红黑树与 Avl 树性能对比

1. 实验思路

根据实验要求，实现红黑树与 Avl 树的插入与查询算法。为排除程序实现差异造成的影响，两种数据结构统一采用 bottom-up 方式、34 重构方法，维护并更新节点的 parent 指针、（黑）高度，代码框架基本相同。

2. 随机插入与顺序插入

共设 5 组输入集，分别存于 data 文件夹的 randomN.txt. 5 组数据规模分别为：50，500，2000，5000，10000，数据范围均为为 1-50000. 分别测试红黑树与 Avl 树的插入性能，重复实验 100 次取平均值，得到以下数据：

表 1 插入操作执行时间

	随机插入(μs)		顺序插入(μs)	
数据规模	Avl 树	红黑树	Avl 树	红黑树
100	17.972	11.934	16.054	11.544
500	84.482	67.106	74.968	70.23
2000	401.084	320.902	290.982	286.056
5000	1140.22	869.756	692.056	693.63
10000	2509.24	1818.80	1431.37	1541.45

图 1 插入操作执行时间折线图

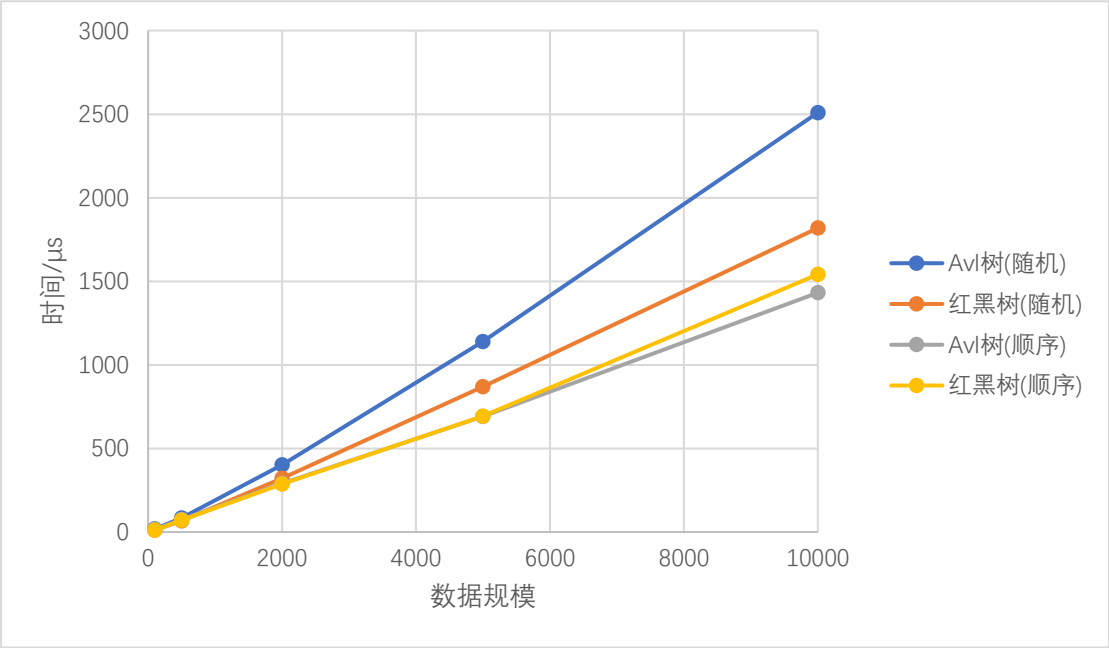
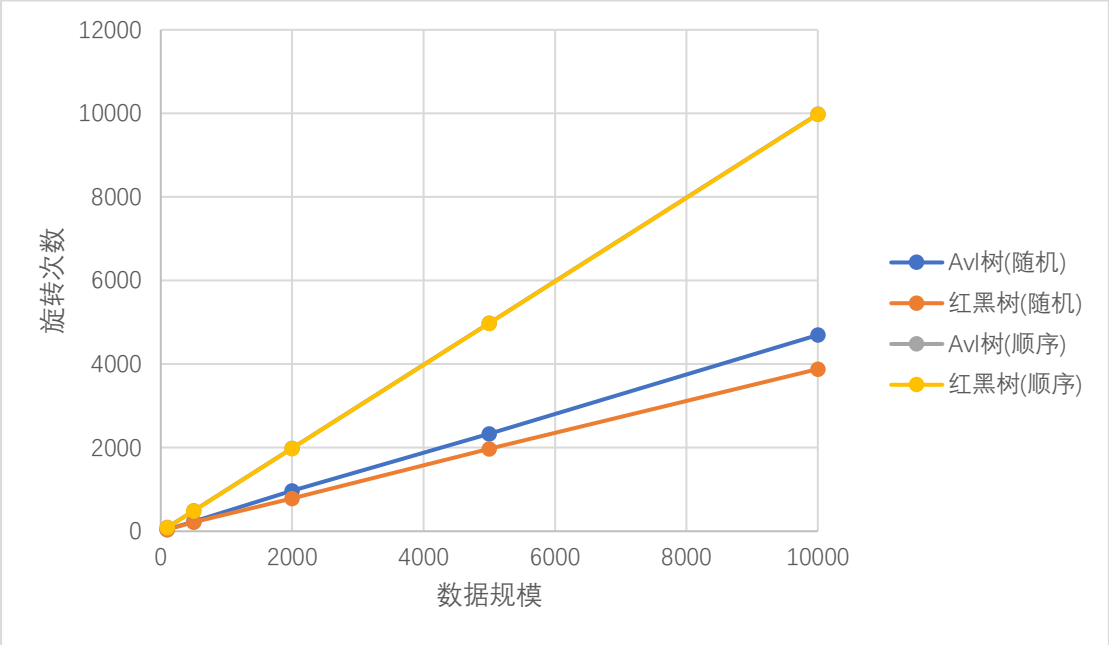


表 2 插入操作旋转次数

	随机插入		顺序插入	
数据规模	Avl 树	红黑树	Avl 树	红黑树
100	44	41	93	89
500	236	215	491	485
2000	968	789	1989	1981
5000	2330	1971	4987	4978
10000	4697	3881	9986	9976

图 2 插入操作旋转次数折线图



根据理论分析，Avl 树的平衡条件更苛刻，导致随机插入过程中旋转次数较多，因而时间成本高于红黑树；顺序插入时二者基本一致。实验数据与理论基本符合，随机插入时 Avl 的旋转次数多于红黑树，时间成本高于红黑树；顺序插入时 Avl 旋转次数与红黑树基本一致（红黑树略小于 Avl 树），时间成本二者并无明显差异，时而红黑树高，时而 Avl 树高。两种数据结构均表现为随机插入性能差于顺序插入，但红黑树的上述差异较小。

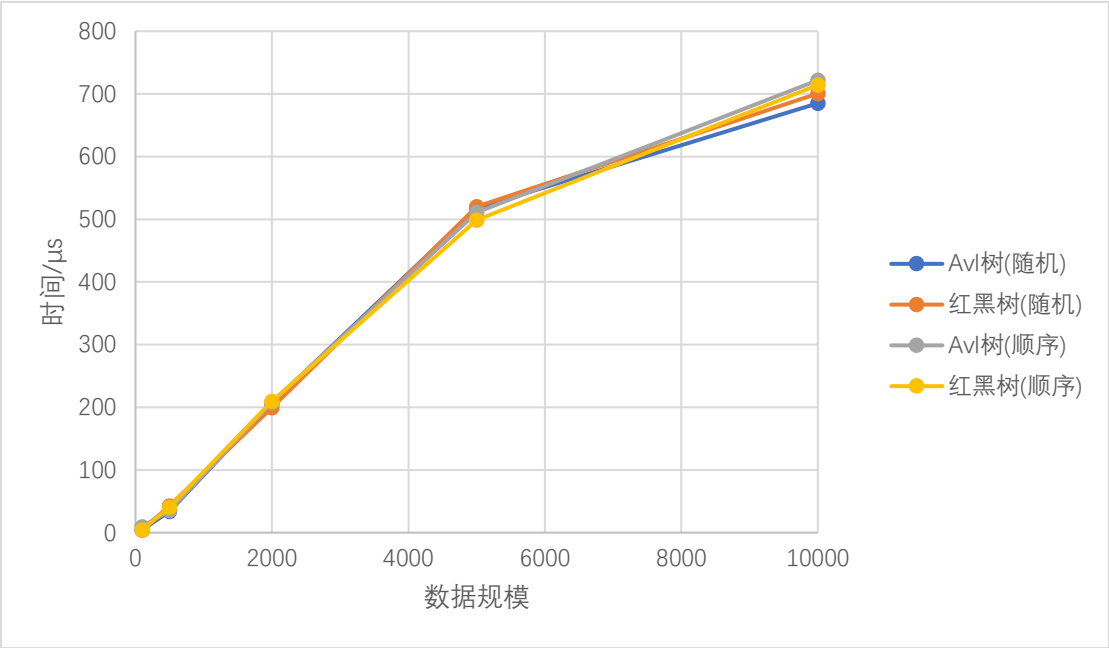
3. 随机查找与顺序查找

针对数据规模为 10000 的输入集，共设 5 组查找集，分别存于 data 文件夹的 searchN.txt 中（查找集与输入集在同一数据范围内，查找不一定命中）。查找集的规模分别为 100，500，2000，5000，8000. 重复实验 50 次取平均值，得到以下数据：

表 3 查询操作时间成本

	随机插入(μs)		顺序插入(μs)	
查询规模	Avl 树	红黑树	Avl 树	红黑树
100	4.518	4.562	9.66	4.5
500	33.73	42.548	35.602	40.654
2000	206.552	199.422	207.788	209.524
5000	517.644	520.056	511.14	498.674
8000	685.088	700.674	721.74	714.204

图 3 查询操作时间成本折线图



根据理论分析，由于 Avl 树平衡条件更严格，树的高度应低于红黑树，查询的平均时间应优于红黑树，而实验结果反映为二者基本一致，且随机插入与顺序插入并无明显区别。当扩大数据规模至 100000 后仍然出现以上现象，说明在目前所尝试的数据规模范围内，查询路径长度反应在时间成本上的变化不明显，若继续增加数据规模，如 10^6、10^7 等，可能会有所区分。

4. 对两种数据结构差异的理解

Avl 树和红黑树的目的均是通过一些调整使得树的高度不至于出现最坏情况，达到较为理想的“平衡”，不同的是 Avl 树对上述平衡的限制更严格，即每个节点左右子树的高度差不得大于 1，而红黑树的限制条件为左右子树高度相差的倍数不超过 2 倍。因此，Avl 树将耗费更多的资源在树结构的调整上，导致随机插入比 Avl 树性能差，而从上面的实验可以看出，红黑树对平衡条件的放松并未导致数据查询性能与 Avl 树出现显著差异，因此，红黑树在更多情况下具有更强的实用性，也因此成为目前使用最广泛的平衡树。