

```

1  /*
2  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。
3
4  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。
5
6  例如，
7
8  给定二叉搜索树：
9
10         4
11        / \
12       2   7
13      / \
14     1   3
15
16  和 插入的值：5
17
18  你可以返回这个二叉搜索树：
19
20         4
21        / \
22       2   7
23      / \  /
24     1  3 5
25
26  或者这个树也是有效的：
27
28         5
29        / \
30       2   7
31      / \
32     1   3
33         \
34         4
35
36  来源：力扣（LeetCode）
37  链接：https://leetcode-cn.com/problems/insert-into-a-binary-search-tree
38  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。
39  */

```

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

```

1  /**
2   * Definition for a binary tree node.

```

```

3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8  * };
9  */
10 class Solution
11 {
12     private:
13         TreeNode* __insertNode(TreeNode* node , int val)
14         {
15             if(node==NULL)
16             {
17                 return new TreeNode(val);
18             }
19             if(node->val > val)
20             {
21                 node->left = __insertNode(node->left, val);
22             }
23             else
24             {
25                 node->right = __insertNode(node->right, val);
26             }
27             return node;
28         }
29     public:
30         TreeNode* insertIntoBST(TreeNode* root, int val)
31         {
32             return __insertNode(root, val);
33         }
34 };
35 /*
36 执行结果:
37 通过
38 显示详情
39 执行用时 :164 ms, 在所有 C++ 提交中击败了40.41% 的用户
40 内存消耗 :32.8 MB, 在所有 C++ 提交中击败了74.65%的用户
41 */

```

方法二:迭代法

```

1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8   * };
9   */
10
11
12 class Solution
13 {
14     public:

```

```

15     TreeNode* insertIntoBST(TreeNode* root, int val)
16     {
17         TreeNode* dstNode = new TreeNode(val) ;
18         TreeNode* temp = NULL ;
19         if(root!=NULL)
20         {
21             temp = root;
22             while(1)
23             {
24                 if(temp->val > val )
25                 {
26                     if(temp->left == NULL)
27                     {
28                         temp->left = dstNode;
29                         break;
30                     }
31                     else
32                     {
33                         temp = temp ->left;
34                     }
35                 }
36                 else
37                 {
38                     if(temp->right == NULL)
39                     {
40                         temp->right = dstNode;
41                         break;
42                     }
43                     else
44                     {
45                         temp = temp ->right;
46                     }
47                 }
48             }
49         }
50         else
51         {
52             root = dstNode;
53         }
54         return root;
55     }
56 };
57
58 /*
59 执行结果:
60 通过
61 显示详情
62 执行用时 :116 ms, 在所有 C++ 提交中击败了83.39% 的用户
63 内存消耗 :32.6 MB, 在所有 C++ 提交中击败了98.12%的用户
64 */

```