522031910747+李若彬+hw3

实践：插入操作

> 从一个空的splay树开始，依次插入给定的8个元素，要求使用自下而上的伸展方式。请画出每步插入元素后树的形态（即每个操作序列要画8个树）。
>
> 1. 插入序列为 [2, 18, 14, 16, 8, 15, 5, 9]
> 2. 插入序列为 [5, 13, 6, 14, 3, 15, 4, 16]

1. 插入序列为 [2, 18, 14, 16, 8, 15, 5, 9]

插入2：

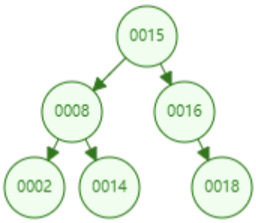
插入18：

插入14：

插入16：

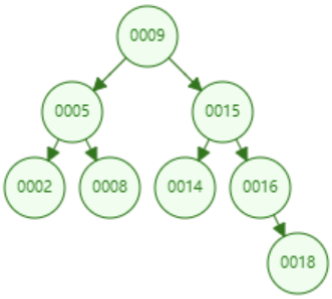
插入8：

插入15：



插入5：

插入9：



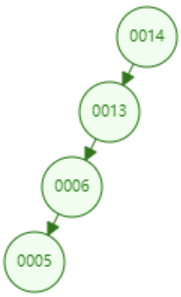
2. 插入序列为 [5, 13, 6, 14, 3, 15, 4, 16]

插入5：

插入13：

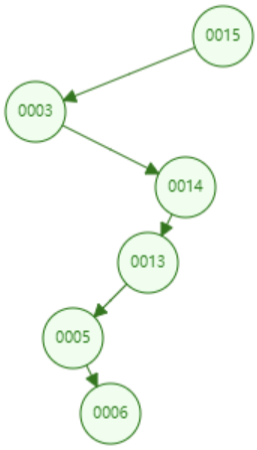
插入6：

插入14：

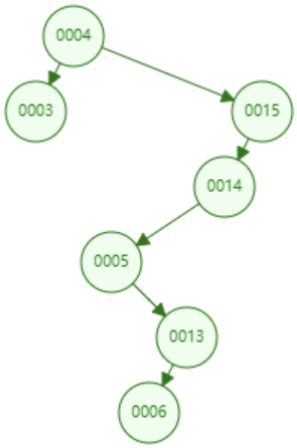


插入3：

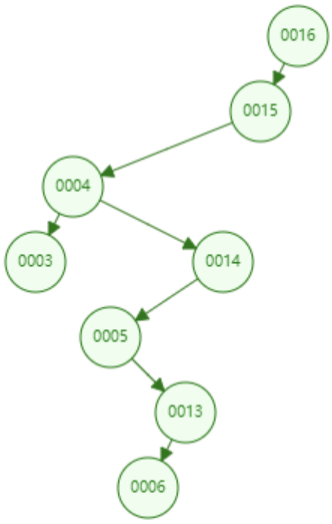
插入15：



插入4：



插入16：



小回答

> 1. 你怎么理解splay的均摊时间复杂度?
> 2. 对于实践环节的两个插入序列，它们最后产生的树的形态有什么区别？从插入序列来看，原因是什么?

1.

均摊时间复杂度即在**最坏**情况下的时间复杂度的平均，只能确保最坏情况性能的每次操作耗费的平均时间，并不能确认平均情况性能。

在`splay tree`中，操作的类型主要分为三种（考虑对称性）：`zig`，`zig-zig`，`zig-zag`。而在`splay tree`中最坏的情况即当树退化为链表且查找最小/大儿子时。现在引入**势能**概念：以节点`v`为根的子树的大小取对数，即$\phi(v)=\log(size(v))$，整棵树的势能是每个节点的势能之和$\phi=\sum_v\phi(v)$。而此时最坏情况下树的势能为：
$\sum_v\log(size(v))=\sum_{k=1}^n\log(k)=\log\prod_{k=1}^n k=\log n!=O(n\log n)$。因此此时的均摊时间复杂度为$O(n\log n)\times\frac{1}{n}=O(\log n)$。

2.

1. 第一棵树的高度更低，平衡性更好；而第二棵树的高度更高，更接近于链表的形式，平衡性较低。
2. 由于第一棵树的插入顺序中第四个元素大小在当时为第二大因此此时树的形态比较平衡，而第二棵树插入了一个最大值导致整棵树的结构变为了一个链表，而后，在第一棵树的插入元素中都不是最大值或最小值而是中间的某一个值，而第二棵树插入的元素的大小则几乎每次都为最大值或最小值，因为当`splay tree`插入一个最大值或最小值时，由于要将其转换为根节点，因此整个树的结构会趋于链表化，最终导致了两棵树截然不同的形态。