```
/*
给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

    节点的左子树只包含小于当前节点的数。
    节点的右子树只包含大于当前节点的数。
    所有左子树和右子树自身必须也是二叉搜索树。

示例 1:

输入:
    2
   / \
  1   3
输出: true

示例 2:

输入:
    5
   / \
  1   4
     / \
    3   6
输出: false
解释: 输入为: [5,1,4,null,null,3,6]。
     根节点的值为 5 ，但是其右子节点值为 4 。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/validate-binary-search-tree
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 方法一(中序遍历法):根据二叉搜索树的定义可以知道,如果对二叉搜索树进行中序遍历的话,生成数据将呈升序排列,我们可以据此判断
  - 中序遍历法: 递归
  - 中序遍历法: 迭代
    - 申请辅助栈空间
    - 从根节点开始,若当前节点有左孩子,就把当前节点入栈,当前节点迁移到左孩子
    - 若当前节点无左孩子,就取栈顶元素作为当前节点,获取当前节点的值,然后把当前节点迁移到右孩子.

方法一: 1.中序遍历法:递归

```
/**
 * Definition for a binary tree node.
```

```
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution
{
    private:
        bool ret_val;

        void __midOrder(TreeNode* node,stack<int>& si)
        {
            if(  (node==NULL)
               ||(ret_val == false)
            )
            {
                return;
            }

            __midOrder(node->left,si);
            if((si.size()<1) || (si.top() < node->val))
            {
                si.push(node->val);
            }
            else
            {
                ret_val = false;
                return;
            }
            __midOrder(node->right,si);
        }

    public:
        bool isValidBST(TreeNode* root)
        {
            stack<int> sil;
            ret_val = true;
            __midOrder(root,sil);
            return ret_val;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :32 ms, 在所有 C++ 提交中击败了19.37% 的用户
内存消耗 :21.1 MB, 在所有 C++ 提交中击败了7.93%的用户
*/
```

方法一: 1.中序遍历法:迭代

```
/**
 * Definition for a binary tree node.
```

```
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution
{

    public:
        bool isValidBST(TreeNode* root)
        {

            stack<TreeNode*>    stn             ;
            TreeNode*           temp    = NULL  ;
            TreeNode*           pre     = NULL  ;

            temp = root;
            while((!stn.empty())||(temp))
            {
                if(temp)
                {
                    stn.push(temp);
                    temp = temp ->left;
                }
                else
                {
                    temp    =   stn.top();
                    stn.pop();
                    if(pre && pre->val >= temp->val)
                    {
                        return false;
                    }
                    pre     =   temp;
                    temp    =   temp -> right;
                }
            }

            return true;
        }
};

/*
执行结果：
通过
显示详情
执行用时 :16 ms, 在所有 C++ 提交中击败了89.50% 的用户
内存消耗 :20.6 MB, 在所有 C++ 提交中击败了52.90%的用户
*/
```

AlimyBreak
2019.09.13