RBTree

实验数据

有序插入

RBTree\数据量	1000	2000	5000	10000	20000
用时（ms）	0.7583	1.2596	3.2369	5.8755	12.4905
旋转次数	983	1981	4978	9976	19974

AVLTree\数据量	1000	2000	5000	10000	20000
用时（ms）	1.0976	1.8264	4.3089	10.1845	18.0233
旋转次数	1980	3978	9974	19972	39970

无序插入

RBTree\数据量	1000	2000	5000	10000	20000
用时（ms）	0.6614	1.2288	3.1098	6.2137	14.6379
旋转次数	629	1203	2947	5788	11802

顺序查找用时

数据量	1000	2000	5000	10000	20000
RBTree用时（ms）	0.3799	0.7483	1.8641	4.6561	7.498
AVLTree用时（ms）	0.3731	0.7845	1.8774	3.8282	8.2325

乱序查找用时

RBTree\数据量	1000	2000	5000	10000	20000
RBTree用时（ms）	0.4523	0.9308	2.5015	5.4003	9.2223
AVLTree用时（ms）	0.419	0.8487	2.1418	4.7749	9.539

实验结论

- 插入
  - 无论在乱序还是顺序插入，AVL树和RB树的性能都较为相近，总的来说RB树更好一点，这点符合二者查找时间复杂的O（logn）的理论分析
  - 顺序插入时，RB树的旋转次数少，而无序插入时，AVL树的旋转次数少且大大低于有序时的次数
- 查找
  - 二者在顺序和乱序下的查找性能相近，但是AVL树相对更好，这可能是由于红黑树并非严格的平衡二叉树，因此树高更大，查找次数可能更多，符合理论分析

对两种数据结构的理解：

- RB树牺牲了一部分的平衡性来换取插入删除结点时旋转次数的减少，但AVL树是严格的平衡二叉树，因此用于平衡的开销会更大,但同时树高一般更小，故在查找是AVL树一般性能更好，而在插入删除时一般RB树更好
- 因此对两种树的选择取决于业务需求是偏向于频繁的查找还是更多的增删，例如在C++的STL中，不少实现便采用了RB树