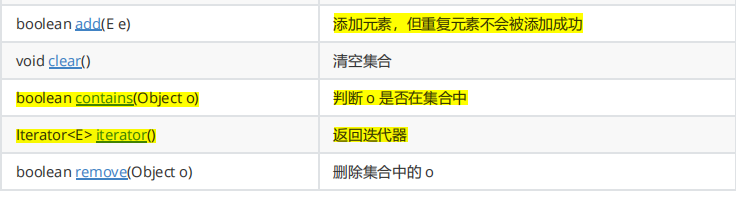
1. **纯 key 模型**，即我们 **Set** 要解决的事情，只需要判断关键字在不在集合中即可，没有关联的 value；

2. **Key-Value 模型**，即我们 **Map** 要解决的事情，需要根据指定 Key 找到关联的 Value。

**Map（插入顺序是依据哈希函数存放的,key可以为null）**



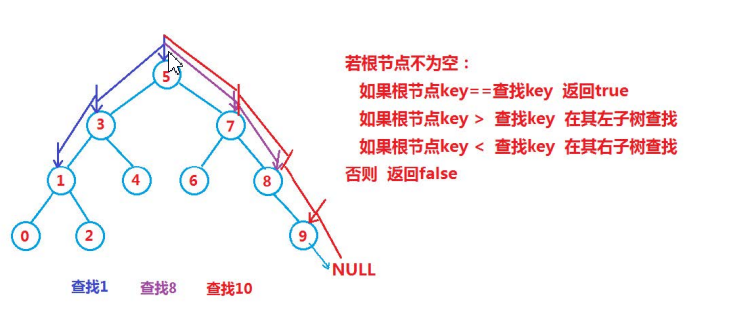
**Set（不允许重复）**



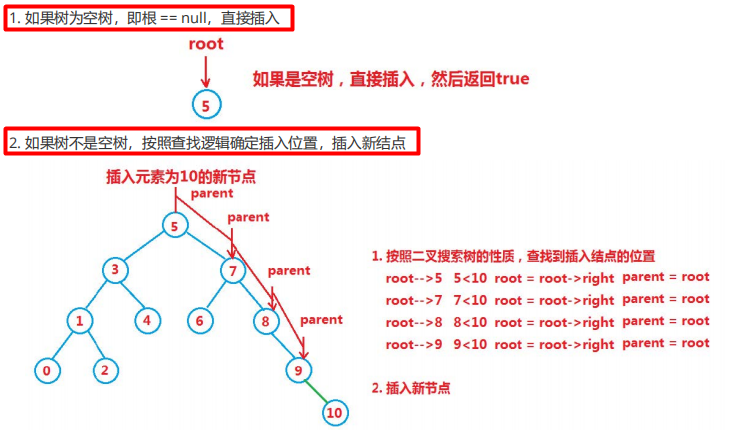
**搜索树(完全二叉树）**

**1.定义：**左子树上所有节点的值都小于根节点的值，右子树上所有节点的值都大于根节点的值

**2.查找**



**3.插入**



**4.删除**

**设待删除结点为 cur, 待删除结点的双亲结点为 parent**

1. **cur.left == null（需要删除的结点左子树为空）**

1. cur 是 root，则 root = cur.right

2. cur 不是 root，cur 是 parent.left，则 parent.left = cur.right

3. cur 不是 root，cur 是 parent.right，则 parent.right = cur.right

2. **cur.right == null（需要删除的结点右子树为空）**

1. cur 是 root，则 root = cur.left

2. cur 不是 root，cur 是 parent.left，则 parent.left = cur.left

3. cur 不是 root，cur 是 parent.right，则 parent.right = cur.left

3. **cur.left != null && cur.right != null**

需要使用**替换法**进行删除，找左子树最大的或者右子树中最小的进行替换

**哈希表（哈希表的插入/删除/查找时间复杂度是O(1) ）**

1.内部数据结构是数组

2.关键字经过变换（hash函数)得到int类型的变量

3.int类型值变为一个合法数组下标

4.把关键字放入数组的该下标

**哈希冲突：不同关键字**通过哈希函数得到**相同的哈希地址**

**冲突避免：冲突无法避免（**因为哈希表底层数组的容量往往是小于实际要存储的关键字的数量），只能尽量**降低冲突率**。、

降低冲突率方法：

哈希函数的设计使得下标均匀分布（除留余数法、平方取中法、直接定制法）

调节负载因子（放入的元素个数 / 散列表长度）间接降低冲突

扩容并进行重新哈希（遍历数组再次计算下标）

**解决冲突：开散列与闭散列**

**闭散列/开放定址法：把key存放到冲突位置中的“下一个” 空位置中去**

查找成功的平均查找长度：将每个元素需要经过查找的次数相加 / 元素个数

查找失败的平均查找长度：数组每个下标经过查找时找不到的次数相加（即从该位置开始遇到存在空白的位置需要次数) / 数组长度

**线性探测：**从发生冲突的位置开始，依次向后探测，直到寻找到下一个空位置为止。

缺陷：产生冲突的数据堆积在一块

**二次探测：**从冲突的为值开始，向后移动i^2的位置，i为第几次冲突

**开散列/哈希桶/链地址法：具有相同地址的关键码放入一个集合，每一个子集合称为一个桶，桶中的元素通过单链表链接起来，各链表的头结点存储在哈希表中。（头插）**

冲突严重时可以改进：

每个桶的背后是另一个哈希表

每个桶的背后是一棵搜索树

**4.与java类集关系**

HashMap 和 HashSet 即 java 中利用哈希表实现的 Map 和 Set

java 中使用的是哈希桶方式解决冲突的

**java 会在冲突链表长度大于一定阈值后，将链表转变为搜索树（红黑树）**

java 中**计算哈希值实际上是调用的类的 hashCode 方法**，**进行 key 的相等性比较是调用 key 的 equals 方法**。所以如果要用自定义类作为 HashMap 的 key 或者 HashSet 的值，**必须覆写 hashCode 和 equals 方法**，而且要做到 equals 相等的对象，hashCode 一定是一致的。

快捷键覆写右击选择Generate选择hashcode()&equals()

不进行覆写得到的hashcode值和equals对象是不一样的

equals相等 hashcode相等反之，不一定可能存在冲突