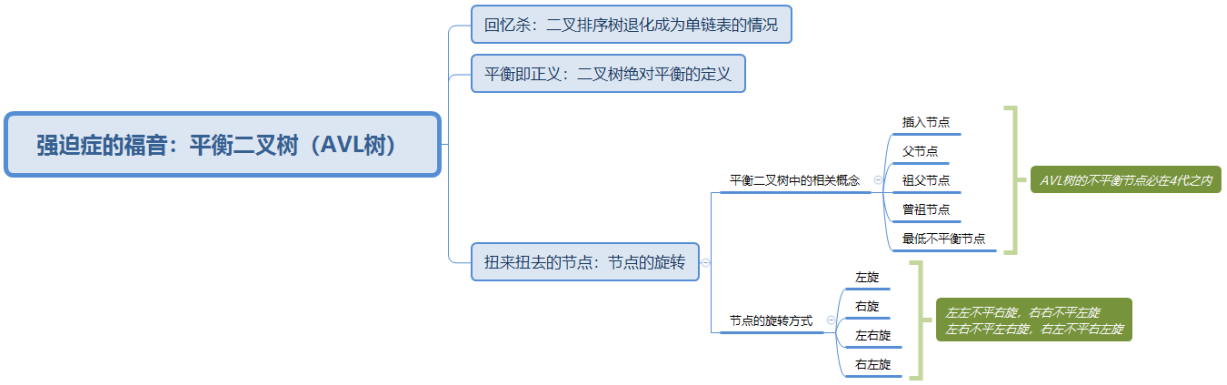


1.强迫症的福音：平衡二叉树（AVL树）

①回忆杀：二叉排序树退化成单链表的情况

之前在上一章节我们已经总结过，在某些特殊情况下，二叉排序树是有可能退化成为单链表的

也就是说，如果一个二叉排序树对已经有序的序列进行排序，那么其中的结构就会退化成为单链表

并且元素的查找效率也会明显下降，那么此时，我们需要通过一些特别的手段，保证这个二叉排序树的“平衡”

进而保证查询元素的效率

②平衡即正义：二叉树绝对平衡的定义

平衡二叉树（AVL树）本身也是一种二叉排序树，其内部的元素之间同样遵从左孩子 < 根节点 < 右孩子或者左孩子 > 根节点 > 右孩子的特性的

但是，在创建平衡二叉树的时候，我们要求树结构中的任何一个节点，左右子树的层数深度之间的深度之差，绝对值不超过1

举个例子：如果在某一个二叉树结构中，某一个节点的左子树的深度是5，则其右子树的深度取值范围在4-6之间，如果超出或者小于这个范围，就不算是平衡二叉树了

那么在向这个二叉树结构中插入新节点，或者删除原有节点的时候，很可能出现导致某节点左右子树不平衡的状况发生，那么此时就需要对指定的节点进行旋转

最终达到让这个节点，乃至整个二叉树结构平衡的目的

下面我们就来讨论一下，向平衡二叉树中添加节点，并通过旋转保持平衡的方式

③扭来扭去的节点：节点的旋转

在我们向AVL树中添加节点，或者删除掉AVL树中节点的时候，难免会造成AVL树原先整体平衡的结构

在某些节点上变得不平衡。那么此时，我们就需要通过对AVL树中节点的旋转，对AVL树的节点结构进行重新调整，

最终AVL树重新回归平衡状态

值得庆幸的是，在添加节点或者删除节点之后，我们最多能够在4代以内的节点之间造成不平衡的关系

所以，我们可以将不平衡的状态按照不平衡节点和其叶子节点之间的深度关系，分为3代不平衡和4代不平衡两种

其中：通过添加和删除节点导致的3代不平衡的情况都比较容易理解和处理，但是一旦涉及到在4代节点之间，

因为添加或者删除元素导致的不平衡的出现，就需要我们分析一些比较复杂的情况了

但是不管情况多么复杂，我们最终都是通过对节点的旋转，来完成对AVL树结构平衡的调整的

而节点的旋转，根据旋转的方向和次数，又可以分为：左旋、右旋、左右旋、右左旋四种

下面，我们就来根据上面说到的这些情况，逐一讨论如何调整AVL树结构的平衡性

1.平衡二叉树中的相关概念

为了方便我们在后面的学习中，能够快速理解一些结构

现在我们对AVL树中的节点，左如下定义：

(1).插入节点

插入节点指的是通过添加操作，新加入AVL树结构中的节点

在向AVL树中添加插入节点的时候，不一定会引起结构的不平衡变化

但是我们依然会在讨论添加节点的时候，使用这个概念

(2).父节点

父节点是一个相对的概念，在一个二叉树结构中，除了根节点之外，所有的节点都存在父节点

父节点一般被定义为：一个节点的上一层中，直接指向这个节点的父节点

(3).祖父节点

祖父节点同样为一个相对概念，即：父节点的父节点

我们使用祖父节点这个概念，方便描述3代以内节点之间的关系

(4).曾祖节点

曾祖节点同样为一个相对概念，即：祖父节点的父节点

我们使用曾祖节点这个概念，方便描述4代以内节点之间的关系

(5).最低不平衡节点

在向AVL树中执行添加节点操作，或者删除节点操作的时候，可能会引起AVL树整体结构的不平衡

但是在这个整体不平衡的AVL树中，又以其中层数最接近叶子节点的一个不平衡节点更为重要

我们称最接近叶子节点的不平衡节点为最低不平衡节点

值得庆幸的是：最低不平衡节点和叶子节点之间的关系，一般仅出现在4代以内

2.节点的旋转方式

前面我们已经介绍过，在AVL树中，节点通过左旋、右旋、左右旋和右左旋的方式来保持整个树结构的平衡性

那么如果我们对上面的4种旋转方式与树的不平衡情况进行对应和总结，就是：左左不平右旋、右右不平左旋、左右不平左右旋、右左不平右左旋

下面，我们从AVL树结构不平衡的状态入手，详细介绍上述4种旋转方式的实现

注意：本文中讨论的AVL树结构的不平衡情况，指的是通过插入节点和删除节点都有可能引发的不平衡情况

这些不平衡的情况正如上文所说的，最远仅可能发生在距离叶子节点4代以内的深度范围当中

2.1左左不平右旋

左左不平的情况根据叶子结点和最低不平衡节点之间的关系可以分为如下2种情况：

(1).3代之内：祖父节点的左子树深度大于右子树深度，且祖父节点左孩子的左子树深度大于等于祖父节点左孩子的右子树深度

(2).4代之内：曾祖节点的左子树深度大于右子树深度，且曾祖节点左孩子的左子树深度大于等于曾祖节点左孩子的右子树深度

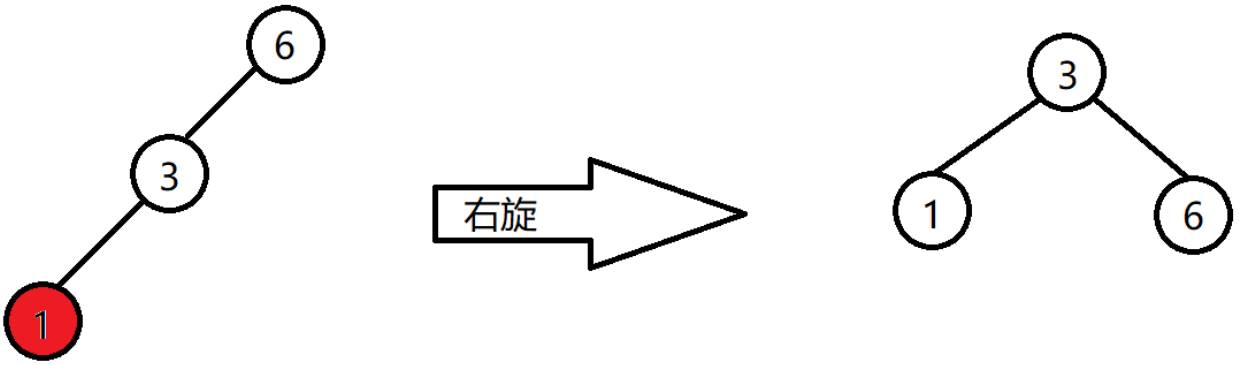
右旋的步骤：

步骤1：最低不平衡节点变成其左孩子节点的右孩子

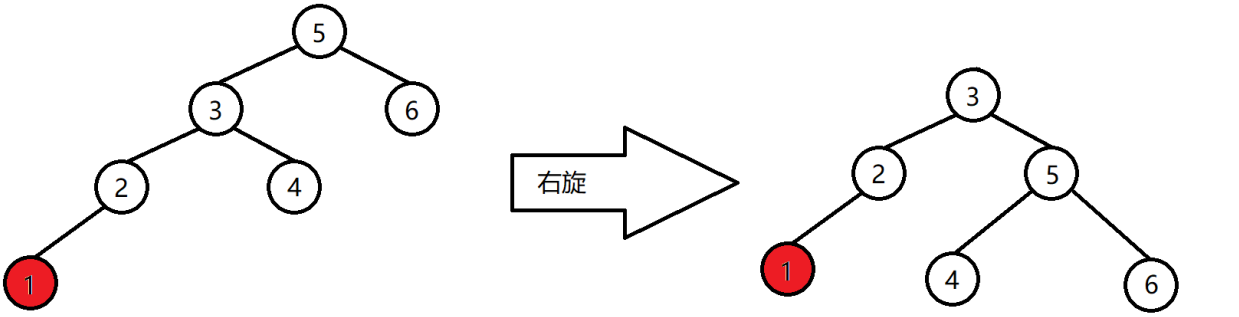
步骤2：如果最低不平衡节点的左孩子本身具有右子树，则在旋转完成后，将最低不平衡节点左孩子的右子树，变成最低不平衡节点的左子树

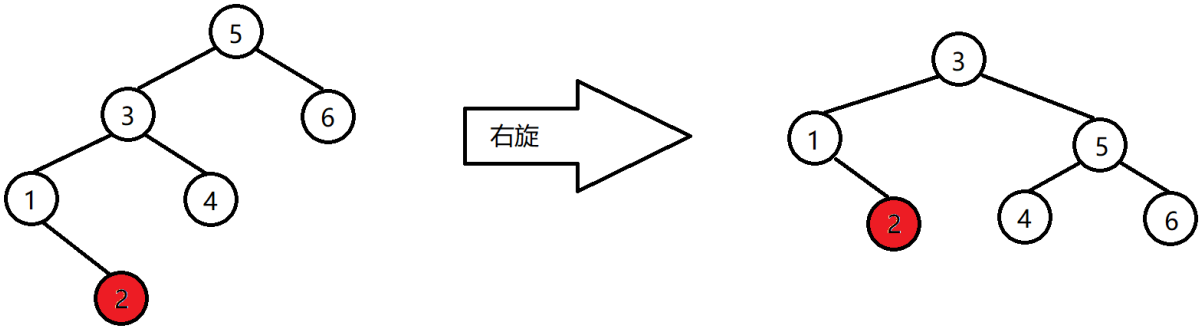
添加节点导致的右旋：

3代以内的情况：



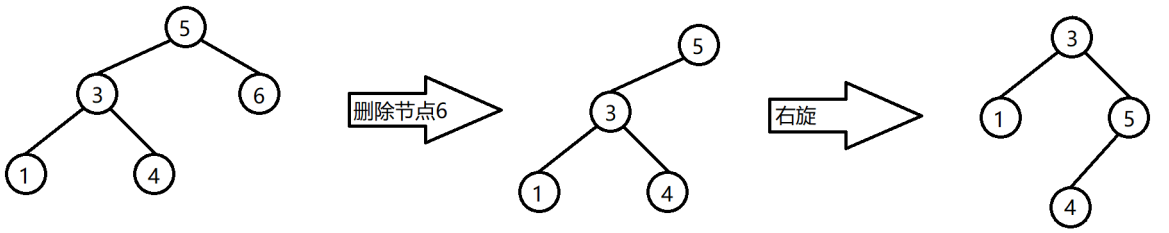
4代以内的情况：

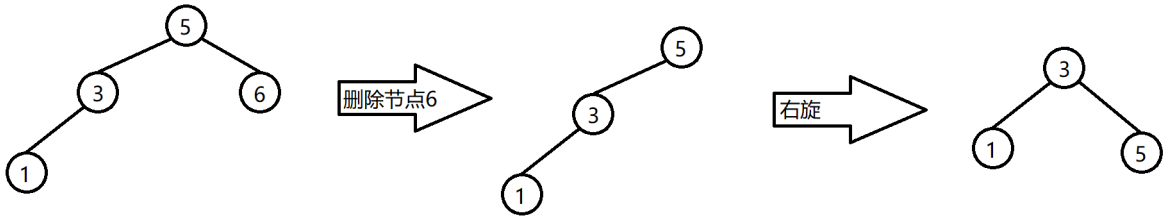




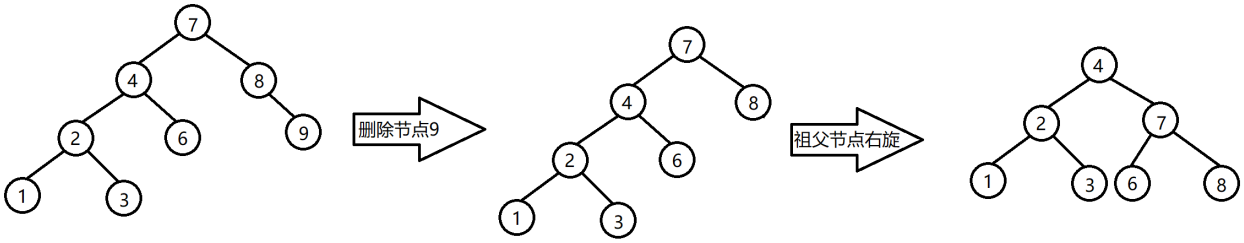
删除节点导致的右旋：

3代以内的情况：





4代以内的情况：



2.2右右不平左旋

右右不平的情况根据叶子结点和最低不平衡节点之间的关系可以分为如下2种情况：

(1).3代之内：祖父节点的右子树深度大于左子树深度，且祖父节点右孩子的右子树深度大于等于祖父节点右孩子的左子树深度

(2).4代之内：曾祖节点的右子树深度大于左子树深度，且曾祖节点右孩子的右子树深度大于等于曾祖节点右孩子的左子树深度

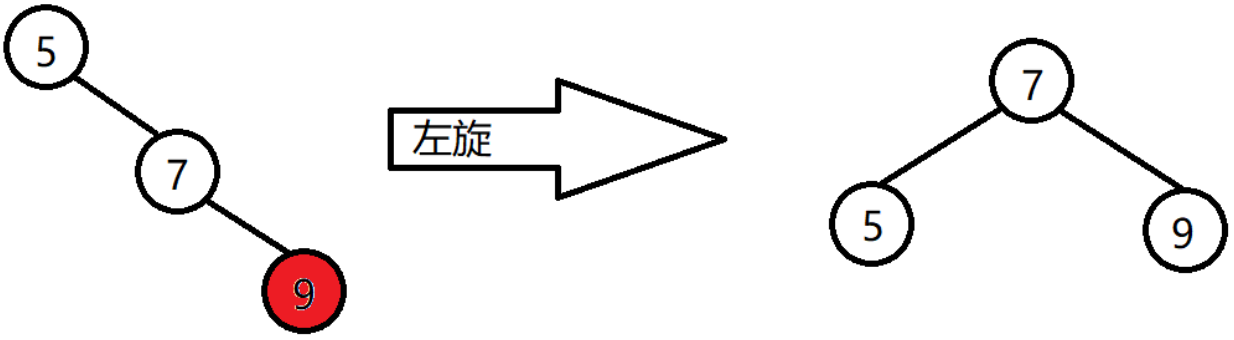
左旋的步骤：

步骤1：最低不平衡节点变成其右孩子节点的左孩子

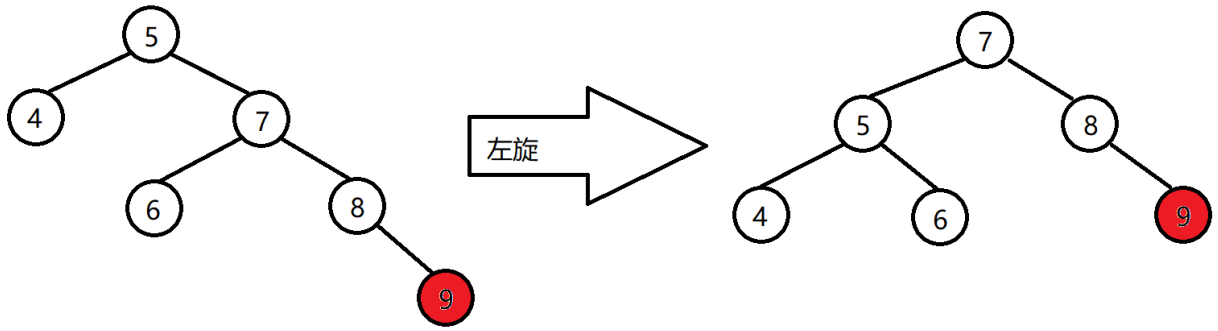
步骤2：如果最低不平衡节点的右孩子本身具有左子树，则在旋转完成后，将最低不平衡节点右孩子的左子树，变成最低不平衡节点的右子树

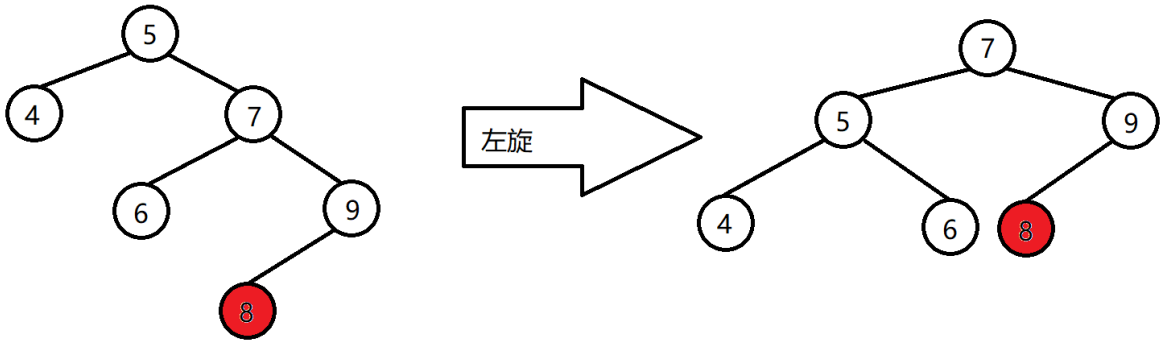
添加节点导致的左旋：

3代以内的情况：



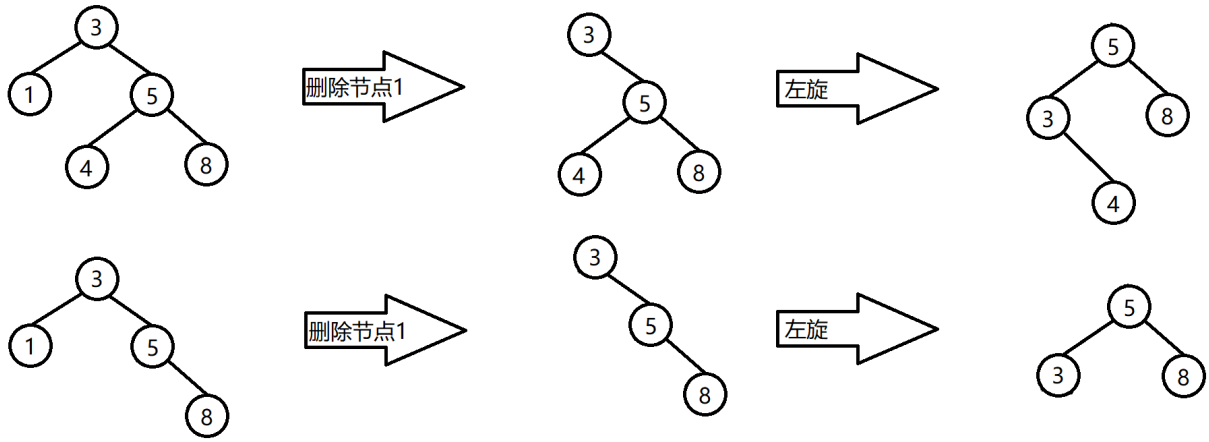
4代以内的情况：



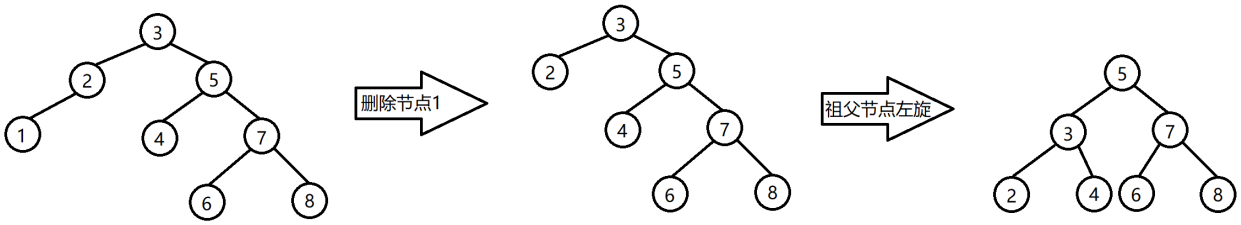


删除节点导致的左旋：

3代以内的情况：



4代以内的情况：



2.3左右不平左右旋

左右不平的情况根据叶子结点和最低不平衡节点之间的关系可以分为如下2种情况：

(1).3代之内：祖父节点的左子树深度大于右子树深度，且祖父节点左孩子的右子树深度大于祖父节点左孩子的左子树深度

(2).4代之内：曾祖节点的左子树深度大于右子树深度，且曾祖节点左孩子的右子树深度大于曾祖节点左孩子的左子树深度

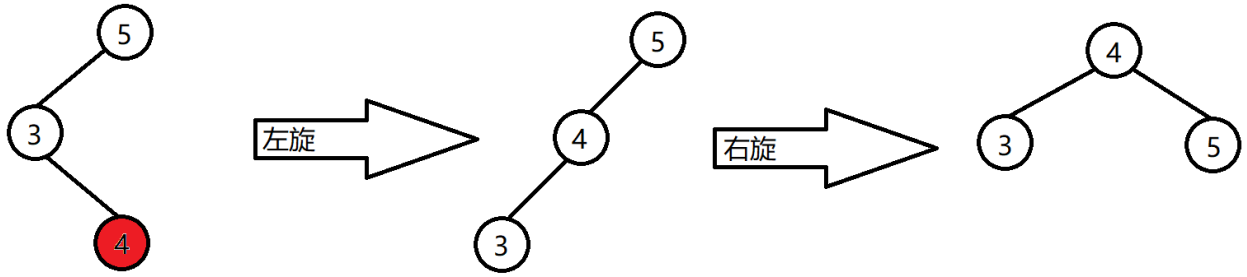
左右旋的步骤：

步骤1：最低不平衡节点的左子树左旋

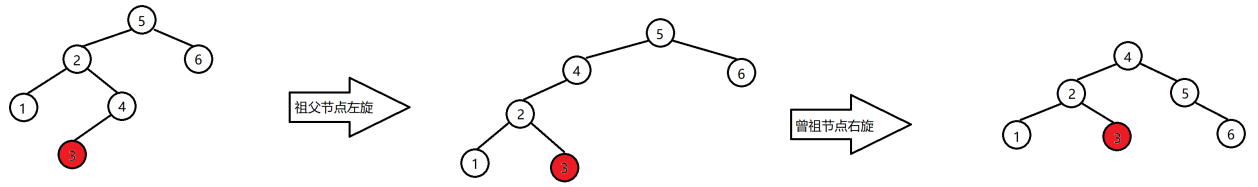
步骤2：最低不平衡节点右旋

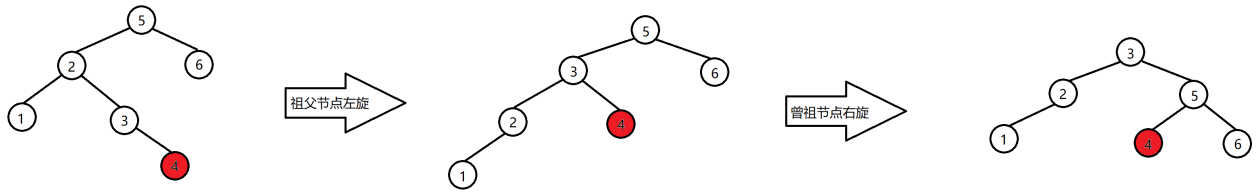
添加节点导致的左右旋：

3代以内的情况：



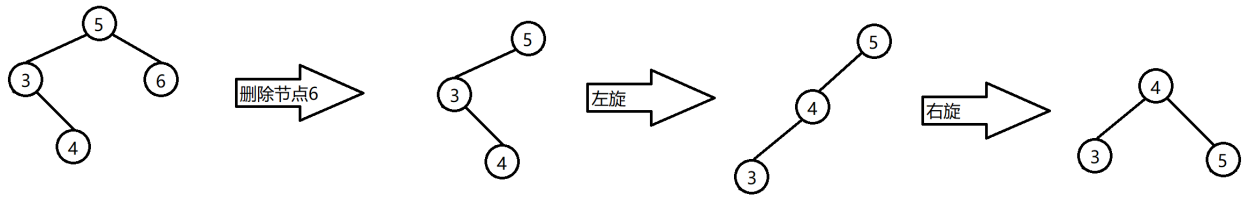
4代以内的情况：



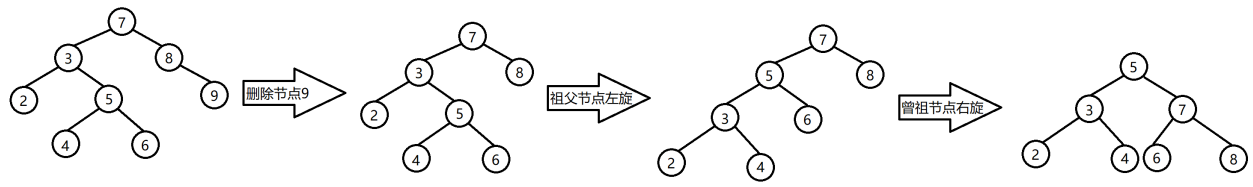


删除节点导致的左右旋：

3代以内的情况：



4代以内的情况：



2.4右左不平右左旋

右左不平的情况根据叶子结点和最低不平衡节点之间的关系可以分为如下2种情况：

(1).3代之内：祖父节点的右子树深度大于左子树深度，且祖父节点右孩子的左子树深度大于祖父节点右孩子的右子树深度

(2).4代之内：曾祖节点的右子树深度大于左子树深度，且曾祖节点右孩子的左子树深度大于曾祖节点右孩子的右子树深度

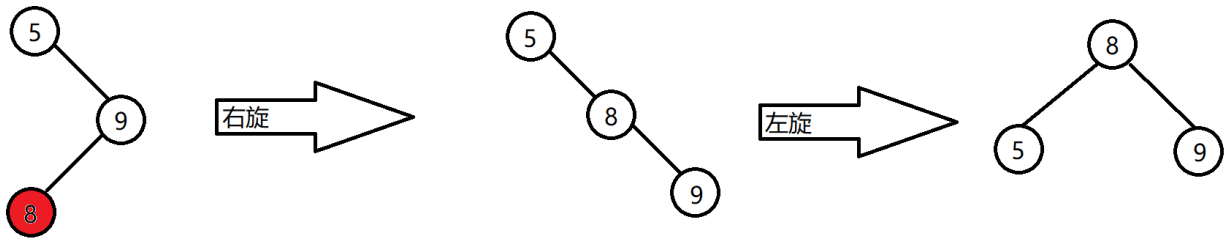
右左旋的步骤：

步骤1：最低不平衡节点的右子树右旋

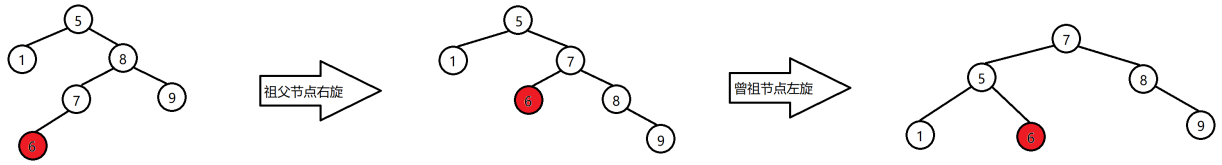
步骤2：最低不平衡节点左旋

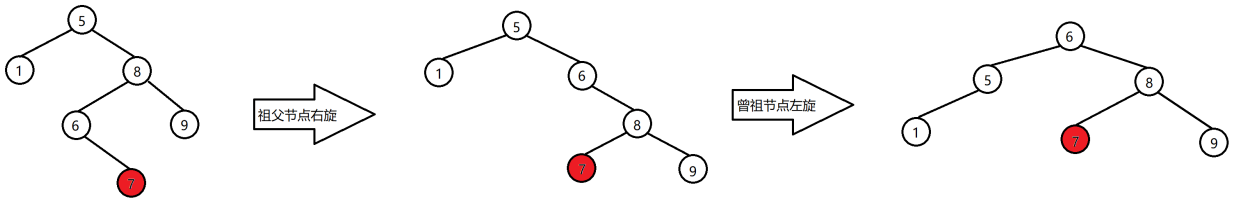
添加节点导致的右左旋：

3代以内的情况：



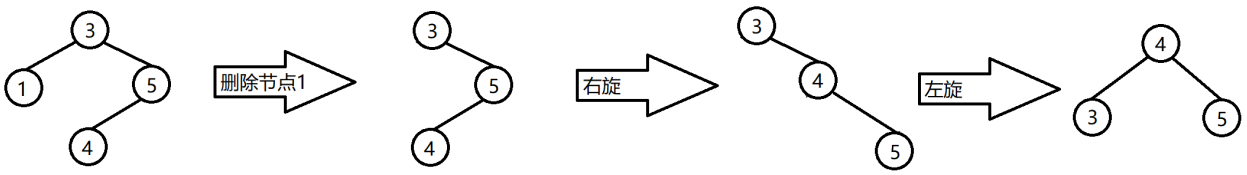
4代以内的情况：



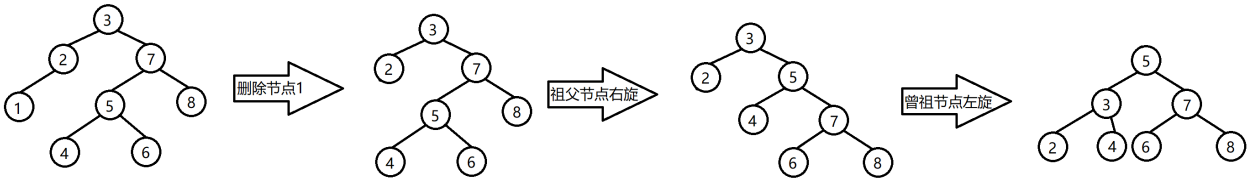


删除节点导致的右左旋：

3代以内的情况：



4代以内的情况：



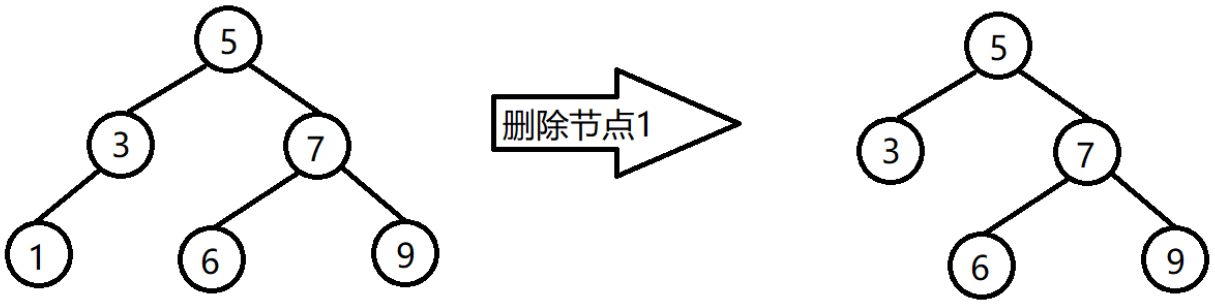
2.5删除节点但是不影响平衡性的情况

假设我们在删除AVL树的一个节点之后，整个AVL树依然是平衡的，那么我们可以根据删除节点的子节点数量，更加详细的分出如下3种情况：

(1).删除的是叶子节点

如果删除的是一个叶子节点，并且在删除之后整个AVL树依然处于平衡状态，那么此时没有必要对AVL树结构做出任何调整

直接删除这个节点即可

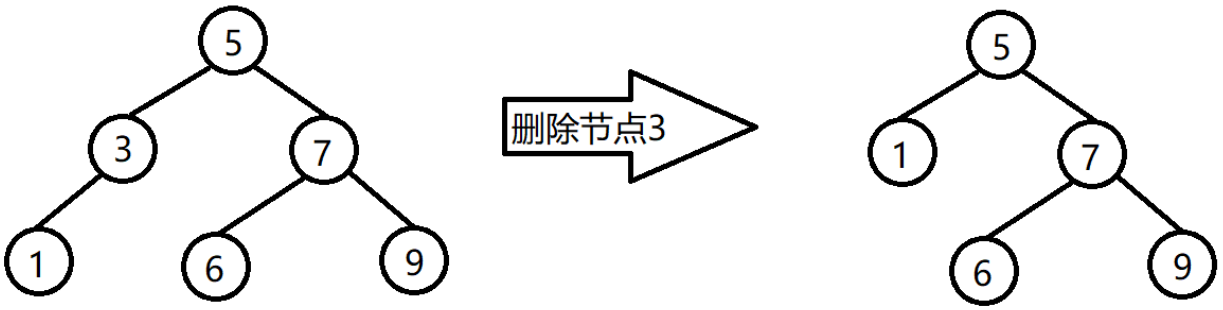


(2).删除带有一个子节点的节点

如果我们删除的节点带有一个子节点，并且在删除这个节点之后，整个AVL树结构依然保持平衡

那么不论这个节点的子节点是其左孩子还是其右孩子，只要使用这个节点的子节点替换这个节点原有的位置即可

此时依然不需要对整个AVL树结构做出任何调整

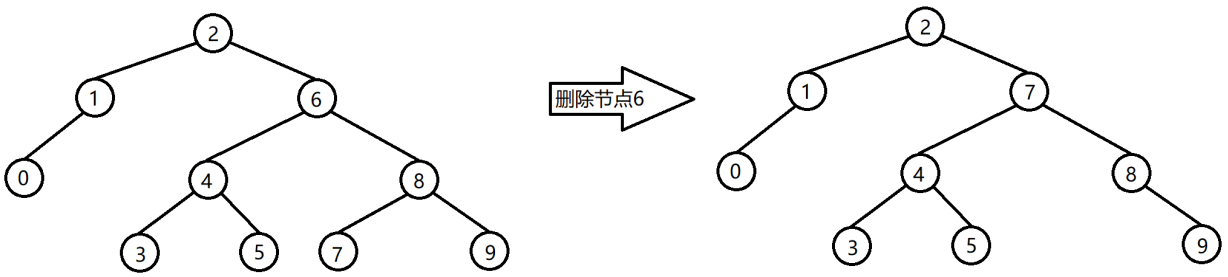


(3).删除带有两个子节点的节点

在这种情况下，我们选择使用当前AVL树中，比删除节点大的节点中，取值最小的节点，来替换删除节点，而这种节点，我们称之为后继节点

大家可能听后继节点的概念有点儿懵……简单通俗解释一下，说白了后继节点就是删除节点右子树中，最靠左下的节点

例如下图中的节点7，就是删除节点6的后继节点：



如果我们删除节点2，那么节点3就是删除节点2的后继节点，并使用节点3来替换节点2

可是如果后继节点同样具有左右孩子怎么办？那么此时我们就如同对待删除节点一样，对待这个后继节点，就好像后继节点被删除了一样