```
/*
给定二叉树根结点 root ，此外树的每个结点的值要么是 0，要么是 1。

返回移除了所有不包含 1 的子树的原二叉树。

( 节点 X 的子树为 X 本身，以及所有 X 的后代。)

示例1:
输入: [1,null,0,0,1]
输出: [1,null,0,null,1]

解释:
只有红色节点满足条件“所有不包含 1 的子树”。
右图为返回的答案。


示例2:
输入: [1,0,1,0,0,0,1]
输出: [1,null,1,null,1]



示例3:
输入: [1,1,0,1,1,0,1,0]
输出: [1,1,0,1,1,null,1]



说明:

    给定的二叉树最多有 100 个节点。
    每个节点的值只会为 0 或 1 。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/binary-tree-pruning
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:
有两种需要剪枝的情况

- '0'为叶子节点
- 某子树的所有元素都为'0'

显然'0'为叶子节点的情况也是子树所有元素为'0'的特殊情况,我们可以利用递归特性,使根深层次的非叶子节点'0'变为叶子节点成为可能.
由于题设没有交代被裁剪节点是否需要释放,默认不需要释放.

方法一:C++_递归且不需要释放

```
/**
 * 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* __dfs(TreeNode* node)
        {
            if(node==NULL)
            {
                return NULL;
            }
            node->left  = __dfs(node->left);
            node->right = __dfs(node->right);
            if(   ( node->left  == NULL  )
                &&( node->right == NULL  )
                &&( node->val   == 0     )
            )
            {
                return NULL;
            }
            return node;
        }

    public:
        TreeNode* pruneTree(TreeNode* root)
        {
            root = __dfs(root);
            return root;
        }
};


/*
执行结果:
通过
显示详情
执行用时 :4 ms, 在所有 C++ 提交中击败了88.46% 的用户
内存消耗 :9.7 MB, 在所有 C++ 提交中击败了82.14%的用户
*/
```

方法二:C++_递归且需要释放

```
/**
 * 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* __dfs(TreeNode* node)
        {
            TreeNode* left_bak ;
            TreeNode* right_bak;
            if(node==NULL)
            {
                return NULL;
            }
            left_bak  = node->left;
            right_bak = node->right;
            node->left  = __dfs(node->left);
            node->right = __dfs(node->right);

            if(left_bak!=node->left)
            {
                free(left_bak);
            }
            if(right_bak!=node->right)
            {
                free(right_bak);
            }

            if(   ( node->left  == NULL  )
                &&( node->right == NULL  )
                &&( node->val   == 0     )
            )
            {
                return NULL;
            }
            return node;
        }

    public:
        TreeNode* pruneTree(TreeNode* root)
        {
            root = __dfs(root);
            return root;
        }
};


/*
运行报错,默认不需要释放.
*/
```

AlimyBreak
2019.09.25