线段树的本质

本质和bvh是一样的，对原来的空间进行划分，然后用树的形式组织起来，查询的过程，就是不断地筛选出所有在查询条件中的子空间（因为树形组织，所以筛选也比较快），然后合并这些子空间的结果。而为了加速这个过程，一般在update的时候，我们会预先把一些子空间的结果合并到子空间的父空间上，那么到时候合并的话就快了。

Update的过程其实也是类似query的过程，本质也是要搜素到符合搜索区间的区间，然后再执行更新合并

思想就是这么简单，凡是能递归合并的问题，都能用线段树去解决。

线段树中的每个Node，代表一个闭合的区间，数字可以是一个这样的区间 [x, x]，val就是你预先存放的合并值，这个值在query的时候才会有作用，仅仅是针对query用到。

动态创建线段树的精髓，可以看自己写的一道题

S:\Personal\Leetcode\MinHightTree\MinHightTree\MyCalendar.cpp

重点就是 query 。

比如说你的根节点是 [0, 5]，然后咨询的是[2, 4]，你想，第一次划分[0, 5]，只能划分成[0, 2] 和 [3, 5]，这个划分显然还是和 [2, 4] 重叠的，那么我们要划分到什么时候呢？答案就是第二轮划分，[0, 1] ,[2, 2] 以及 [3, 4], [5, 5]，你可以看到经过第二轮，就只会有[2, 2],[3, 4] 都属于 [2, 4]，这个时候，我已经能得到 [2, 4] 的咨询结果了，我就不用再继续第三轮划分了，也就是 [0, 1], [5, 5]不需要再往下创建子节点了，达到了稀疏创建的效果。那么我们划分停止的关键是什么，就是 [2,4] 包含 [2, 2],[3,4]，这些节点是被咨询的范围所完全包含的，而不是互有重叠。

你再看update函数，其实也是类似的操作，不断地往下递归，直到划分到节点是被咨询范围所包含的，这里还有个叶子节点的判断，其实是因为如果已经存在了叶子节点，你的父节点要被update，那你也应该要被update，毕竟父节点的信息是来自子节点的。