二叉搜索树(BST)

我们能不能做到高效率的兼顾**动态修改**和**静态查找**？

概念与核心思想

call by key——根据关键码而彼此区分

关键码需要支持比较（大小）和比对相等。

数据+其他乱七八糟的附加信息，统一包装成词条 `entry`

顺序性

在任一节点的左（右）子树中，所有节点（若存在）均不大于（不小于） r

中序遍历必然单调非降。

操作

查找

从根节点出发，逐步地缩小查找范围。直到发现目标（成功）或缩小到空树（失败）。

成功时，返回找到的（真实存在的）节点；失败时，返回最后到达的 NULL 节点的引用（应该是可以改的）；

失败时，不妨假设这个 NULL 节点是一个等效的节点，值为我们要查找的 e 。

【需要记录父亲节点 _hot，不过一定程度上也可以用引用传指针变量解决】

插入

直接插在最底层，记录的父亲 _hot 底下查找到的那个 NULL 位置。

删除

查找被删除节点。

- 如果待删除节点有一个子节点/没有子节点，就接上去；
- 如果待删除节点有两个子节点，**在右侧节点找到后继，把后继与待删除结点交换后，删掉后继**，把待删除节点的值替换成后继节点的值。

平衡二叉搜索树(BBST)

在最坏情况下，树高会是 $O(n)$

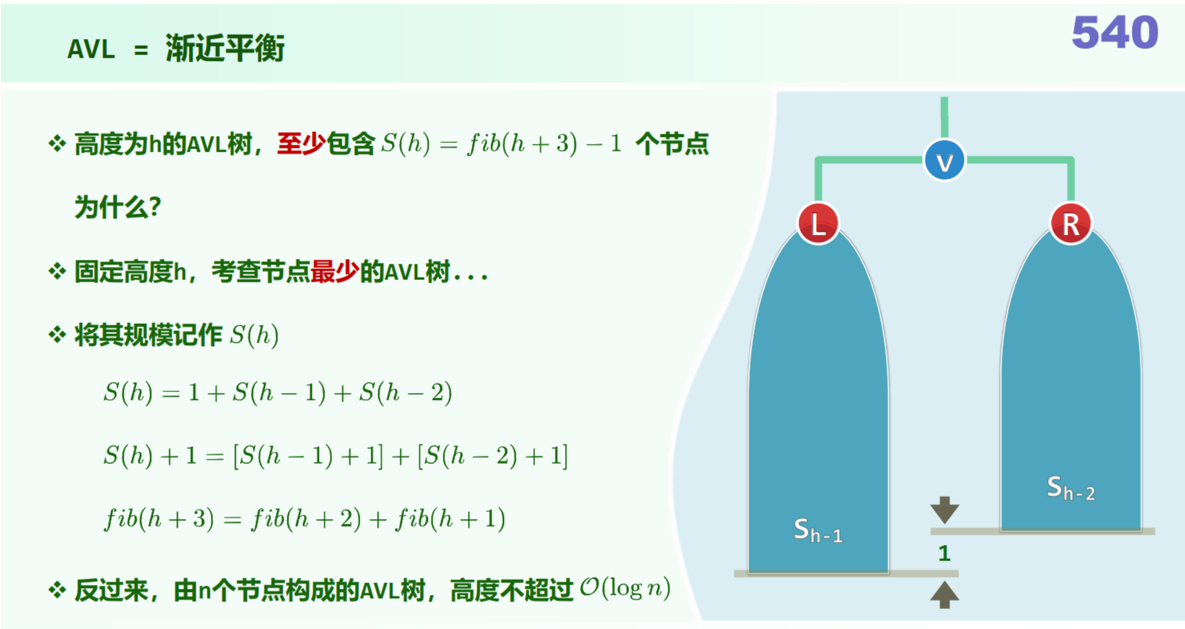
期望树高

- 随机生成词条，随机插入树中。各个排列出现概率相等，期望树高 $\Theta(\log n)$ 。
- 互异的词条，在遵守顺序性的前提下，随机确定拓扑联接关系，期望树高 $\Theta(\sqrt{n})$ 。【这样的树一共有 $\text{catalan}(n)$ 棵】

平衡

理想平衡——就是 $h = \lfloor \log_2 n \rfloor$

渐进平衡—— $h = \mathcal{O}(\log n)$ 。



旋转——重要的 BST 等价变换

上下可变 左右不乱

BBST 会给出一些精心设计的平衡限制条件。

单次动态修改操作后，至多 $O(\log n)$ 处局部不再满足限制条件。

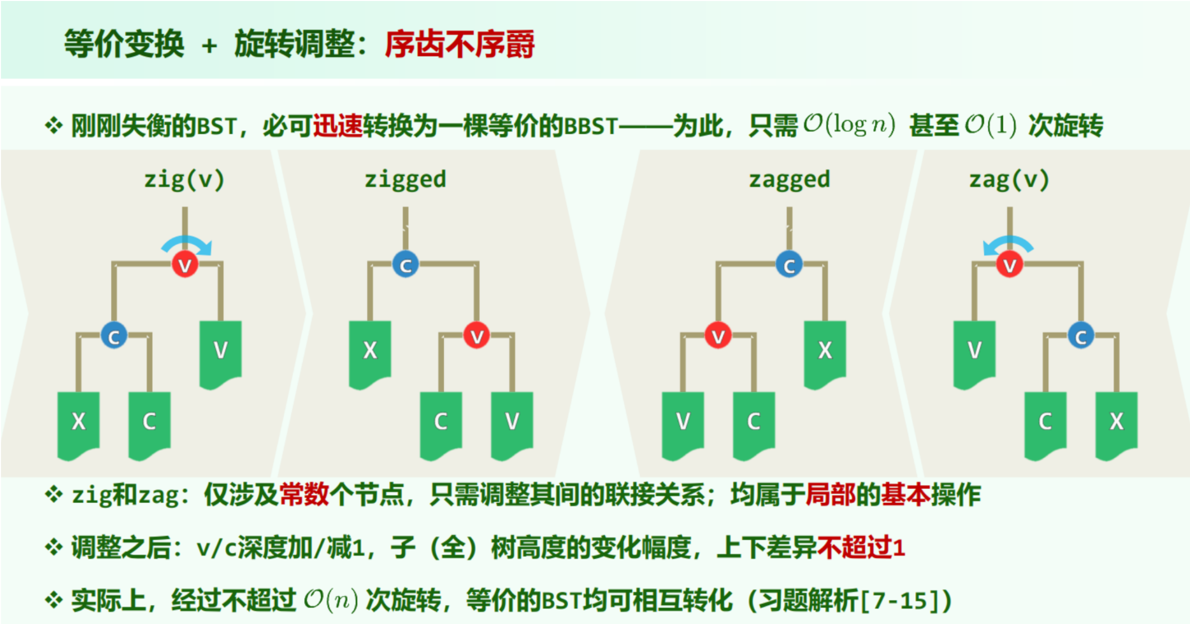
可以在 $O(\log n)$ 时间内，修复这些局部条件。

例子：一颗刚刚失去平衡的 BST，可以在 $O(\log n)$ 或者 $O(1)$ 次旋转内恢复平衡。

事实上，所有的 BST 都可以在 $O(n)$ 次旋转内相互转化。（习题7-15，任何BST都可以在n-1-s次旋转内变为最左侧通路，其中s为初始最左侧通路的长度）

zig（右旋）：将左孩子变为根，原本的根变为右孩子

zag（左旋）：将右孩子变为根，原本根变为左孩子



AVL Tree

平衡策略

插入

可能导致祖先路径上所有结点失衡，但只需要一次调整就能恢复。因为局部子树复衡后，高度必然复原；其祖先亦必如此，故调整结束。如果插入引起了旋转，则树高必然不变，而如果未引起旋转，反而可能树高增加。

删除

同一时间只有一个失衡节点，失衡不会向上传递（旋转可能导致树高降低，因此会向上传播），但可能需要$\mathrm{O}(logn)$次调整才能恢复

3-4 重构

从失衡结点往下找到最长藤上的祖孙三代及其子树，按照中序遍历的顺序重构。

从单旋和双旋的角度，一字型只转祖父，之字型先转父亲再转祖父。

AVL叶结点深度

可以通过归纳证明，Avl树中，任何叶节点的深度不小于 $\lceil h/2 \rceil$