```
/*
给定一个树，按中序遍历重新排列树，使树中最左边的结点现在是树的根，并且每个结点没有左子结点，只有一个右子结点。
示例 ：
输入：[5,3,6,2,4,null,8,1,null,null,null,7,9]
       5
      / \
    3    6
   / \    \
  2   4    8
 /        / \
1        7   9

输出：[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]
 1
  \
   2
    \
     3
      \
       4
        \
         5
          \
           6
            \
             7
              \
               8
                \
                 9
提示：
给定树中的结点数介于 1 和 100 之间。
每个结点都有一个从 0 到 1000 范围内的唯一整数值。
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/increasing-order-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:利用队列保存中序遍历(递归)的结果,然后根据出队顺序建立链表.
- 方法二:在中序遍历(递归)的过程中,直接根据遍历顺序建立链表.
- 方法三:在中序遍历(迭代)的过程中,直接根据遍历顺序建立链表.(todo)

方法一:C++_DFS+队列

```
/**
 * 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:
        queue<TreeNode*>  qtn;
        /*中序遍历*/
        void __midOrder(TreeNode* node)
        {
            if(node==NULL)
            {
                return;
            }
            /* 左 根 右*/
            __midOrder(node->left);
            qtn.push(node);
            __midOrder(node->right);
        }

    public:
        TreeNode* increasingBST(TreeNode* root)
        {
            TreeNode*         ret_val = NULL;
            TreeNode*         cur     = NULL;
            queue<TreeNode*>  empty;
            swap(empty,qtn);
            if(root)
            {
                /*获取中序遍历的结果*/
                __midOrder(root);
                /*依次取出队列的值组成"链表"*/
                ret_val = qtn.front();
                cur     = ret_val;
                while(1)
                {
                    qtn.pop();
                    if(qtn.empty())
                    {
                        cur->left   =   NULL;
                        cur->right  =   NULL;
                        break;
                    }
                    cur->left   = NULL;
                    cur->right  = qtn.front();
                    cur         = cur->right;
                }
            }
            return ret_val;
        }
};
/*
执行结果:
通过
显示详情
执行用时 :40 ms, 在所有 cpp 提交中击败了96.31%的用户
```

```
内存消耗 :18.7 MB, 在所有 cpp 提交中击败了91.46%的用户
*/
```

方法二:C++_DFS

```
/**
 * 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*         ret_val = NULL;
        TreeNode*         cur     = NULL;
        /*中序遍历*/
        void __midOrder(TreeNode* node)
        {
            if(node==NULL)
            {
                return;
            }
            /* 左 根 右*/
            __midOrder(node->left);
            if(ret_val)
            {
                cur->left   =   NULL;
                cur->right  =   node;
                cur         =   cur->right;
            }
            else
            {
                ret_val =   node        ;
                cur     =   ret_val     ;
            }
            __midOrder(node->right);
        }

    public:
        TreeNode* increasingBST(TreeNode* root)
        {
            ret_val =   NULL;
            cur     =   NULL;
            __midOrder(root);
            if(cur)
            {
                cur->left = NULL;
                cur->right = NULL;
            }
            return ret_val;

        }
```

```
};

/*
执行结果：
通过
显示详情
执行用时 :44 ms, 在所有 cpp 提交中击败了91.46%的用户
内存消耗 :14.7 MB, 在所有 cpp 提交中击败了100.00%的用户
*/
```

AlimyBreak
2019.10.18