判断二叉搜索树是否合法

> 验证一棵树，是否为二叉搜索树。

分析：

1. 二叉排序树的左子树全部小于根节点，右子树全部大于根节点；
2. 在判断时，对左子树，左子树全部小于当前根节点，右子树全部大于当前根节点。

```
class Solution {
public:
  bool isValidBST(TreeNode *root) {
    return isValidBST(root, nullptr, nullptr);
  }

private:
  bool isValidBST(TreeNode *root, TreeNode *min, TreeNode *max) {
    if (root == nullptr) {
      return true;
    }

    if (min && root->val <= min->val) {
      return false;
    }
    if (max && root->val >= max->val) {
      return false;
    }

    return isValidBST(root->left, min, root) &&
           isValidBST(root->right, root, max);
  }
};
```

BST查找一个元素

```
class Solution {
public:
  TreeNode *searchBST(TreeNode *root, int val) {
    if (root == nullptr) {
      return nullptr;
    }

    if (root->val < val) {
      return searchBST(root->right, val);
    }

    if (root->val > val) {
```

```
      return searchBST(root->left, val);
    }

    if (root->val == val) {
      return root;
    }

    return nullptr;
  }
};
```

BST中插入元素

```
class Solution {
public:
  TreeNode *insertIntoBST(TreeNode *root, int val) {
    if (root == nullptr) {
      return new TreeNode(val);
    }

    if (root->val < val) {
      root->right = insertIntoBST(root->right, val);
    }

    if (root->val > val) {
      root->left = insertIntoBST(root->left, val);
    }

    return root;
  }
};
```

BST中删除一个元素

分析：

1. 当`root->val == key`时，主要有以下情况：
    - 只有左子树，直接返回其左子树即可；
    - 只有右子树，直接返回其右子树即可；
    - 若同时有左右子树，交换右子树的最左节点与`root`节点，递归的从`root->right`上删除指定`key`；
2. 当`root->val > key`, 递归从`root->right`删除；
3. 当`root->val < key`, 递归从`root->left`删除。

```
class Solution {
public:
  TreeNode *deleteNode(TreeNode *root, int key) {
    if (nullptr == root) {
```

```
      return root;
    }

    if (key == root->val) {
      // 只有右子树时
      if (root->left == nullptr) {
        return root->right;
      }

      // 只有左子树时
      if (root->right == nullptr) {
        return root->left;
      }

      // 含左右子树时
      TreeNode *minNode = getMin(root->right);
      root->val         = minNode->val;
      root->right       = deleteNode(root->right, minNode->val);
    } else if (root->val > key) {
      root->left = deleteNode(root->left, key);
    } else {
      root->right = deleteNode(root->right, key);
    }

    return root;
  }

private:
  TreeNode *getMin(TreeNode *root) {
    while (root->left) {
      root = root->left;
    }

    return root;
  }
};
```