问题：

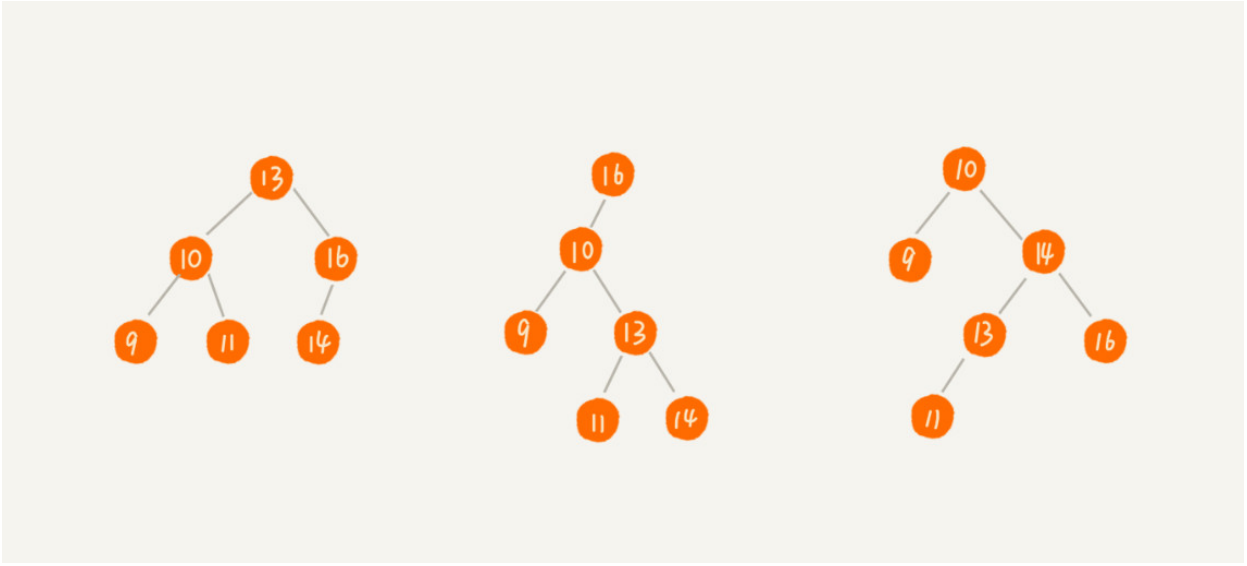
二叉查找树最大的特点就是，支持动态数据集合的快速插入、删除、查找操作。

散列表也支持这种操作，甚至还更快，那么为什么不用散列表还要有二叉树呢？

一、二叉查找树

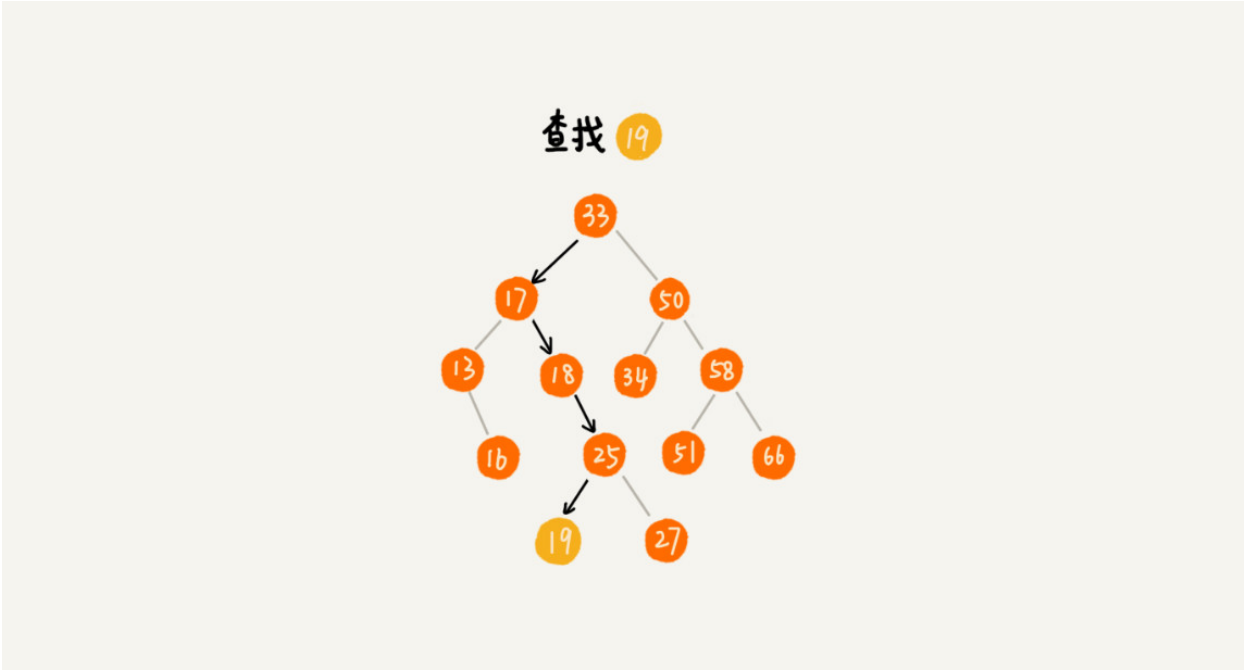
二叉查找树是二叉树的一种，也叫二叉搜索树。不仅支持快速查找数据，还支持快速插入、删除一个数据。

原理是：在树中的任意一个节点，其左子树中的每个节点的值，都要小于这个节点的值，而右子树节点的值都大于这个节点的值。



二、二叉查找树的查找操作

先取根节点，如果它等于我们要查找的数据，那就返回。如果要查找的数据比根节点的值小，那就在左子树中递归查找；如果要查找的数据比根节点的值大，那就在右子树中递归查找。



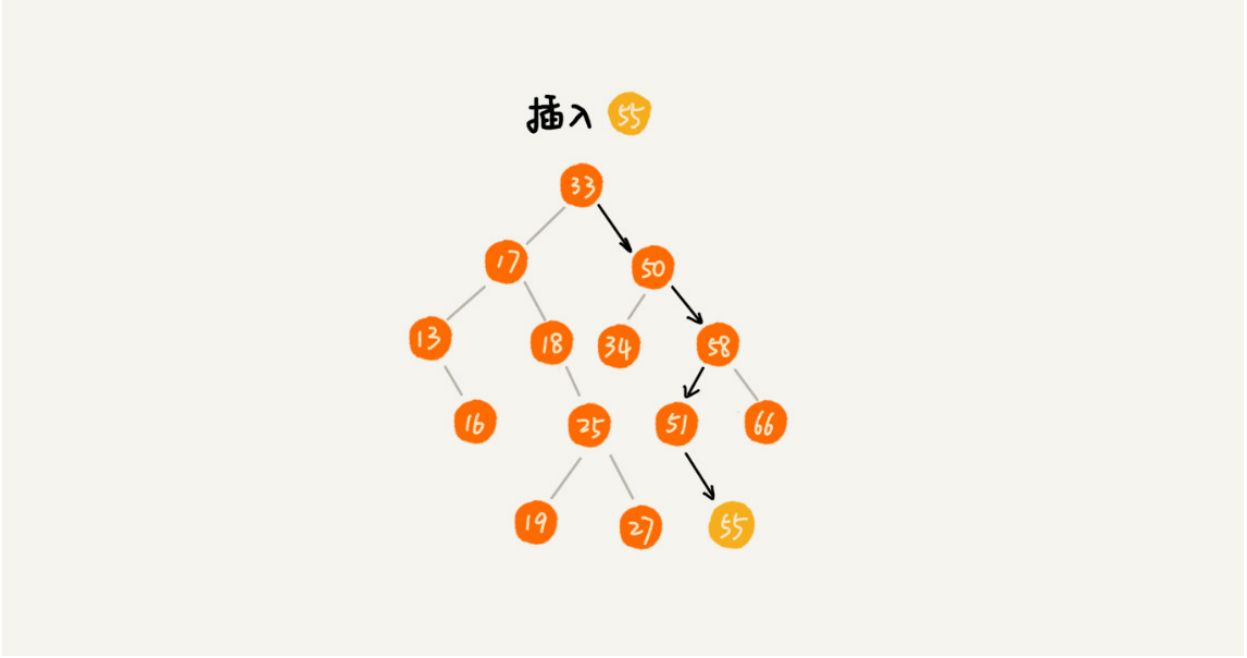
代码实现步骤：

```
public class BinarySearchTree {
  private Node tree;

  public Node find(int data) {
    Node p = tree;
    while (p != null) {
      if (data < p.data) p = p.left;
      else if (data > p.data) p = p.right;
      else return p;
    }
    return null;
  }

  public static class Node {
    private int data;
    private Node left;
    private Node right;

    public Node(int data) {
      this.data = data;
    }
  }
}
```

三、二叉查找树的插入操作

从根节点开始，依次比较要插入的数据和接地那的大小关系。

如果插入的数据比节点的数据大，并且节点的右子树为空，就将新数据直接插到右子节点的位置；如果不为空，就再递归遍历右子树，查找插入位置。同理，如果要插入的数据比节点数值小，并且节点的左子树为空，就将新数据插入到左子节点的位置；如果不为空，就再递归遍历左子树，查找插入位置。



代码实现步骤：

```
public void insert(int data) {
  if (tree == null) {
    tree = new Node(data);
    return;
  }

  Node p = tree;
  while (p != null) {
    if (data > p.data) {
      if (p.right == null) {
        p.right = new Node(data);
        return;
      }
      p = p.right;
    } else { // data < p.data
      if (p.left == null) {
        p.left = new Node(data);
        return;
      }
      p = p.left;
    }
  }
}
```

极客时间文档：https://time.geekbang.org/column/article/68334#previewimg