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**：石季凡

上机实践名称：二叉搜索树　　**学号**：10225501403　　**上机实践日期**：2023.4.25

上机实践编号：No.8　　**组号**：1-403

一、目的

1．熟悉算法设计的基本思想

2．掌握二叉树搜索的基本思想，并且能够分析算法性能

二、内容与设计思想

1. 编程实现二叉搜索树，能根据给定字符串（50,20,80,null,null,60,90,null,null,null,100）构建对应二叉搜索树，实现二叉搜索树的各种操作，包括先序遍历、取最小值、取最大值、搜索节点、获取前驱节点以及获取后驱节点。

2. 根据二叉搜索树的中序遍历，输出二叉搜索树的先序和后序遍历结果。

3. 选择合适的数据规模，计算二叉搜索树在不同数据量下查询操作的所需时间。

4. 思考题：对比二叉搜索树、顺序访问和散列表的性能。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. *写出算法的源代码；*

2. *以合适的图来表示你的实验数据*

五、总结

首先根据《算法导论》实现各个函数算法

[插入]：将目标元素依次与 tmp 指针的 key 值进行对比，若大于则 tmp 传递至右子，反之则左子，直到 tmp 指针指向 NULL，同时用一 p 指针记录每一步 tmp 的父指针，这样在 tmp 指向 NULL 时，根据目标元素大小建立新树权为 p 的右子或左子。

[先序遍历]：即优先打印本节点 key 值，然后先递归至左子再递归右子。中序遍历与后序遍历其实本质是调整打印与递归的顺序。即中序为先递归左子树，再打印当前节点，最后递归右子树，后序遍历为先递归左子树，再递归右子树，最后打印当前节点。

[取最小（大）值]：从根部出发沿左（右）子一直行进，直到 NULL

[搜索节点]: 从根部开始不断比较，大于则行进至右子，反之则左子，直到 key 值与 target 相等，返回该节点。

[获取前驱节点]：搜索到目标节点，然后开始寻找比该节点小的最大节点，即右分支中的最大值，但是如果左子为 NULL，则寻找其父代，若其为父代的右子，则返回父代，否则向父代行进，直到找到第一个作为其右子树中元素的节点并返回，若直到 NULL 也没找到，说明其为整个树的最小值，没有前驱结点。

[获取后驱节点]：搜索到目标节点，然后开始寻找比该节点大的最小节点，即左分支中的最小值，但是如果右子为 NULL，则寻找其父代，若其为父代的左子，则返回父代，否则向父代行进，直到找到第一个作为其左子树中元素的节点并返回，若直到 NULL 也没找到，说明其为整个树的最大值，没有后继结点。

下面进行搜索实验，即在不同的数据量下检测搜索的耗时

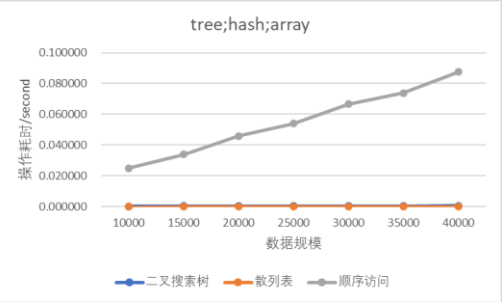
数据量：10000 15000 20000 25000 30000

搜索量：1000

进行实验的数据结构有二叉搜索树、散列表、顺序访问

首先是实验数据

	二叉搜索树	散列表	顺序访问
10000	0.000337	0.000040	0.025018
15000	0.000391	0.000042	0.033900
20000	0.000442	0.000044	0.045954
25000	0.000487	0.000049	0.054058
30000	0.000516	0.000048	0.066690
35000	0.000540	0.000054	0.073871
40000	0.000569	0.000066	0.087582



其中时间的单位是秒

通过图像我们可以清晰地看见，由于顺序访问每次搜索都需要遍历所有数据，使其搜索时间与另外二者完全不在同一数量级上，且基本随着数据规模的线性增加而线性增加。

由于该图像对于二叉搜索树和散列表的时间刻画不明显，我将二者另外画图，得到以下图像。

根据以上结果我们其实可以发现，随着数据量的增大，搜索的时间不断增加，同时，如果我们建立树的时候足够幸运（即为平衡二叉树），整个树的高度为 lgN，这样搜索的话每一层只需要经过一次即可，即每次搜索最多只需要搜索 lgN 次即可得到结果。但是由于我们的数据时随机生成的，这导致每一次插入的数据不一定恰好为中位数，这样就导致整个树会不平衡，但是实际上每次搜索的期望仍然为 lgN 次，但是期望如此并不代表最坏情况如此，如果我们足够“不幸运”，即每次按升序或者降序插入，这样我们就会得到一棵高度为 N 的树，这会最终导致搜索操作变得极为缓慢。

而对于插入的优化其实是非常有局限性的。如随机取值插入，首先由于我们插入在实际情况下插入的数据是一步一步操作的，而不是一次给到所有的数据，其次由于我们的数据本来就是随机生成，因此随机取值插入对于实验结果的优化非常局限。而想要本质上改变搜索的耗时就得尽量建立平衡的二叉搜索树。而尽可能建立平衡的二叉搜索树，这就要涉及到后面的红黑树与 val 树了。

对于二者的对比，由于在本次实验中我们的散列表大小足够大（size=200000），所以当我们增加插入数量时，对于装载因子的影响比较小，即发生哈希冲突的可能性并不会明显

增加，因此当我们增加数据规模时，我们可以发现搜索耗时的增长非常小，毕竟由于装载因子增加不多，单次搜索时间也不会增加太多。

对于本次实验的总结，我发现了二叉搜索树一种优秀的数据结构，但是其插入函数存在非常大的优化空间，构建平衡的二叉搜索树是缩短每一次搜索时间期望与最坏情况的必要条件。