**链表**：链表就是一种存储地址不连续，且没有规律的数据元素组成的线性表的链式存储结构。每一个数据元素叫做一个结点，节点由两部分组成，一个是需要存储的数据，另外一个是指向下一个数据的指针。对于头节点，定义数据为空，并指向下一个节点。对于链尾结点，定义其指针为null。

正是因为每一个节点都有指向下一个节点的指针，所以链表中的数据的地址单元才可以是无序的。每一个节点只需记住其下一个元素的位置，这个链表就是完整的，当有数据插入或者删除，只需改变插入位置前后的元素的指针。这样就比较灵活，不需要初始化链表的大小。缺点是：对于单链表来说每个节点只能记录下一个节点的位置，所以如果需要访问第i个数据的时候，就必须从链首开始遍历链表，效率比较低。

**单链表节点的特性**：由于一个节点保存了下一个节点的地址，所以如果直接返回某一个节点，可以从两个角度来看待：

1. 返回的就是一个节点，可以查看节点的值
2. 返回的是从该节点为头节点一直到尾节点的链表，就像是链表被剪了一段的感觉。

这是因为该节点保存着下一个节点的地址，而下一个节点又保存这下下个节点的地址，一直可以寻找到尾节点，那么这不就是一个链表吗？

**栈**：栈是一种先进后出的数据结构

使用场景：一般用于成对出现涉及到成对抵消的题优先考虑栈来解决。以及一些升序，降序的题目，都是需要用到栈。

**二叉树**：二叉树就是子结点最多不超过两个的树。常用的是排序二叉树。排序二叉树规定为：左子结点的数值不能大于或等于父结点数值，而右子结点的数值只能大于父子结点的数值。所以左子树中结点的值一定小于右子树结点的值。二叉排序树的节点数值大小关系如下图所示：



之所以这样是因为每一次插入时，都是从根节点开始遍历的，当满足了二叉排序树的条件，方可在约束条件的位置插入。

由于有约束条件在，所以二叉排序树上删除节点的时候是比较特殊的。删除一个节点的时候一共分成三种情况来讨论：

情况1：当前节点是一个叶节点，那么直接删除就好了。

情况2：当前接待你有一个左节点或者一个右节点，那么直接删除该节点，并将其子节点接上就可以了

情况3：当前节点既有左子节点，又有右子节点。由于二叉排序树的约束条件，那么应该怎么样删除该节点，能够维持原本二叉树的结构呢？由于删除该节点之后，左右子节点都没有了父节点，破坏了结构，但是又不能舍弃其他没有被删除的节点。那么规定，需要从左子树中寻找到最大数值的一个节点，将这个节点的值赋给被删除的节点。然后再删除左子树中最大数值的节点。另外一种方法就是，在右子树中，寻找一个最小数值的节点，同样是将该数值赋给被删除节点，然后再删除右子树中最小数值的节点。

子所以要寻找左子树中最大的数值的节点，这是因为，在左子树中，任何节点的值，都是小于父节点的值，那么只有找到左子树中最大的哪一个数值，替换掉父节点的数值，再删除左子树中数值最大的节点，这样才能继续满足左子树的节点的值，一定小于这个棵树的父节点的值。寻找右子树最小的值的节点也是同样的道理。