Project3红黑树和区间树实验报告

徐煜森 PB16110173

1. **实验要求**

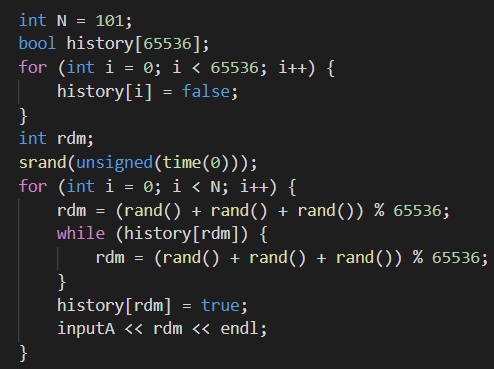
实验1：实现红黑树的基本算法， 分别对整数 n =20、40、60、80、100，随机生成n 个互异的正整数（K1, K2, K3, ……, Kn）, 以这 n 个正整数作为节点的关键字，向一棵初始空的红黑树中依次插入n 个节点， 然后随机选择其中n/10个节点进行删除，统计插入和删除算法运行所需时间，画出时间曲线。

实验2：实现区间树的基本算法，随机生成30个正整数区间，以这30个正整数区间的左端点作为关键字构建红黑树，向一棵初始空的红黑树中依次插入30个节点，然后随机选择其中3个区间进行删除。实现区间树的插入、删除、遍历和查找算法。

1. **实验环境**
2. Windows10 64位x86，机器内存8G，时钟主频2.59GHz
3. 软件环境：Visual Studio 2017
4. **实验过程**
5. **编译选项**

注意编译选项 –std = c++11

1. **Ex1 红黑树**
2. 随机生成数据

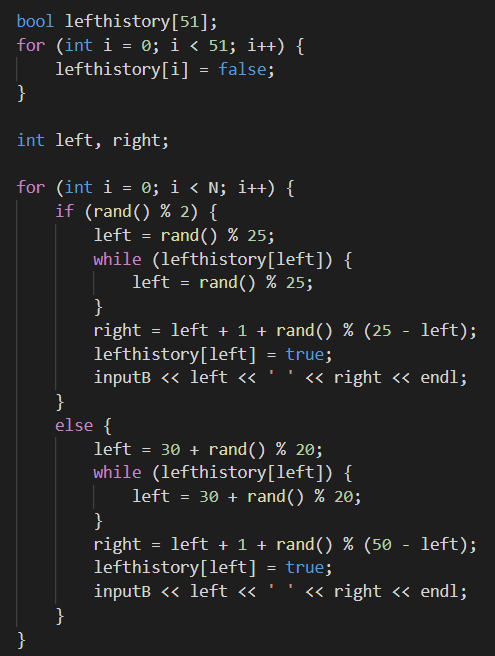


1. 按照书上代码实现红黑树的各个基本操作，按照实验要求调用操作，并进行格式化输出。期间遇到了不少问题，代码中也有部分需要注意的细节，将在关键代码部分和总结部分叙述。
2. 程序运行截图

将部分内容进行屏幕输出方便调试。

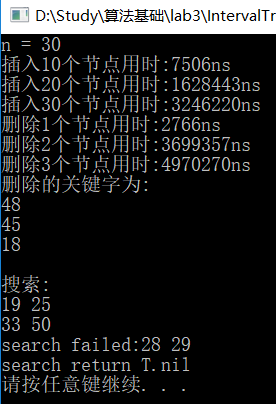


1. **Ex2 区间树**
2. 随机生成数据



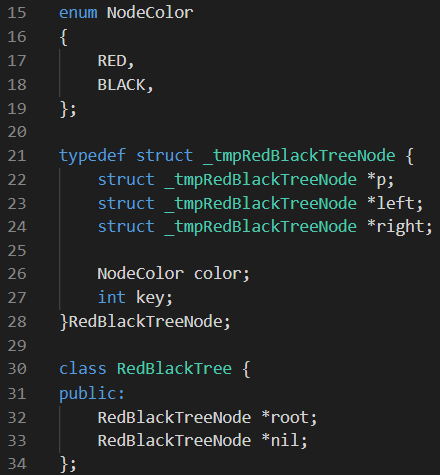
1. 按照书上算法实现区间树的数据结构扩张，和区间搜索函数。思考后修改左旋、右旋、插入、删除函数，维护节点max值。
2. 程序运行截图

输出运行信息到屏幕方便调试。



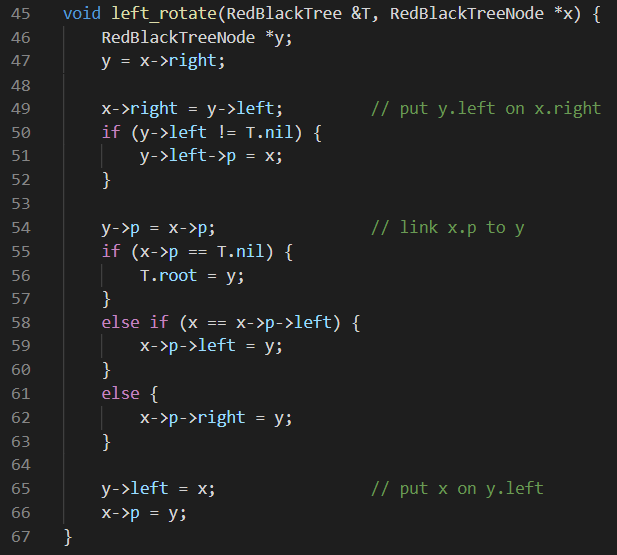
1. **关键代码截图**
2. **Ex1 红黑树**

**树结构的实现：**



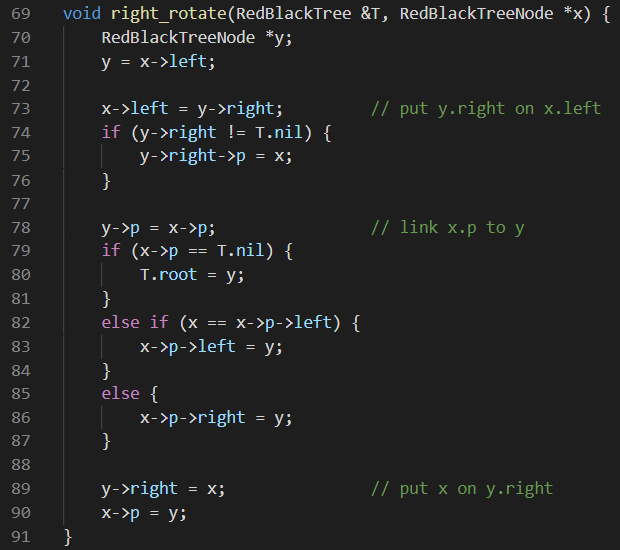
1. 左旋

按照书上的方法实现左旋函数，图中注释标明了三个步骤：



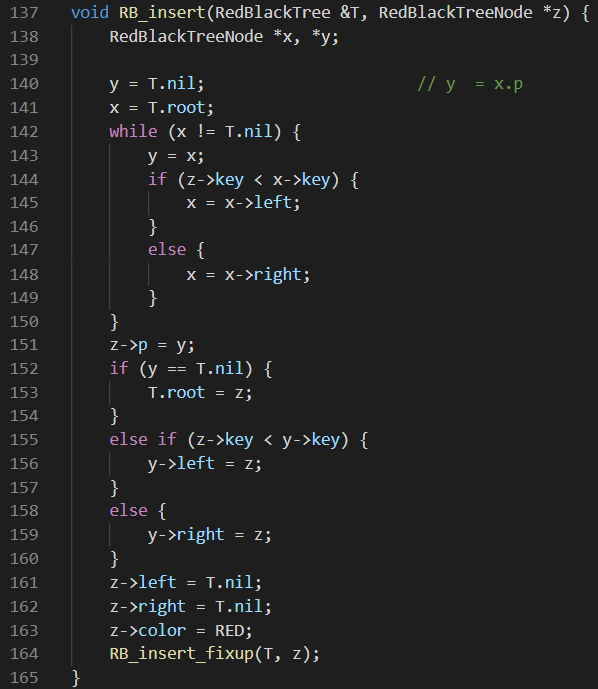
1. 右旋

按照书上的方法实现右旋函数，与左旋对称：



1. 插入

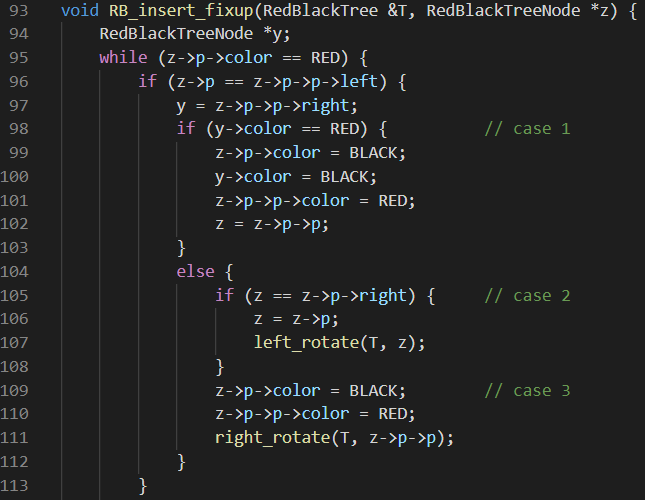
按照书上方法实现插入函数，其中y为插入节点的父亲：

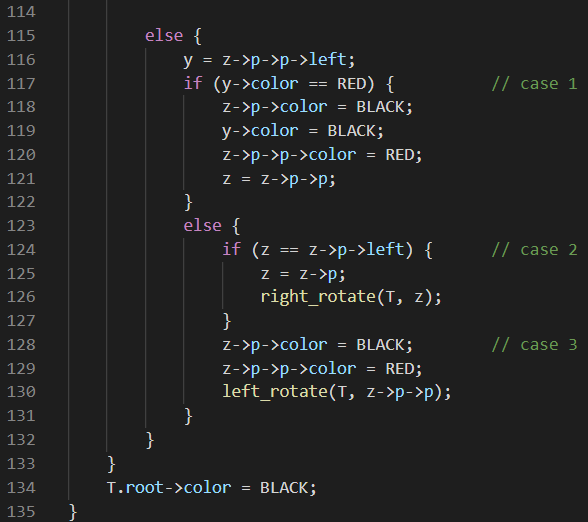


1. 插入后修正

按照书上方法实现insert\_fixup函数：

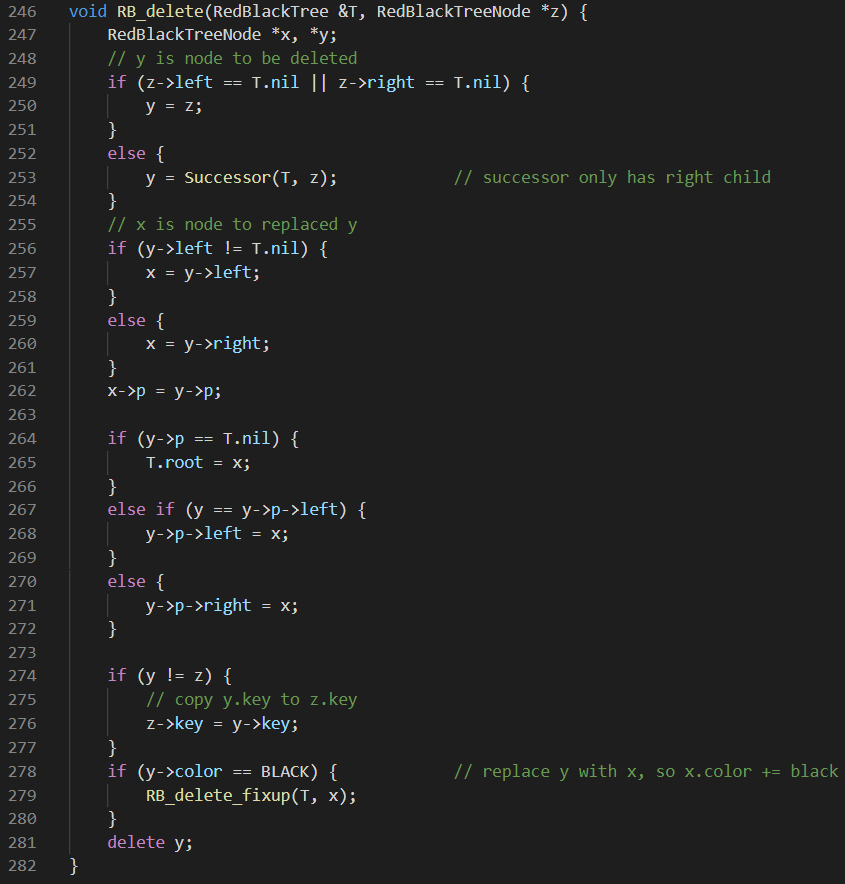
其中两种情况的case1/2/3都已在注释中标明。值得注意的是因为执行case2之后一定会进入case3，所以将case2/3按顺序写在同一个else语句块中。





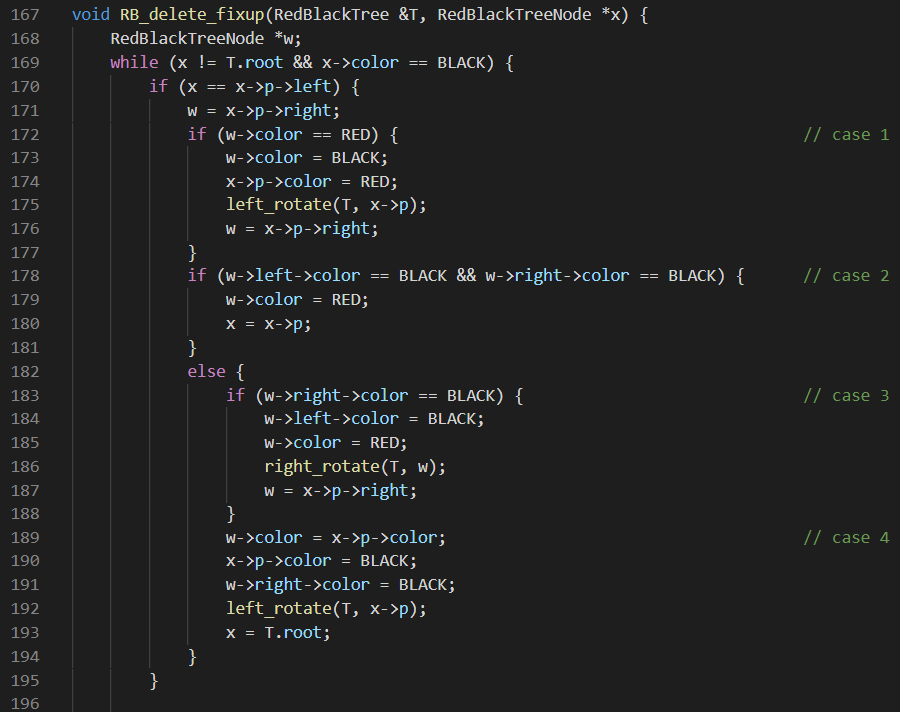
1. 删除

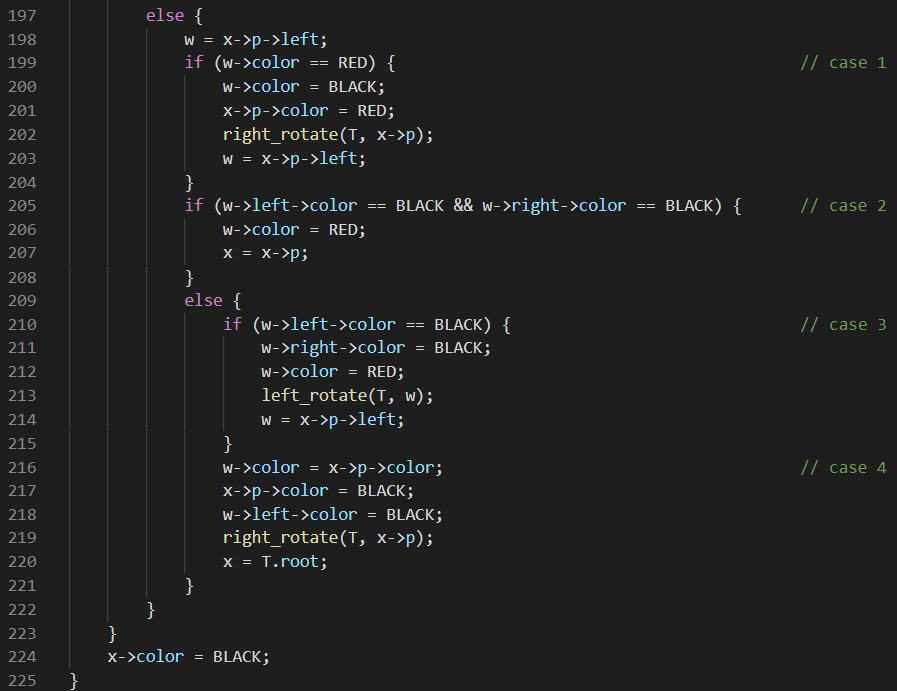
按照第二版书上方法实现删除函数，其中各个步骤已在注释中标明：



1. 删除后修正

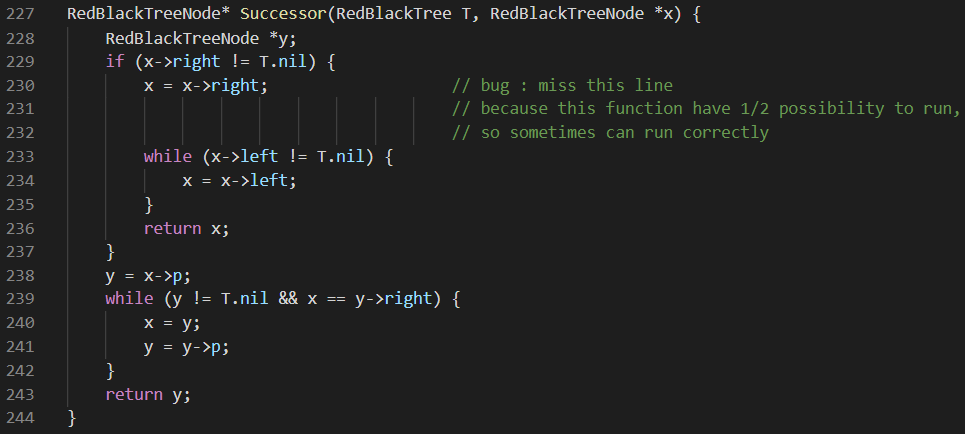
本函数中的case1/2/3/4已在注释中标明，值得注意的是case1执行后有可能进入case2/3/4，所以没有在调整好case1后就进入下一循环。另外，case3执行后一定进入case4，所以将case3/4按顺序写在同一个else语句块中。



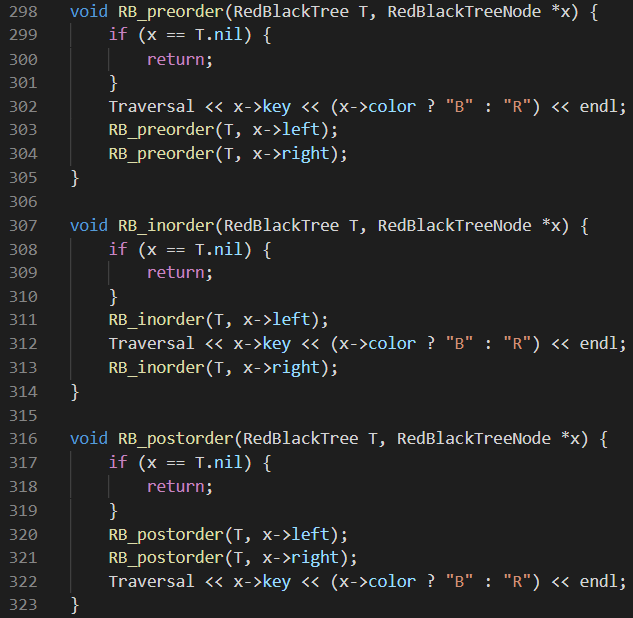


1. 寻找节点的中序遍历后继

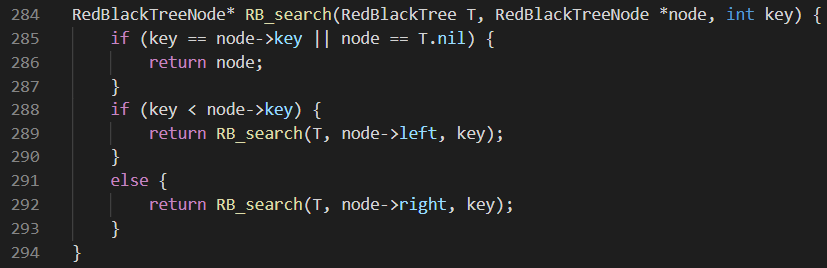
在这里我写错了一行代码，但是因为在delete函数中只有1/2的概率执行本函数（1/2的节点为非叶子节点），所以这个bug一直时隐时现。



1. 遍历



1. 精确查找（用于删除，返回指向查找节点的指针）

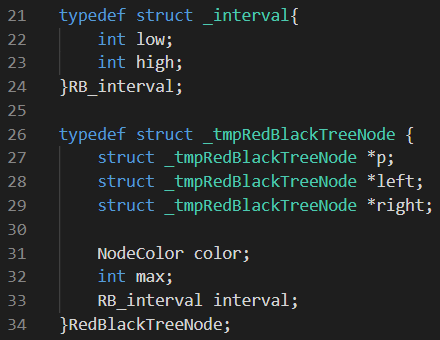


1. **Ex2 区间树**

区间树的代码与红黑树类似，只是增加了对节点max值的维护，

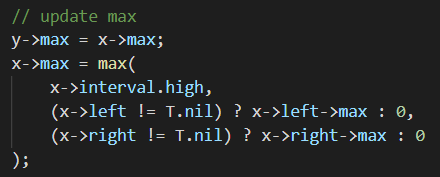
下面为与红黑树有区别的代码：

树的结构增加max值，将key换为结构体“区间”，其包含左端点和右端点，把左端点当key建树。



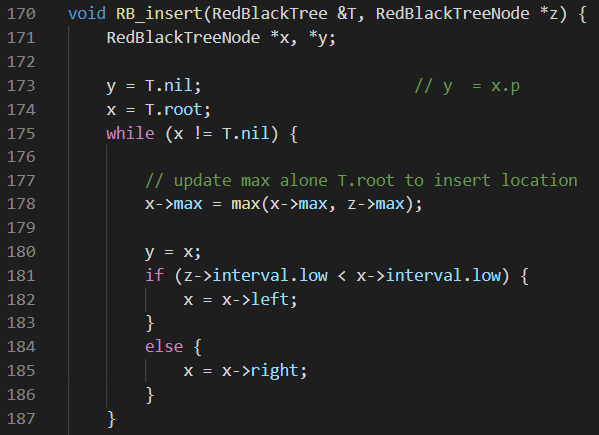
1. 左旋与右旋

在左旋、右旋函数最后增加代码如下：



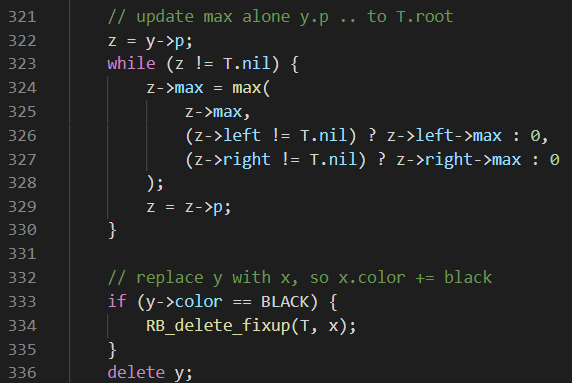
1. 插入

在图中如下位置（沿T.root向下走的路径上）加入更新max值的语句：



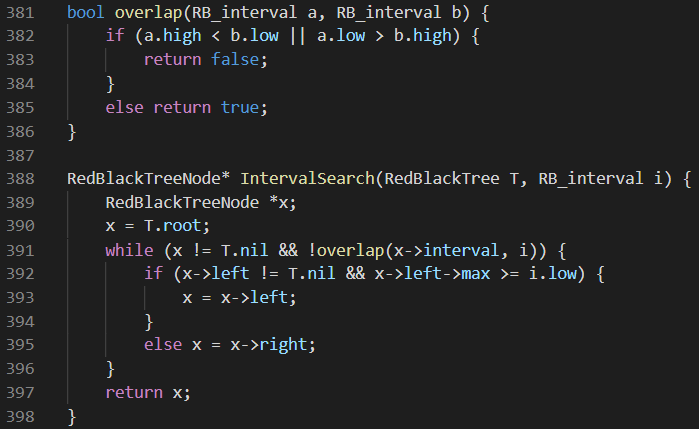
1. 删除

在调用fixup函数前加入更新max值的语句，从实际要删除的y节点一路向上更新max，直到T.root：



1. 区间查找

其中Overlap函数用于判断两个节点是否重叠。

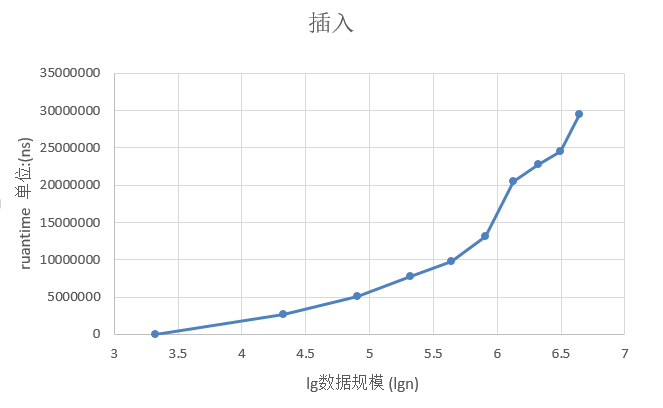


1. **实验结果及分析**
2. **Ex1 红黑树**

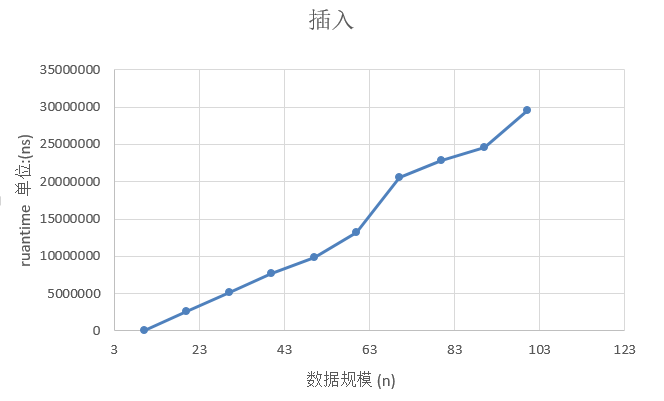
将size20/40/60/80/100中测出的运行时间相对应取平均值，期望插入和删除的运行时间都是○(logn)，所以对n以2为底取对数，处理结果如下：



1. 插入



将lg(n)-运行时间绘成图表发现，曲线的形状与线性有些差别。再将n-运行时间绘成图表



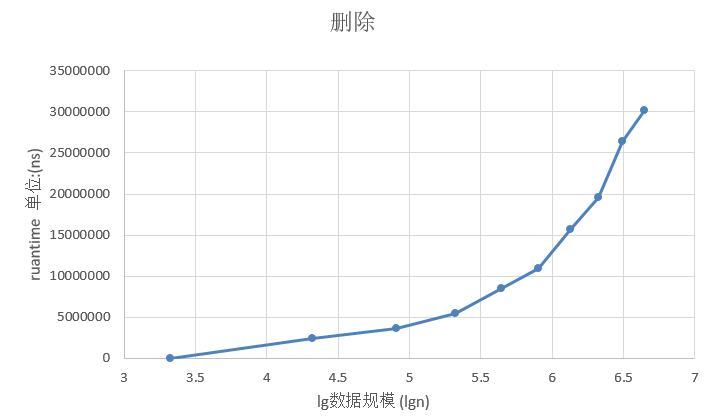
发现运行时间与n几乎成线性，与期望不太相符。

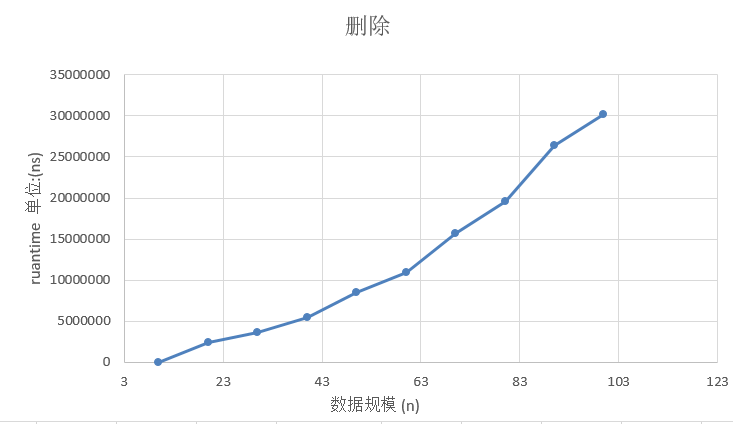
为了探明原因，使用C++中库的红黑树（set的内部实现为红黑树）测试相同的数据，发现其运行时间也几乎成线性，与上图一致。于是推测：红黑树在节点数较少（10-100）时，其插入删除操作的运行时间与n近似成线性关系（类似对数函数曲线的局部放大曲线）。

分析原因：

1. 在节点数较少时，红黑树插入删除操作之后调用的fixup函数运行时间可能与插入删除运行时间相当，使运行时间○(logn)在不同的范围有不同的常数级别的增加。
2. 在节点数较少时，运行时间可能受到硬件与操作系统影响较大，数据从cache中调入调出对运行时间的测量有影响。
3. 删除

lg(n)-运行时间与n-运行时间的图表如下，分析造成这种情况的原因与插入相同：





造成这种情况的原因与插入部分中分析的相同，即节点数较少时fixup函数运行时间和硬件影响较大。

1. **实验总结**

总结一下本次实验中收获的经验：

1. c++中面向对象，默认函数参数传递对象为值传递，非引用传递。
2. 对一个输入流ofstream.open后需要close，才能open下一次。
3. 插入删除fixup函数中最后两个case的执行一定是按顺序的（如插入中先执行case2，之后一定进入case3），所以最后两个case写在同一个else语句块里面。
4. 找节点后继函数写错了，发现bug时隐时见：因为只有约1/2的节点为叶节点，所以删除里面找后继的语句只有约1/2的概率运行。