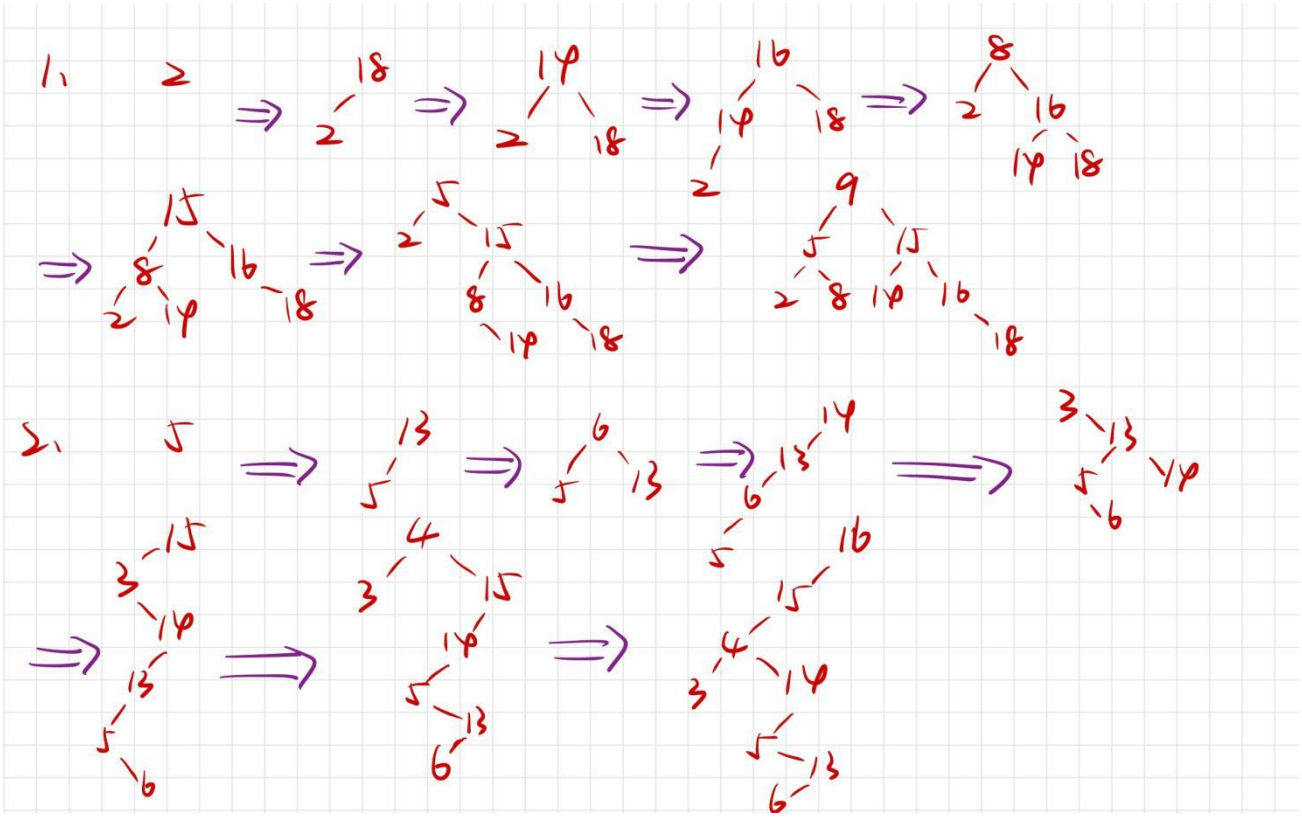
HW3 Splay Tree

一、插入 Splay Tree

如下图：



二、问答

1. splay tree 的均摊复杂度可以用**势能分析法**证明。

经查阅资料，我对于势能分析法的理解是：定义一个与数据结构的特性有所关联的势函数（通常与复杂度有关），这时每一次实际操作的代价中包含了对于势能变化量做贡献的代价，即当势能变大，意味着这一次操作有额外的代价对数据结构进行了做功；当势能变小，意味着这一次操作有之前积累的势能释放所帮助。当定义的势函数满足初始势能是最小值时，可以证明任意时刻的总均摊代价（即实际代价与势能变化量之和）是总实际代价的上界。倘若我们选择越合适的势函数，这个上界就会越精确。势能分析均摊代价的意义在于，当我们分析平均代价或者是最差代价时，往往是一个实际情况下总代价的模糊上界，比如 vector 的扩容操作带来的额外代价，往往在较少的情况下出现，而均摊代价意味着这种操作的**额外代价被其余低代价所均摊**，由此也就导出了更精确的上界。

对于伸展树，最坏情况下的代价被均摊，能够反映出伸展树自适应调整树的结构使得常访问元素移动到根节点位置从而提高了访问的效率的特性。我们可以定义势函数为所有树上节点 x 的 log|x|之和，其中|x|代表 x 节点子树的大小（节点个数）。经过数学推导，可以得出三种操作（插入，查询，删除）的摊还代价是 O(logn)的，也就是说伸展树的实际代价上界是 **O(logn)**。

2. **第一棵树较平衡，第二棵树则比较倾斜。**

我认为的可能原因如下：

两个插入序列都是“小、大、小、大.....”的规律，但是第一个序列小和大的数据范围没有明显差别，较为随机，在调整时也就自然比较平衡；而第二个序列的小数和大数明显在不同范围，**导致一字型的结构经常出现**，而伸展树对于一字型无法显著降低平衡度，当一字型一直出现时，就会产生像第二棵树那样的长链结构。