HW4 RB-TREE

一、

1. 主要是为了**维持红黑树的性质不改变**。如果插入的节点默认为黑色，那么性质 5（所有叶子节点的黑深度相同）一定会被破坏，并且需要较大的代价去重新维护这个性质。如果插入的是红色节点，那么性质 1,2,3,5 都是不会被破坏的，只有性质 4（红色节点的儿子必须是黑色的）有可能改变，但是通过简单的旋转和变色操作就可以重新恢复。因此我们选择默认插入红色节点，接下来的恢复性质的操作也只需要针对性质 4 进行分类；

2. 参考课本上的分类，将插入节点 x 的父亲和祖父记为 p 和 g，叔父记为 u。因为 p 是红色的，由性质 4 知道 g 是黑色的。

- 若为 LL 型：

  p 染黑，解决连续红色节点问题，但是导致左子树的黑深度多 1；

  对 p 和 g 右旋，同时将 g 染红，解决失衡问题同时保持了性质 4。

- 若为 LR 型：

  先对 x 和 p 进行左旋，子树形状变成了 LL 型，随后同 LL 型的操作进行恢复。

- 若为 RL 型和 RR 型：

  与 LR 和 LL 型完全对称，同样方式恢复。

个人认为课本上的 43 重构理解方式更为简洁，从 B 树的角度来看，无论哪种情况都是父节点关键码中红色相邻导致的，只需要调整颜色即可，对应到红黑树中的操作就是进行一次 43 重构。

二、

红黑树的设计理念应该就是把一棵 24-树转换成等价的二叉树。

从性质上看，红色黑色用于区分父子关系和兄弟关系，把多叉树转换成了二叉树；红色节点不相邻的性质是用于保证 24-树的关键码的数量限制的，既保证了关键码至少为 1（显然），又保证了不超过 3（因为最多是红黑红结构）；黑深度相同的性质则保证了红黑树能够对应映射一颗 24-树，即所有外部节点在转换为 B 树之后处于同一个实际深度。

从操作上看，不论是插入还是删除操作，都可以通过对应的 24-树进行理解，以插入中的 RR-2 情况（u 为红色）为例，等价于 24-树中的上溢情况，对应的操作也就是 B 树中的上提关键码分裂当前节点，上溢导致的递归操作也是可以从 B 树的角度理解的。

从我个人的理解来看，作者创造红黑树这个数据结构的初心应该是把 B 树和 AVL 数的优点杂交保留下来，形成一个平衡操作代价比较小，不需要维护额外变量，同时又能保留 B 数的最大高度较小的优点。

三、

顺序插入会导致红黑树的**旋转操作变多**，从结构上看，从小到大的数据插入会导致红黑树的右半部分始终大于等于左半部分的深度，红色节点也始终出现在右边，而插入的新节点又在最右端，因此经常需要进行旋转操作来维护性质 4；但是顺序插入的**耗时应该会比较少**，因为顺序插入对于缓存的命中率有利，根据数据局部性原理，系统应该会把刚刚插入的数据附近的空间放入缓存，减少了下一次插入操作的耗时（仅为猜测，不是很了解）。

相反地，乱序插入应该会导致旋转操作比较少，但是耗时比较多。