```
/*
给定二叉搜索树（BST）的根节点和要插入树中的值，将值插入二叉搜索树。 返回插入后二叉搜索树的根节点。 保证原始二叉搜索树中不存在新值。

注意，可能存在多种有效的插入方式，只要树在插入后仍保持为二叉搜索树即可。 你可以返回任意有效的结果。

例如，

给定二叉搜索树:

        4
       / \
      2   7
     / \
    1   3

和 插入的值: 5

你可以返回这个二叉搜索树:

         4
       /   \
      2     7
     / \   /
    1   3 5

或者这个树也是有效的:

         5
       /   \
      2     7
     / \
    1   3
         \
          4

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/insert-into-a-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:

- 基本思想:根据二叉搜索树的性质,通过比较获取待插入节点值所在的范围,直到遇到空节点,再插入目标节点.
- 方法一:递归法
- 方法二:迭代法

方法一:递归法

```
/**
 * 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:
        TreeNode* __insertNode(TreeNode* node , int val)
        {
            if(node==NULL)
            {
                return new TreeNode(val);
            }
            if(node->val > val)
            {
                node->left = __insertNode(node->left,val);
            }
            else
            {
                node->right = __insertNode(node->right,val);
            }
            return node;
        }
    public:
        TreeNode* insertIntoBST(TreeNode* root, int val)
        {
            return __insertNode(root,val);
        }
};
/*
执行结果:
通过
显示详情
执行用时 :164 ms, 在所有 C++ 提交中击败了40.41% 的用户
内存消耗 :32.8 MB, 在所有 C++ 提交中击败了74.65%的用户
*/
```

方法二:迭代法

```
/**
 * 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:
```

```
        TreeNode* insertIntoBST(TreeNode* root, int val)
        {
            TreeNode*   dstNode =   new TreeNode(val)   ;
            TreeNode*   temp    =   NULL                ;
            if(root!=NULL)
            {
                temp = root;
                while(1)
                {
                    if(temp->val > val )
                    {
                        if(temp->left == NULL)
                        {
                            temp->left = dstNode;
                            break;
                        }
                        else
                        {
                            temp = temp ->left;
                        }
                    }
                    else
                    {
                        if(temp->right == NULL)
                        {
                            temp->right = dstNode;
                            break;
                        }
                        else
                        {
                            temp = temp ->right;
                        }
                    }
                }
            }
            else
            {
                root = dstNode;
            }
            return root;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :116 ms, 在所有 C++ 提交中击败了83.39% 的用户
内存消耗 :32.6 MB, 在所有 C++ 提交中击败了98.12%的用户
*/
```

AlimyBreak
2019.09.27