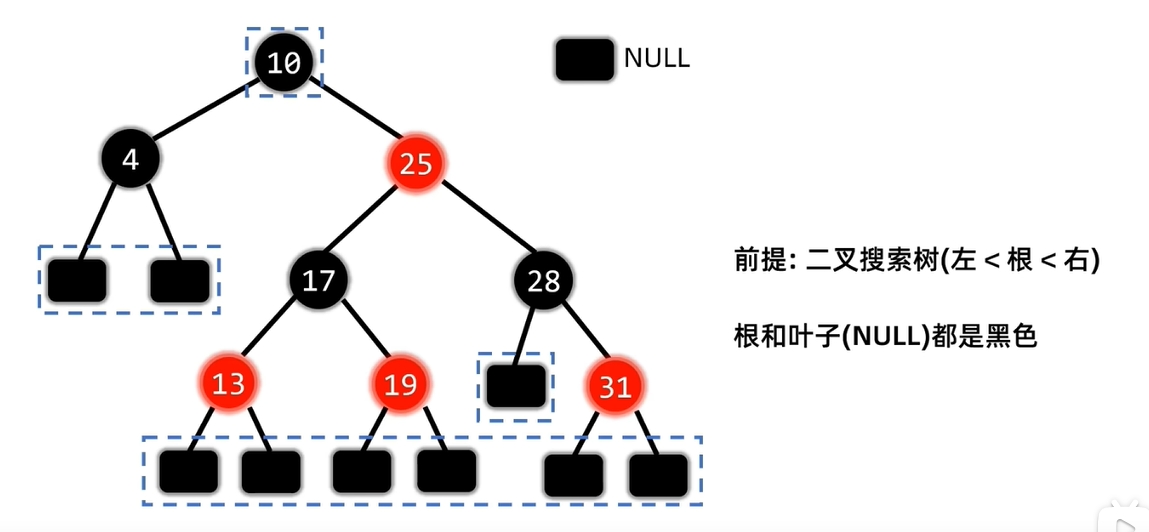
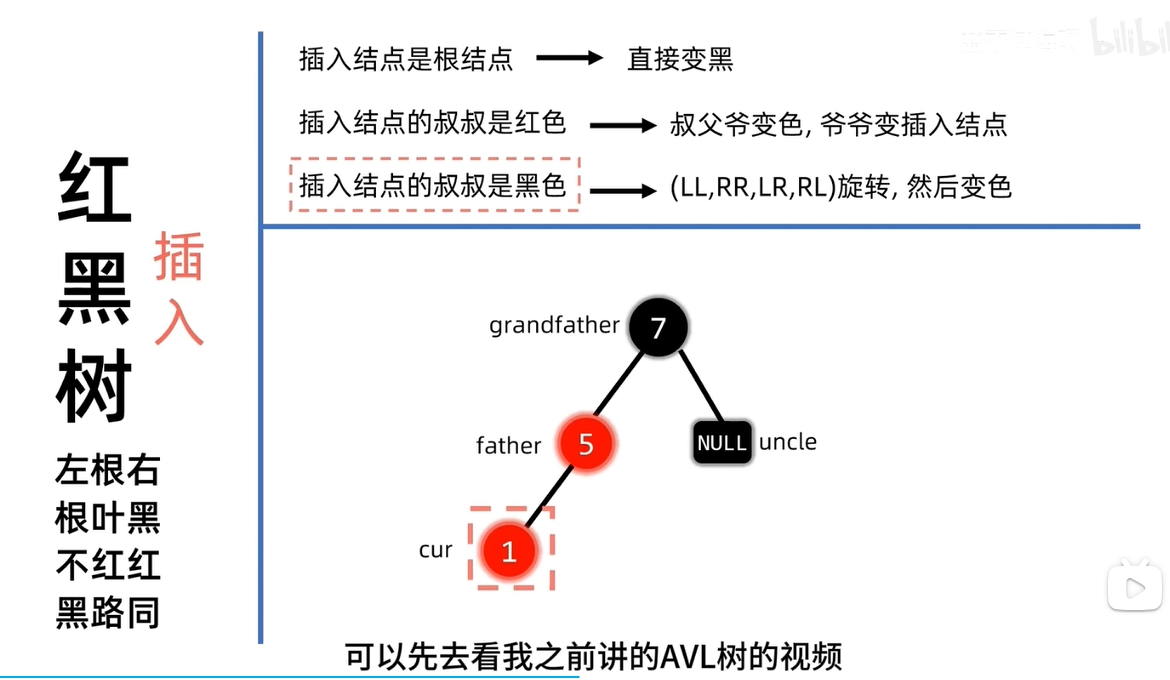
1. **红黑树**

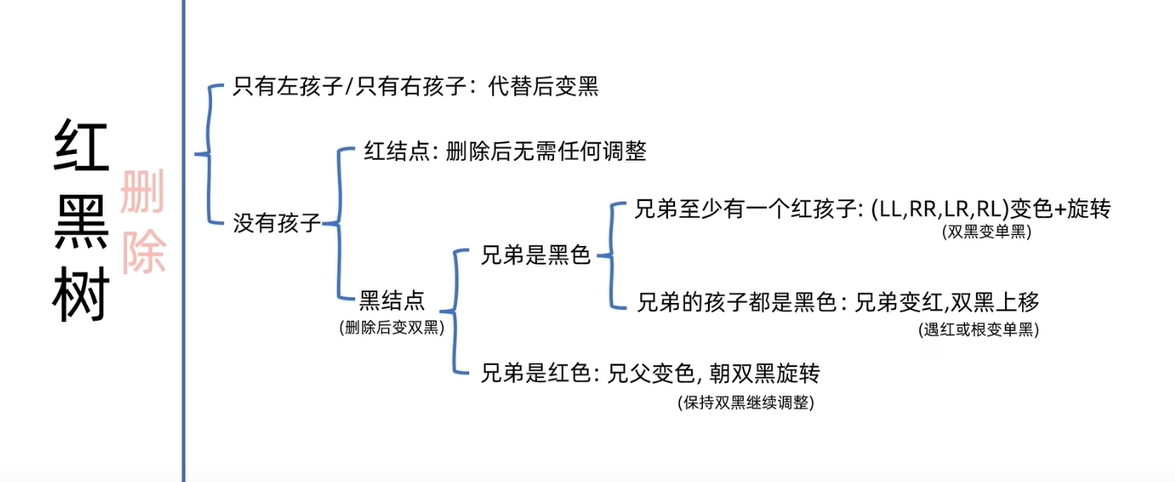


实际上是一棵二叉搜索树，红黑树的根节点和叶子节点（NULL）必须为黑色，红色节点的叶子节点必须为黑色，即不能出现两个连续的红色节点，每个节点到它的叶子的所有路径上的黑色节点数，称为黑高，必须为同一常数

**左根右，根叶黑，不红红，黑高同，**红黑树任一节点，属于它的两棵子树的高度差不会超过2倍，AVL树查询更高效，红黑树插入和删除更高效

**插入和删除：**默认先将插入节点设置为红色，有可能会破坏根叶黑和不红红，

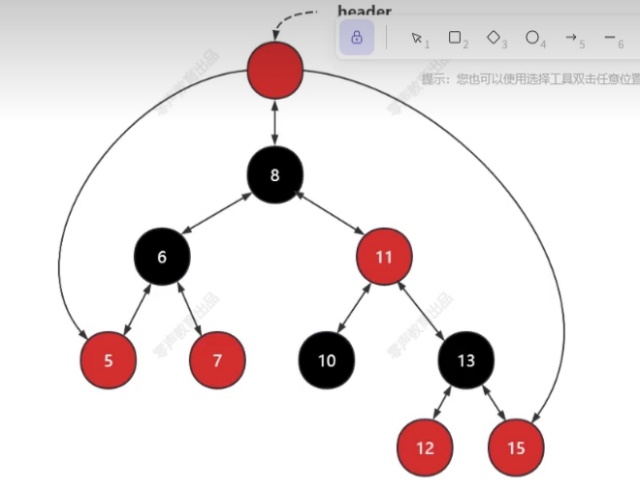
****

****

1. Epoll底层使用的是红黑树，红黑树尤其适用于节点数量级不确定的情况
2. Sk\_buff底层使用红黑树，将一个客户端视为一个节点，其数据包均从此节点进行索引
3. C++中的map，set
4. 网关流量统计
5. 内存块管理

红黑树使用的比较器默认是std::less

STL源码中

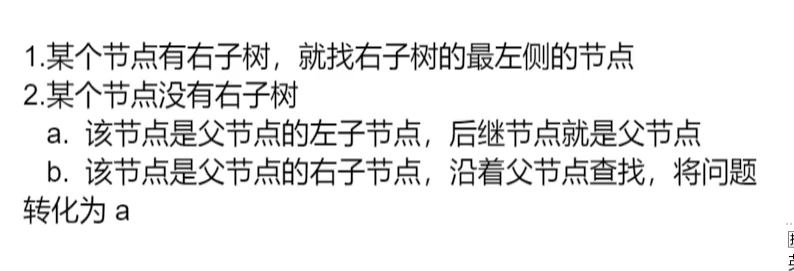


**a.** 红黑树多了一个头节点，方便快速找到最大和最小的key

**b.** key相同时，覆盖或放在相同key结点的右子树，保证先来先服务，即使key相同

**c.** 插入节点时，会优先比较是否大于最大值和小于最小值

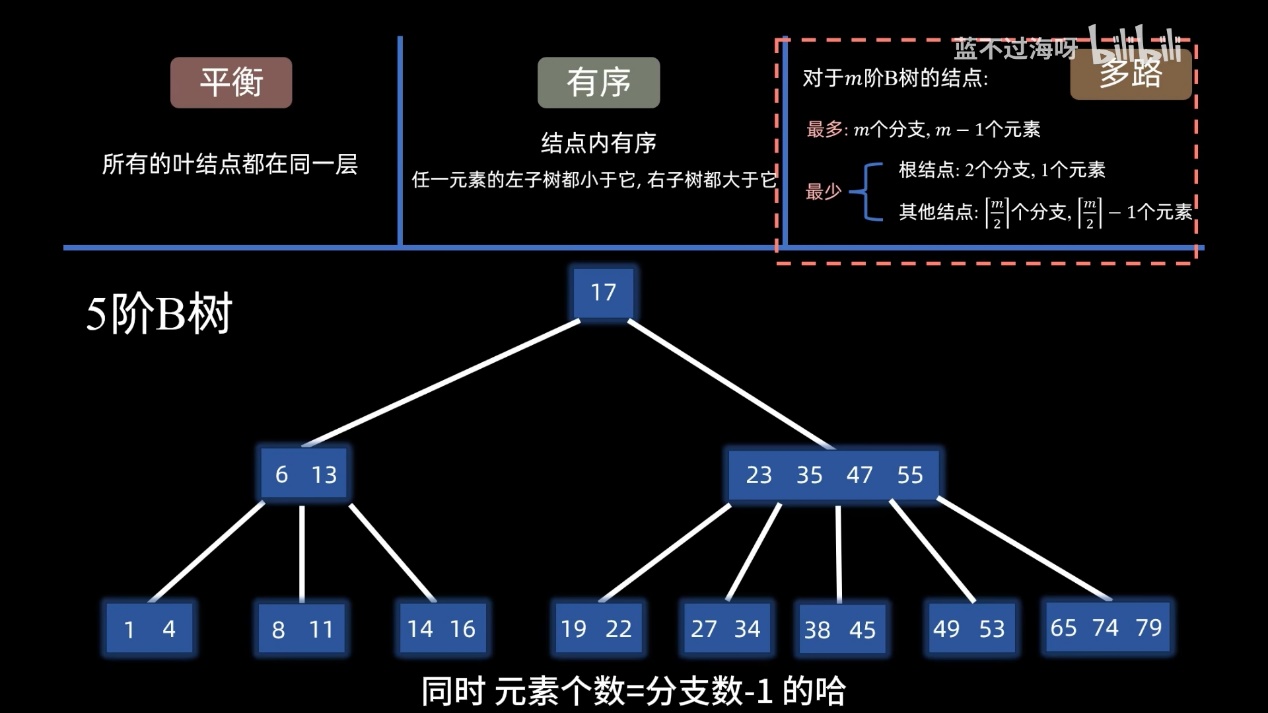
STL中红黑树找后继



**STL的组成部分：**容器、算法、迭代器、仿函数（函数对象）、适配器、空间配置器（内存分配器）

1. **B树/B+树**

所有叶子节点在同一层；每个节点至多M棵子树，根节点至少两棵子树，其他节点至少M/2子树，向上取整；所有节点都存储数据（B+树只有叶子节点存储数据，其余内节点做索引用）

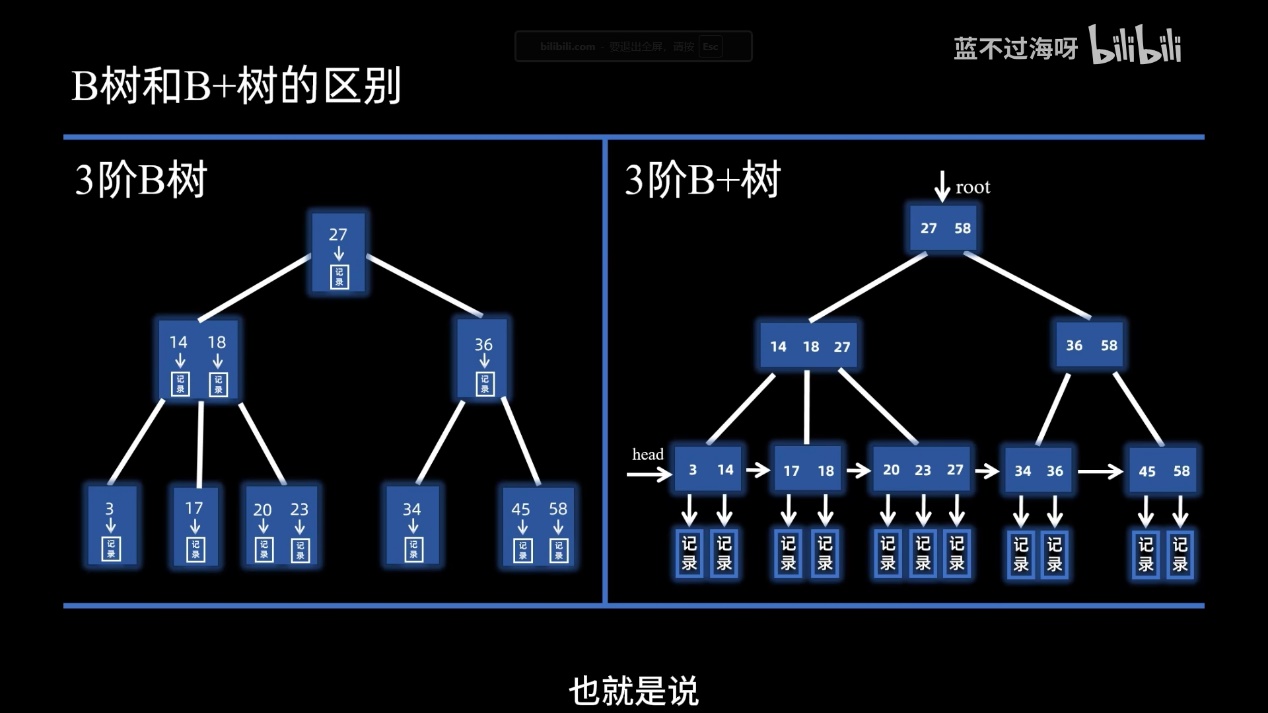


访问节点是在硬盘上进行的，而节点内的顺序查找可在内存中处理

**插入：**先找到应插入的叶子节点位置，如果没上溢，无需调整，如果上溢，以M/2下取整（索引从0开始）为中心节点，左右值分裂为两个节点，向上继续调整

**删除：**先找直接后继或前驱，保证删除叶子节点；如果下溢，兄弟够借，父下来，兄上去；不够借，父下移，和兄弟合并，继续检查父节点

B+树叶子节点层存放所有数据（索引键值key以及索引指针），常作为各种数据库的底层结构，又支持顺序查找，有支持随机查找；元素个数与分支相同；



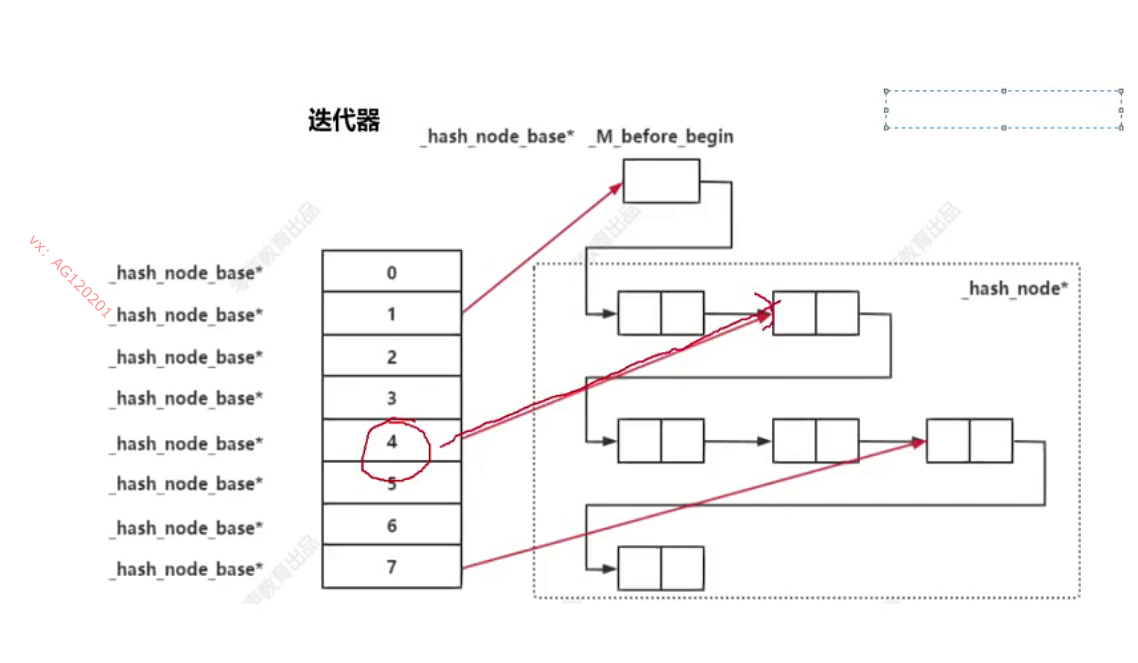
1. **Hash**

Hash函数负责将输入的key值映射到数组下标中。

选择计算速度快、强随机分布性的哈希函数，例如siphash

负载因子大小合理，使用链表法、开放寻址法、双哈希法处理冲突；过小，缩容，以节省空间，过大则扩容，然后rehash。

标准库中实现哈希表是使用单链表的，方便迭代器。



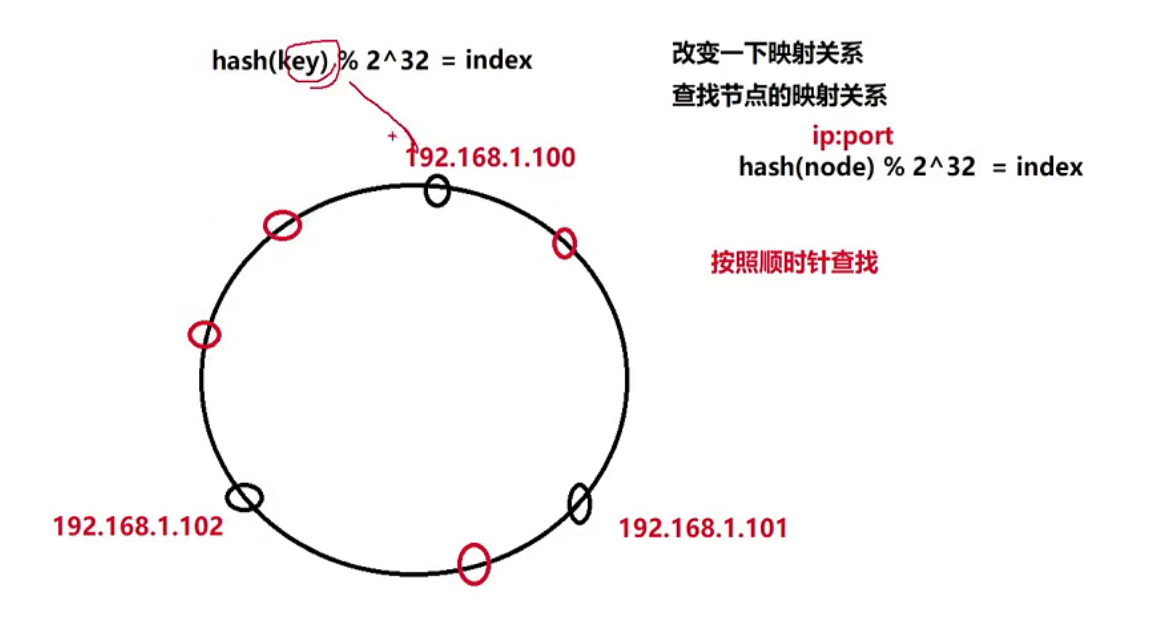
每个base指针指向上一层的最后一个节点，方便头插

**布隆过滤器：**只想知道某个key是否存在，而不关注其具体内容以及对应的val，通过一个位图和多个hash函数实现，将key对应的多个位 置为1；查询时，只要有一个位为0则不存在，否则可能存在，因为有可能被多个key重复设置；布隆过滤器不支持删除操作（hur.st/bloomfilter），hash函数个数取31时假阳率较低

uint32，最多表示到21.49亿

**面试题：**使用2GB的内存从20亿个整数中找出出现次数最多的数，分成10个文件，使用hash函数确保同样的整数映射到同一个文件中

分布式一致性hash：



仍会出现局部缓存失效，使用哈希迁移解决