华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析　　**年级：**22 级　　**上机实践成绩：**

指导教师：金澈清　　**姓名：**郭夏辉

上机实践名称：红黑树　　**学号：**10211900416　　**上机实践日期：**2023.04.27

上机实践编号：No.9　　**组号：**1-416

一、目的

1．熟悉算法设计的基本思想
2．掌握构建红黑树的方法

二、内容与设计思想

1. 编写随机整数生成算法，生成 S 到 T 范围内的 N 个随机整数并输出；
2. 编写红黑树构建算法，中序遍历各节点，输出颜色和值；
3. 随机生成 1e2、1e3、1e4、1e5、1e6 个**不同的**数，使用红黑树构建算法，并画图描述不同数据量下的运行时间差异；
4. （思考题选做）对比红黑树和普通搜索二叉树在不同情况下插入和查找的性能差异。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. 写出红黑树构建算法的源代码
2. 使用合适的图表表达你的实验结果

红黑树是一棵二叉搜索树，它在每个结点上增加了一个存储位来表示结点的颜色，可以是 RED 或 BLACK。通过对任何一条从根到叶子的简单路径上各个结点的颜色进行约束，红黑树确保没有一条路径会比其他路径长出 2 倍，因而是近似于平衡的。

树中每个结点包含 5 个属性：*color*、*key*、*left*、*right* 和 *p*。如果一个结点没有子结点或父结点，则该结点相应指针属性的值为 NIL。我们可以把这些 NIL 视为指向二叉搜索树的叶结点（外部结点）的指针，而把带关键字的结点视为树的内部结点。

一棵红黑树是满足下面红黑性质的二叉搜索树：

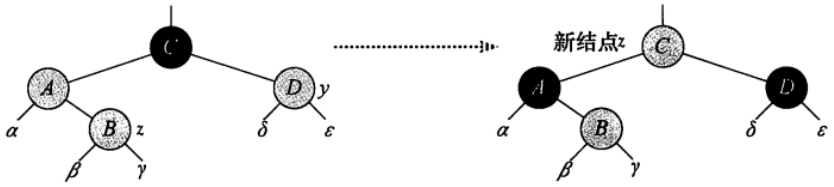
1. **每个结点或是红色的，或是黑色的。**
2. **根结点是黑色的。**
3. **每个叶结点(NIL)是黑色的。**
4. **如果一个结点是红色的，则它的两个子结点都是黑色的。**
5. **对每个结点，从该结点到其所有后代叶结点的简单路径上，均包含相同数目的黑色结点。**

红黑树是一种平衡的二叉搜索树，可以保证在最坏的情况下基本操作的时间复杂度是 O(lgn).

在课堂上，老师着重讲解的是红黑树的查找和插入操作，今天的实验我也是主要围绕这两者展开的。红黑树的查找操作很简单，因为本质上还是一棵二叉搜索树，所以 BST 的性质可以直接拿来使用。这其实是一个类似于二分查找的操作，因为 BST 的左子树值均小于根结点的值而右子树的值均大于根节点的值，所以为了搜索对应的结点，我们从整个 BST 的根节点开始，不断迭代，如果当前结点值为目标值，则直接返回，如果当前节点值小于目标值，则去右树中迭代寻找，大于目标值则去左树迭代寻找。

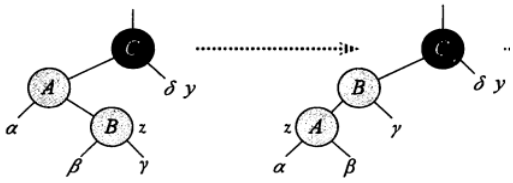
红黑树的插入操作，核心的思想还在于维持红黑树那 5 个性质的不变，具体的流程其实在《算法导论》上有详细的介绍了，这里我并不打算赘述，只是精练地概括一下相应的流程。最开始，要插入的结点按照类似普通 BST 插入操作插入到近似的位置，然后被赋为红色，这个结点被设为当前调整的结点 z。一直向上调整 z，直至不用调整(已经满足红黑树的 5 条性质或者 z 已经抵达整个红黑树的根部)。

- z 的叔结点 y(z 的父结点的父结点的另外一个子结点)为红色



在这样的情况下，z 的颜色不变(还是红色)，z 的父节点和 y 变为黑色，z 的父结点的父节点由黑色变成红色。最后 z 上移两层。

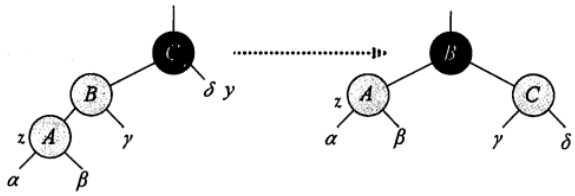
- z 的叔结点 y 为黑色且 z 是一个右孩子



在这样的情况下，以 z 的父节点为中心左旋，颜色不变，但 z 变成了原先 z 的父节点。

- z 的叔结点 y 为黑色且 z 是一个左孩子

在这样的情况下，z 不变，但是以 z 的父节点的父节点为中心右旋，z 先前的父节点由红色变为黑色，而 z 先前的父节点的父节点由黑色变为红色。



三种情况下的不断向上调整，直至红黑树的性质完全满足了。具体的分析和证明还是要参考课本，相关的代码我在附加的代码已经写好了。

五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

随机生成 1e2、1e3、1e4、1e5、1e6 个不同的数，使用红黑树构建算法，并画图描述不同数据量下的运行时间差异。这个任务还是比较简单的，重点是要生成不同的数据，方法和上次实验类似，使用 random_shuffle 对数据进行打乱，再取前 N 个数就可以了。

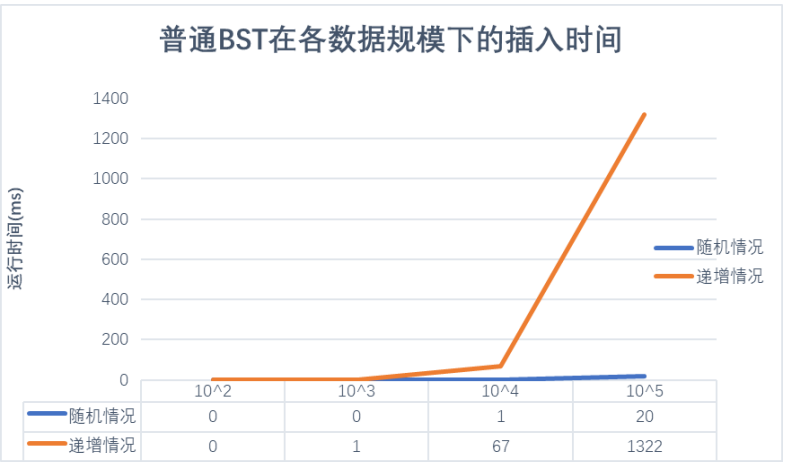
当红黑树有 n 个结点时，插入一个新结点的时间复杂度是 O(lgn)，结合红黑树的性质，通过归纳法可以证明构建 n 结点红黑树的时间复杂度为 O(nlgn)，实验的结果与理论可以相互印证。

（思考题选做）对比红黑树和普通搜索二叉树在不同情况下插入和查找的性能差异。

我们先来进行一些理论分析。普通二叉搜索树在最好情况下插入、查询和删除操作的时间复杂度都是对数级的 Θ(lgn)。二叉搜索树各个操作的时间复杂度取决于树的深度，不同的插入顺序可能会得到完全不同形态的树状结构——对于一个已经有序的序列的插入会退化成一个单链表。而红黑树能保证二叉搜索树的深度为 O(lgn)，确保了相关的基础操作时间复杂度维持在对数级 O(lgn)而不会退化。

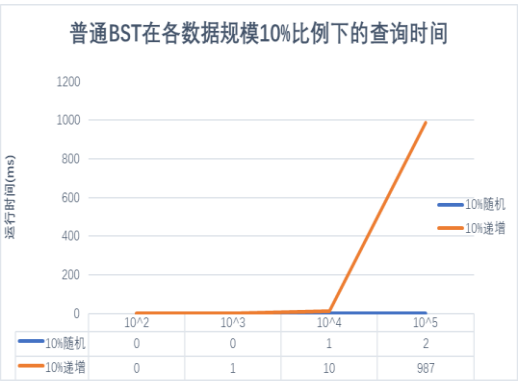
为了展现不同的情况，也为了简化相关的过程，我选择递增数据作为额外的数据集（其实递减数据也是一样的，只要是有序，无论递增还是递减，原理是类似的）。

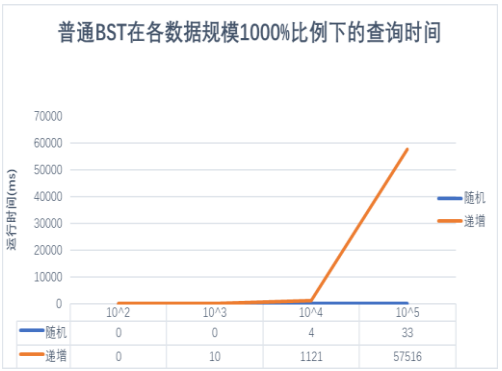
让我们先来考察两种树的插入时间:



结合插入的过程(一个类似于二分查找的操作将要插入的结点放到对应的位置)，插入的时间复杂度由二叉搜索树的形态而定。可以看到，在有序的数据插入二叉搜索树时，普通 BST 的形态由于退化为链式结构而显著地增大了；但是与此同时采用红黑树插入时间却并没有什么显著的变化。

接下来我们再看一下两者查询的时间。为了便于观察，我分别选择了每种数据规模下的 10%，100%，1000%数量的数据作为查询集合。





由于篇幅实在有限，所以我不得不采用上述的方法来呈现结果。(PDF 文档中可以任意缩放，这样可以看清具体的细节)。通过对比可以发现，红黑树在随机情况和递增情况（数据有序）时查询时间并没有显著的差异；与之相对比的是普通 BST 的查询时间在随即情况时也是较为平缓的，但是到了递增情况时就“爆炸式”地增长。

究其本质，还是在于决定二叉搜索树各项基本操作的时间复杂度的重要因素是树的深度，采用红黑树、AVL 树等操作的 BST 可以确保树的深度为 O(lgn)，从而保证了像普通 BST 那样有序数据输入时退化为单链表的恶劣情况不会发生。在未来的学习过程中，希望自己能最大化利用红黑树的价值！