std::set 嘲讽我，std::set 超越我

范昊翀

2024 年 6 月 1 日

1 前言

这次我实现了三种动态查找表数据结构，按实现顺序分别为 AVL 树，红黑树，以及可持久化无旋 Treap，为了测试它们三者的效率，我设计了多个测试程序，比较它们的运行时间。

2 N=3e6，纯插入，纯删除

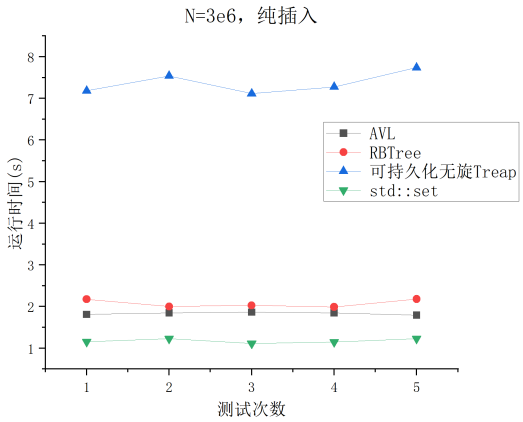

图 1: 插入操作测试

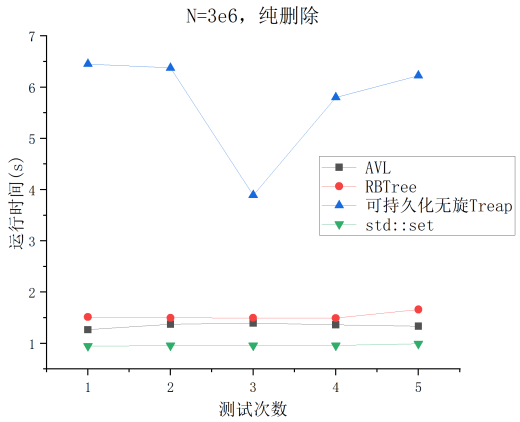

图 2: 删除操作测试

观察到有 std::set < AVL < RBTree « Persistent-FHQTreap，在接下来的测试中，由于 Treap 支持可持久化操作，会因为每次操作都要加点，会导致效率变得奇低。另外比较奇怪的一点是，我写的 AVL 居然会比红黑树快？按理说 RBTree 应该比 AVL 对插入删除操作更加友好？因为打破平衡的次数应该更少，可能是我的实现方式不够优雅。此外，由于 Treap 依赖随机数，效率不稳定，比如在第三次删除测试中就出现了一个“好点”，但总体效率还是更低的。

3 N=3e6，++it

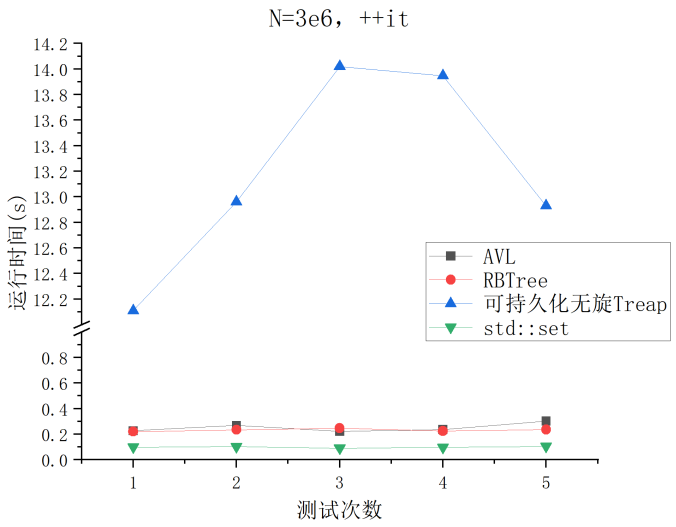

图 3: 迭代器操作测试

又被 std::set 嘲讽了，AVL 和红黑树的效率这次差不多，都是在树上走路，均摊复杂度的期望是 $O(1)$，而可持久化无旋 Treap 每次都要 split 再 merge，复杂度稳定在 $O(logn)$，比另外三种结构慢情有可原。

4 N=1e5，先插入，再复制 1000 次

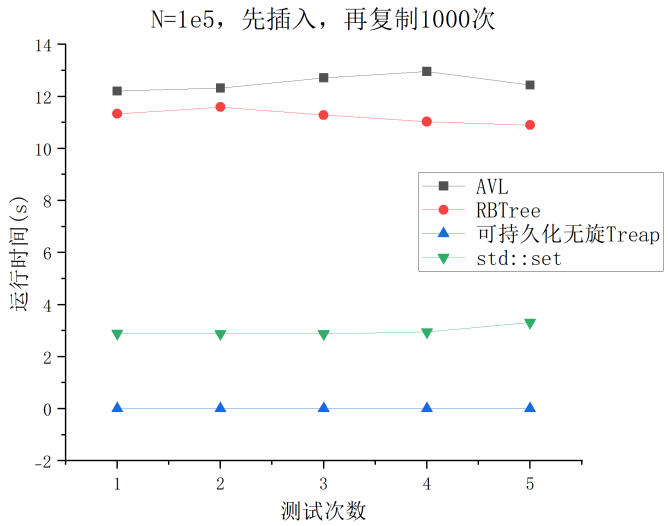

图 4: 复制操作测试

一方面，我成功嘲讽了 std::set，因为显然我的可持久化无旋 Treap 在复制时复杂度是 $O(1)$ 的，这比 std::set 会快许多，但为什么我的 AVL 和红黑树，同样是没有可持久化，比 std::set 要慢这么

多?（每次复制复杂度是 $O(n)$ 的）

5 结论

据我猜测，std::set 内部可能进行了某些懒操作，即如果我没有访问某些元素，那关于这些元素的操作永远不会被下放，这从复制操作中或许可以看出来，因为如果同样是遍历并拷贝树结构，效率不该差太多。如果说哪种结构比较适合实现 set，我感觉我的红黑树和 AVL 在效率上几乎没有什么区别。理论上来看，AVL 层数更小，对 find 更友好，红黑树调整更少，平衡条件相对宽松，对 insert 和 erase 更友好，但实际操作下来反而 AVL 更快一些。