AVL 树实验

1. 实验思路：

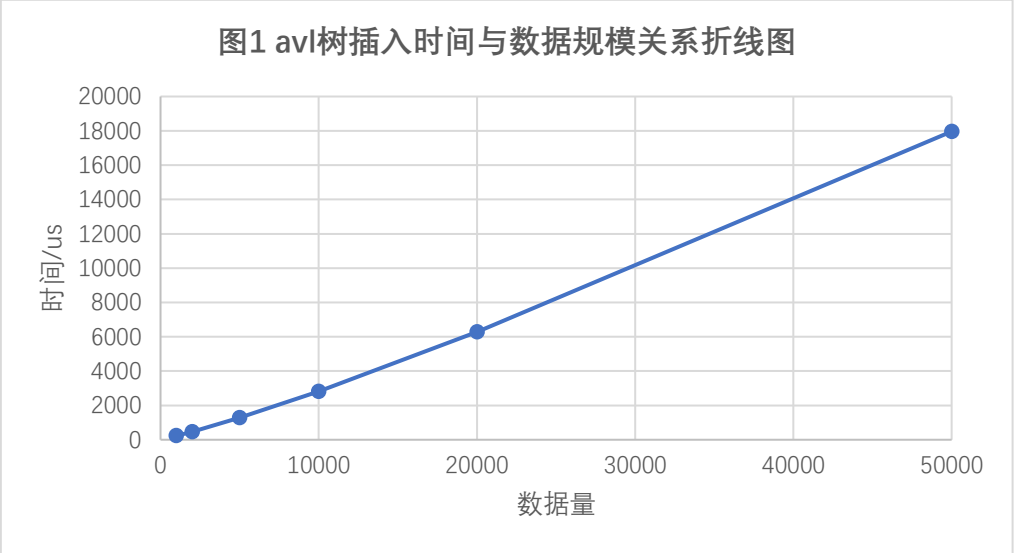
为避免数据差异造成误差，本实验采用 rand()函数生成无重复数字的随机数表，不同实验组均使用该随机数表，顺序插入与逆序插入仅对该随机数表进行相应的排序操作；相同实验组的插入操作重复 50 次，求得平均耗时。

2. avl 树随机插入所需时间

实验中得到的数据如下：

表 1 不同数据规模 avl 树随机插入耗时

数据量	耗时/μs
1000	230.442
2000	467.958
5000	1290.63
10000	2818.01
20000	6285.72
50000	17963.1



3. avl 树与跳表插入性能对比

根据实验要求，在数据规模均为 50000 的条件下，avl 树与跳表的插入时间数据如下：

时间/us	随机插入	升序插入	降序插入
avl 树	17963.1	12592.3	12435.3
跳表	29868.0	14645.3	11909.7

稍后将对以上数据的异常情况进行分析。

4. 数据分析

4.1 avl 树随机插入所需时间成本与数据量的关系

理论表明，n 次插入应符合 nlog(n)的时间复杂度，步骤 3 中的折线增长趋势基本符合.

4.2 avl 树与跳表插入性能对比

4.2.1 随机插入

avl 树 n 次次随机插入的理论时间复杂度为：

$$O_{avl}(n) = \sum_{1}^{n}(log_{1.618}(n) - 1.3277)$$

增长概率为 p=1/2 的跳表 n 次随机插入的理论时间复杂度为：

$$O_{skip}(n) = \sum_{1}^{n}(log_2(2n) + 2)$$

理论上数据规模为 50000 时，跳表的性能优于 avl 树由于两种算法的实际实现存在一定差异，如 avlTree 采用递归方式插入数据，更新每个节点的高度；skipList 每次插入需要动态申请指针数组、调用随机层数生成函数等，程序性能将导致一定程度的差异。

4.2.2 顺序插入与逆序插入

理论上 Avl 顺序插入与逆序插入的时间复杂度相同，本实验中在误差范围内可以认为顺序与逆序不存在明显差异。值得一提的是，在一开始的程序中更新查找树节点的高度时采用条件选择语句，导致顺序插入与逆序插入出现明显的性能差异[1]，经过长时间细致分析，并与同学的交流，最终确定原因并更正。

理论上 avl 树随机插入时执行旋转的次数少于顺序插入与逆序插入，而查找树的增长高度较高，因此性能差于顺序插入与逆序插入，本实验符合。

理论上 SkipList 三种插入方式的性能为：降序插入>随机插入>升序插入，而本实验中降序插入确实优于升序插入，而随机插入性能最差，经过长时间分析讨论，暂时未能确定具体原因。受到[1]的启发，重新对程序进行分析，仍未得出结论。对不同插入方式的插入路径长度的程序计算结果与理论分析吻合，说明以上异常仅由程序性能导致，而非程序实现错误。

4.3 Avl 树与 skipList 的选择：

由于实验数据的局限性，很难做出确定的选择。本实验中仅在逆序的情况下跳表的性能优于 avl 树，并且差距并不明显，仅据此得出逆序插入的情况下跳表更优将有失偏颇。根据理论分析，若随机插入时跳表的性能测试符合理论值，将明显优于 avl 树，因此，理论上随机插入时应选择跳表。关于本实验的异常现象的解释以及更优算法的实现，将在后续继续探索。