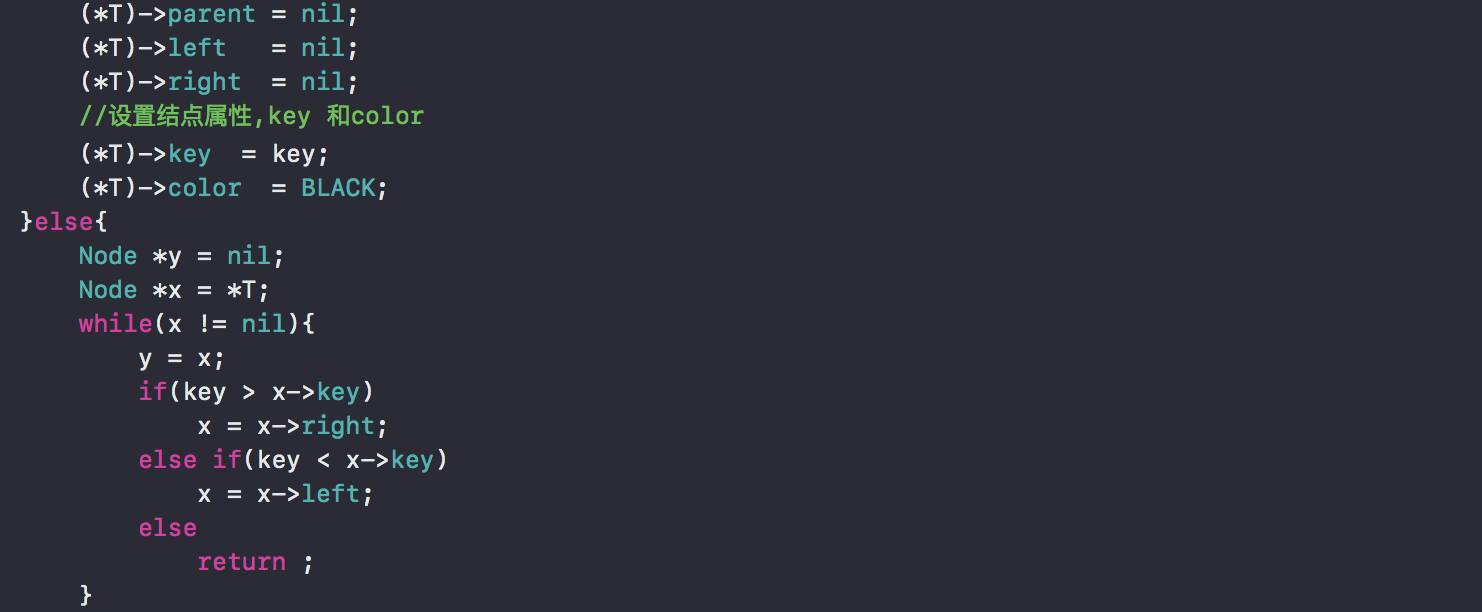
****

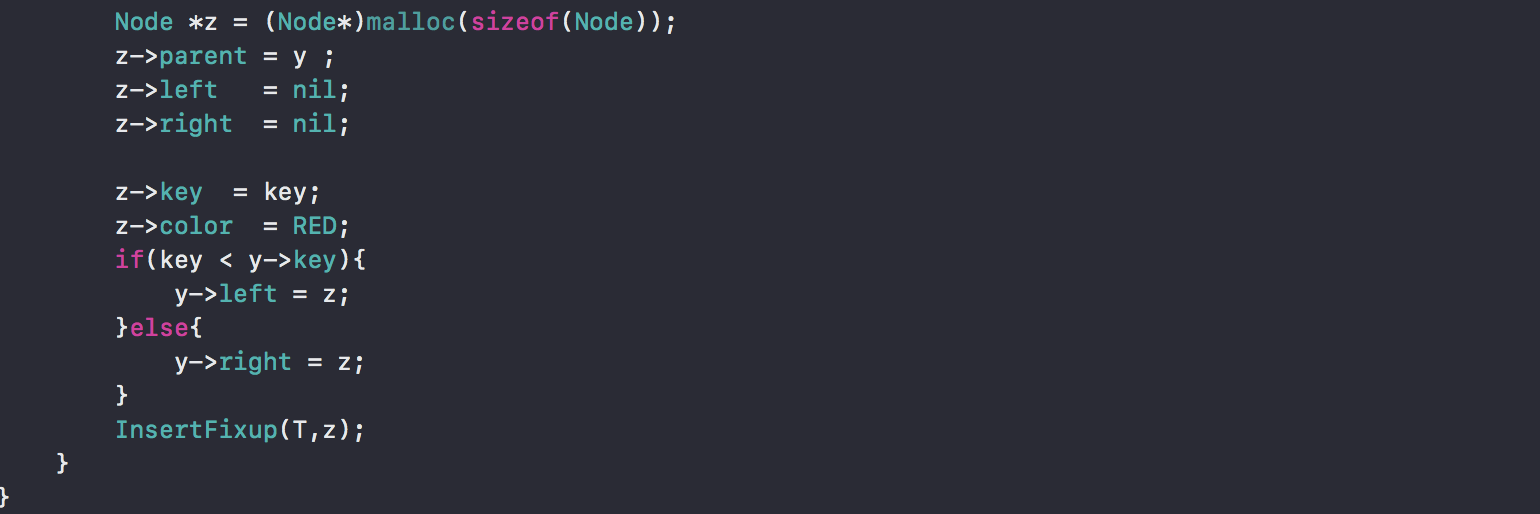
**void read\_num(in tarray[],int scale) 是顺序读取前scale个整数的函数，从文 件 input\_intergers.txt 读到数组 array 中。**

****

**void rbInsert(RBTree \*T,int key); 是插入节点函数。首先，将红黑树当作一颗二叉查找树，将节点插入;然后，将节点着色为红色;最后，通过"旋转和重新着色"等一系列操作来修正该树，使之重新成为一颗红黑树。即分为三步，第一步将红黑树当作一颗二叉查找树，将节点插入。第二步将插入的节点着色为"红色"。第三步通过一系列的旋转或着色等操作，使之重新成为一颗红黑树。**

****

****

****

**红黑树插入修正函数 void InsertFixup(RBTree \*T,Node \*z);在向红黑树中插入节点之后(失去平衡)，再调用该函数，目的是将它重新塑造成一颗红黑树。当向一棵空树中插入结点时，则新结点将作为整棵树根结点，且为黑色(性质 2)。当向一棵非空树中插入一个结点时，新结点的颜色都是为红色。插入成功后，需要判断是否破坏了红黑树的 5 个性质。向非空树中插入一个结点，不可能破坏性 质 1、2、3、5，唯一可能被破坏只有性质4 —— 出现 2 个连续的红结点[新节 点和父节点为红色]，且性质 4 被破坏。考虑书上给的 6 种 case 情况:**

**前提 1:父节点 P 为祖父节点 G 的左孩子**

**情况 1):叔结点为红色**

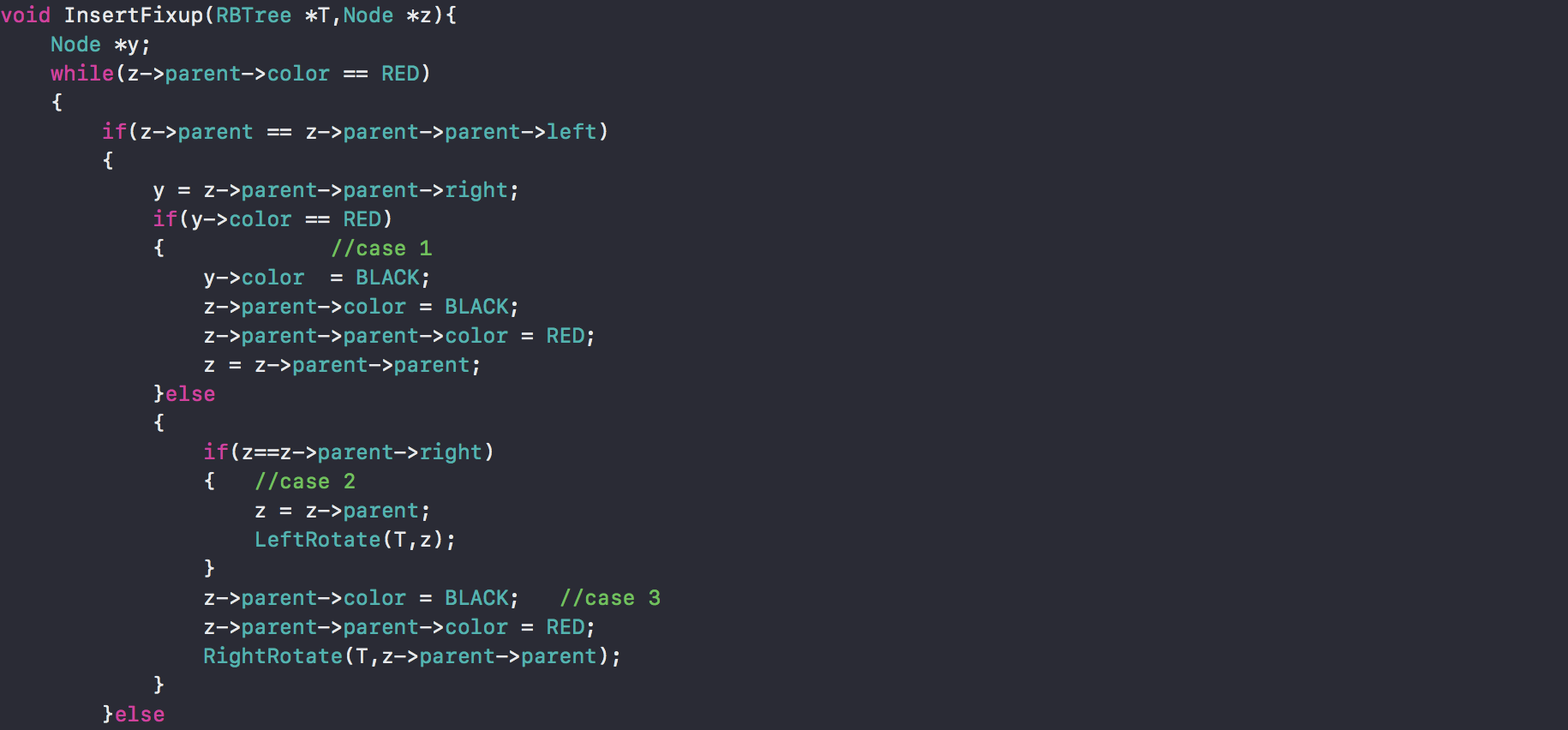
**情况 2):叔结点为黑色，新结点为左孩子**

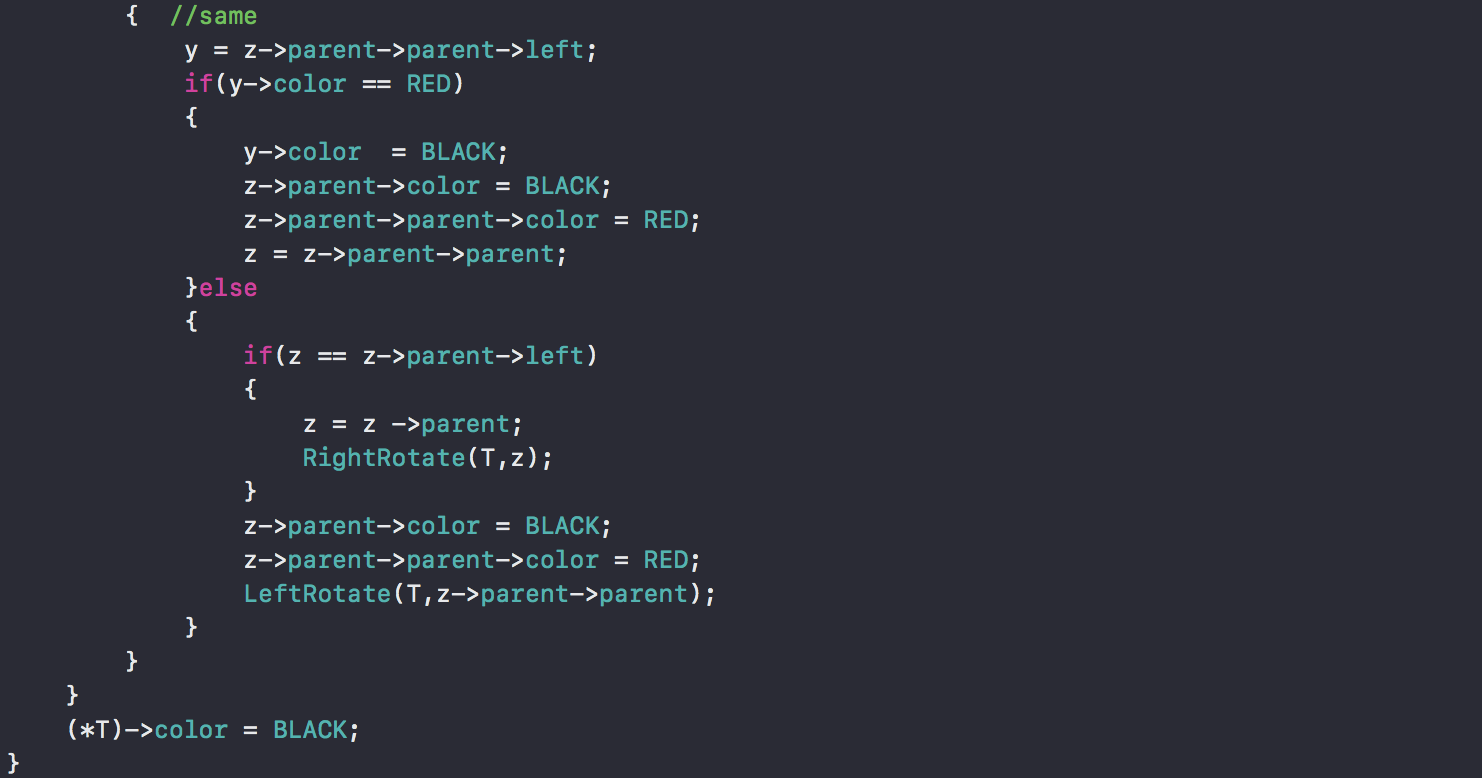
**情况 3):叔结点为黑色，新结点为右孩子**

**前提 2:父节点 P 为祖父节点 G 的右孩子**

**情况 4):叔结点为红色**

**情况 5):叔结点为黑色，新结点为父结点 P 的左孩子 情况 6):叔结点为黑色，新结点为父结点 P 的右孩子**

****

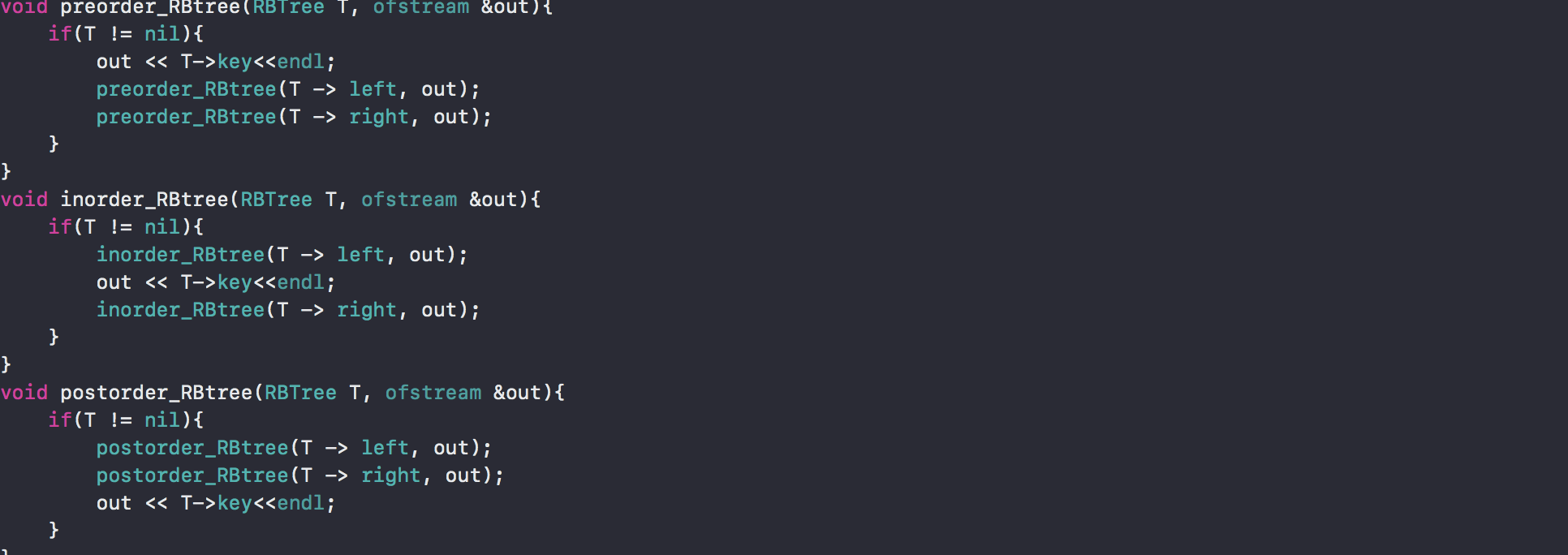
****

**对树 T 分别进行前序、中序、后序遍历，用递归实现。**

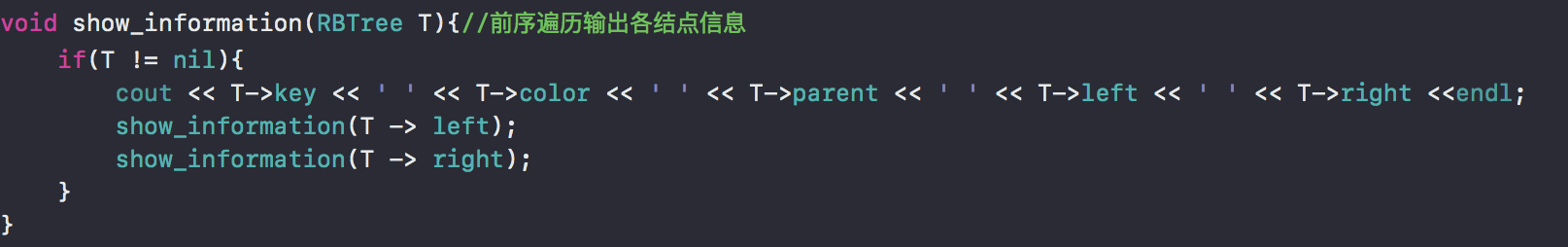
**前序遍历:在 void preorder\_rbtree(RBTree \*T,int scale)中打开文件指针，再调 用 void preorder(RBTree T,FILE \*fout)进行前序遍历。**

**中序遍历:在 void inorder\_rbtree(RBTree \*T,int scale)中打开文件指针，再调用 void inorder (RBTree T,FILE \*fout)进行中序遍历。**

**后序遍历:在 void postorder\_rbtree(RBTree \*T,int scale)中打开文件指针，再调 用 void postorder (RBTree T,FILE \*fout)进行后序遍历。**

****

**PPT中要求的显示节点信息的函数如下：**

****

**为了找出上述生成的红黑树中第 n/4 小的节点和第 n/2 小的节点，需要在 树的节点域上加上 size 整型变量，用来标记以该节点为根的树中有多少节点。在 红黑树中需要修改的是在插入和删除节点时按课本给出的方法重新调整 size 的数值。**

**Node \*os\_select(Node \*x,int i)函数查找以 x 为根的子树中包含第 i 小关键 字的结点。**

****

**void rbDelete(RBTree \*T,Node \*z)是删除结点函数。使用老师上课讲的删除代码实现。将红黑树内的某一个节点删除。需要执行的操作依次是:首先，将 红黑树当作一颗二叉查找树，将该节点从二叉查找树中删除;然后，通过"旋转 和重新着色"等一系列来修正该树，使之重新成为一棵红黑树。详细描述如下:**

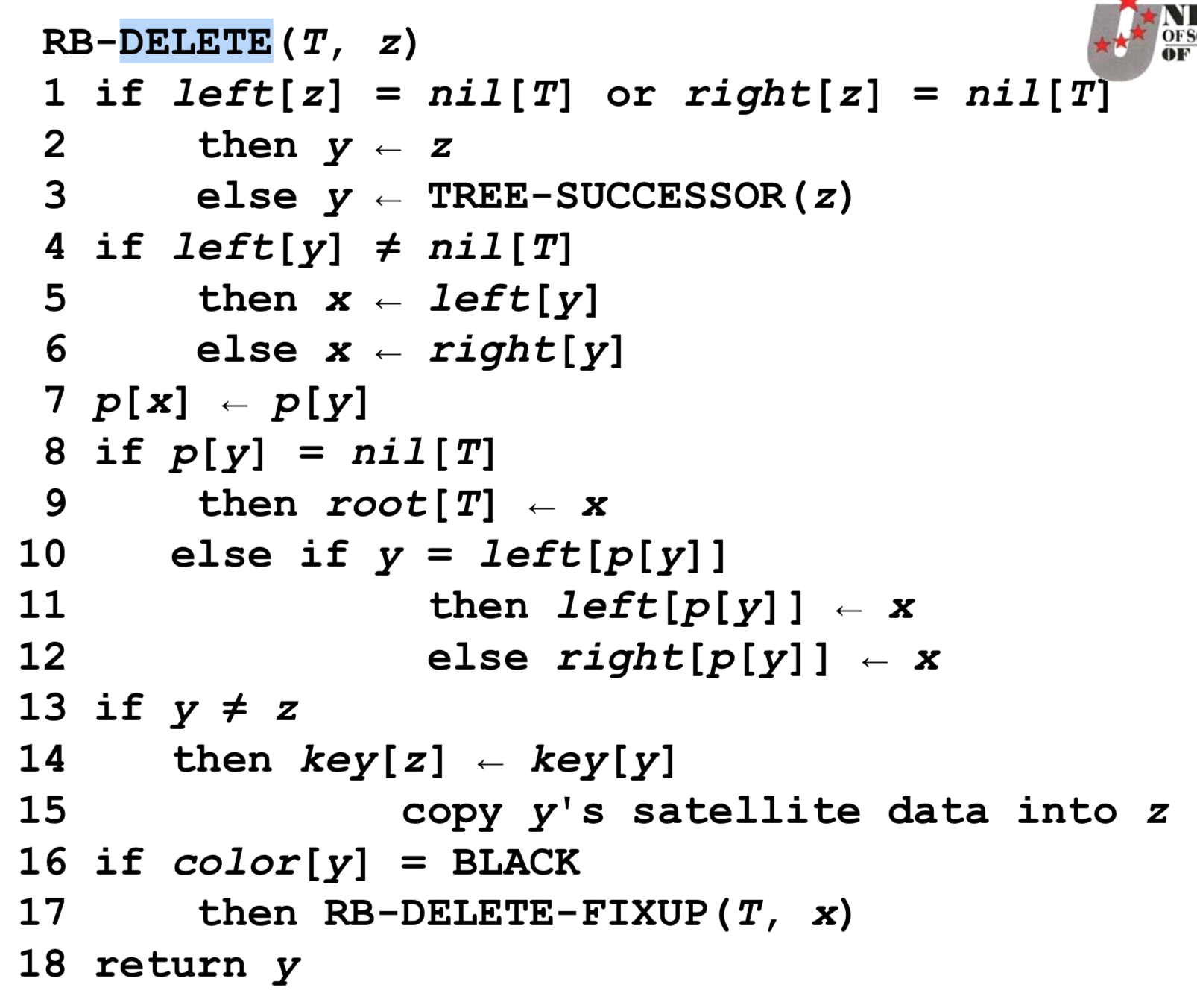
**第一步:将红黑树当作一颗二叉查找树，将节点删除。这和"删除常规二叉查找树中删除节点的方法是一样的"。分 3 种情况:**

**1 被删除节点没有儿子，即为叶节点。那么，直接将该节点删除就 OK 了。**

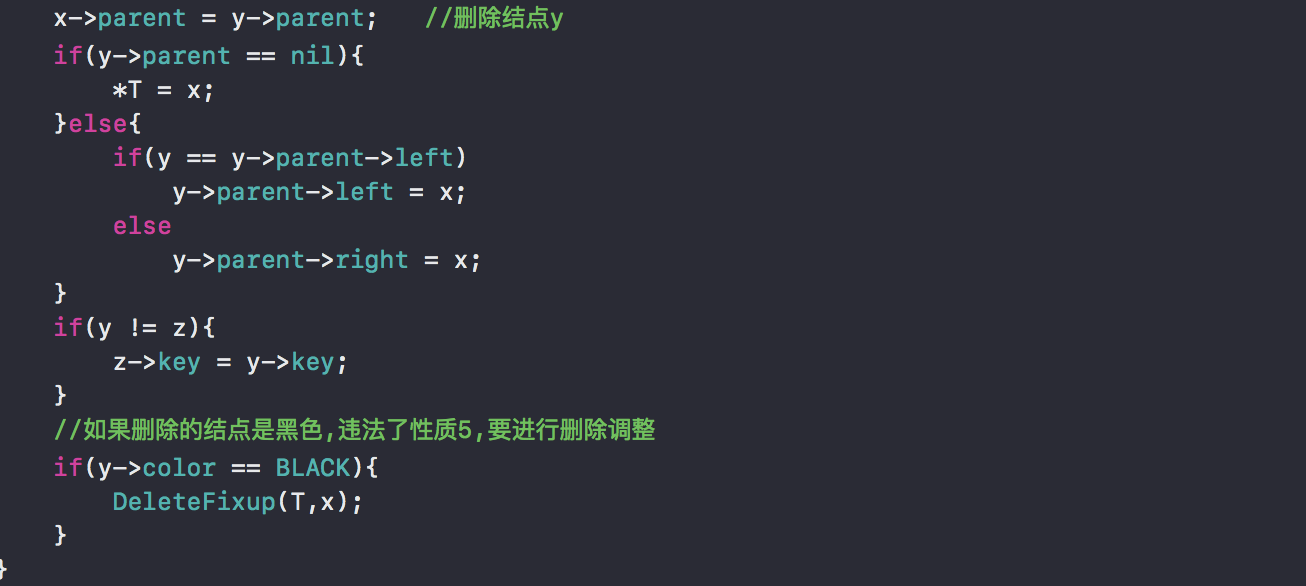
**2 被删除节点只有一个儿子。那么，直接删除该节点，并用该节点的唯一子节点顶替它的位置。**

**3 被删除节点有两个儿子。那么，先找出它的后继节点;然后把“它的后继节点的内容”复制给“该节点的内容”;之后，删除“它的后继节点”。**

**第二步:通过"旋转和重新着色"等一系列来修正该树，使之重新成为一棵红 黑树。因为"第一步"中删除节点之后，可能会违背红黑树的特性，所以需要通过 "旋转和重新着色"来修正该树，使之重新成为一棵红黑树。**

****

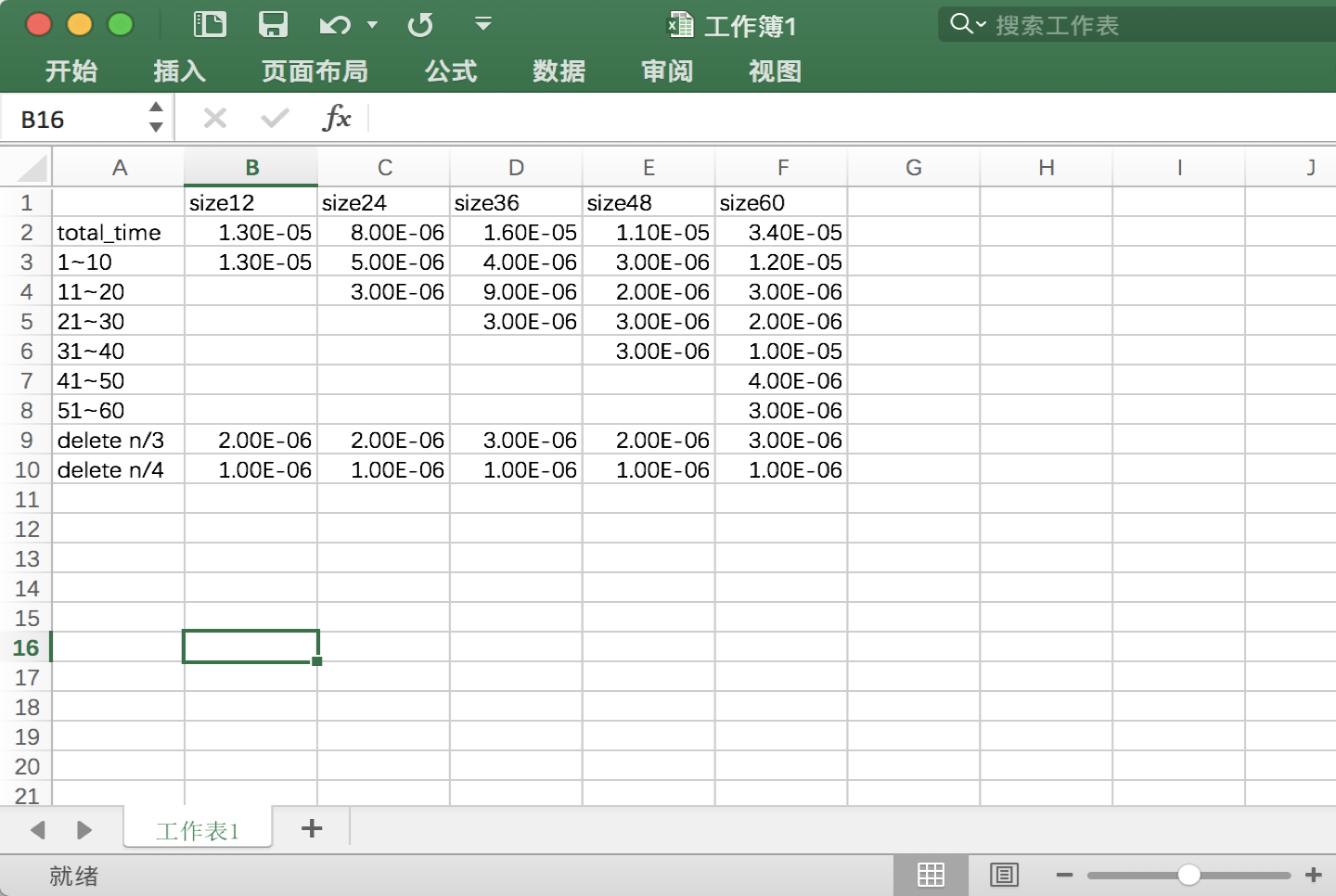
****

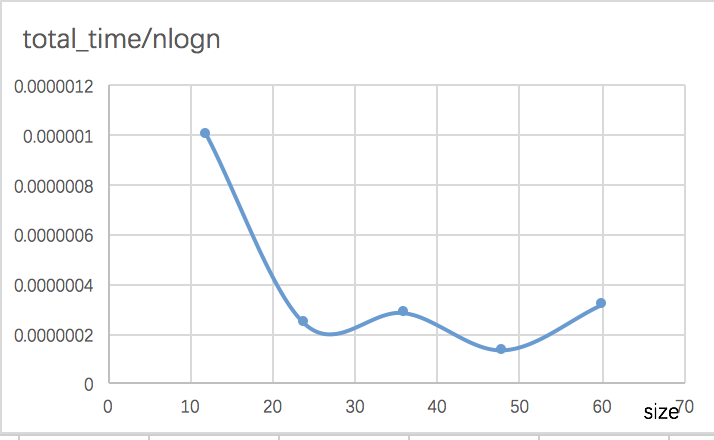
****

1. **实验结果、分析（结合相关数据图表分析）**

实验中记录的原始记录时间如下：

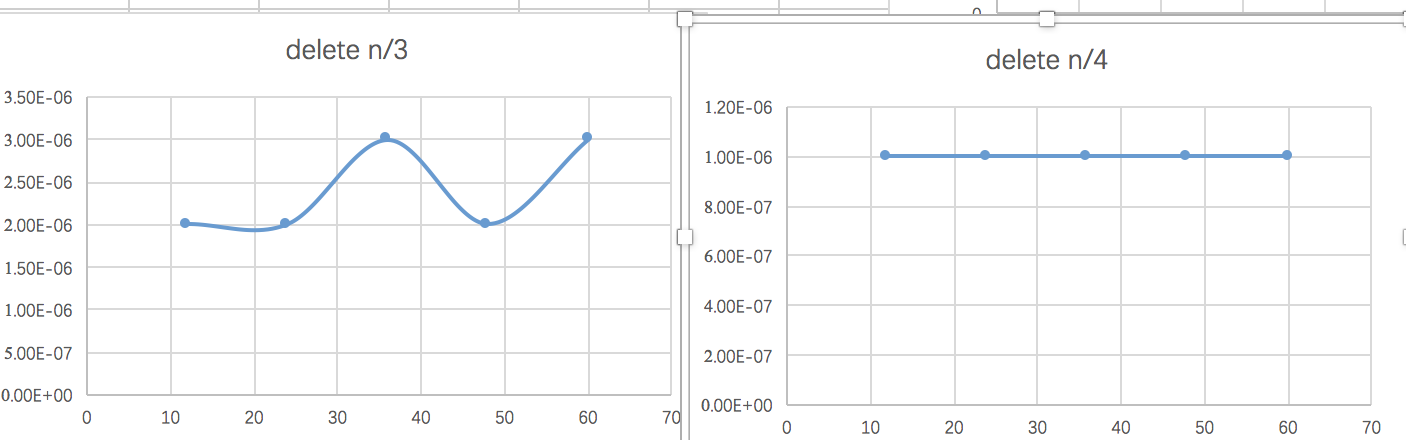
插入节点所用时间：





可以看出，n>20时total\_time/nlogn大致为直线在数据规模 n 为 20 时，比 A\*nlogn 大了不少，猜测应该是插入起始阶段产生了额外的开销，虽然时间较小，但是对小规模数据( 即时间较短时 )影响会很大，偏离曲线。仔细分析程序，发现在程序中应该是开始生成并初始化nil结点耗时较多，且不属于插入操作，对 n=20产生了影响。

**删除节点所用时间：**

****

可以看出，删除 n/3 结点用时比n/4结点用时长，因为红黑树可以通过在删除节 点时进行旋转来调整高度，以使整棵树的状态保持相对平衡。n/3 结点是在靠近根处，n/4 结点远离根，根据算法可知，理论上删除 n/3 结点用时比n/4 结点用时长，与实验结果相符。

1. **实验心得**

本次实验过程中本同学正经历严重流感，实验报告设计较为简陋，望助教理解