

```

1  /*
2  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。
3
4  返回移除了所有不包含 1 的子树的原二叉树。
5
6  （节点 x 的子树为 x 本身，以及所有 x 的后代。）
7
8  示例1:
9  输入: [1,null,0,0,1]
10 输出: [1,null,0,null,1]
11
12 解释:
13 只有红色节点满足条件“所有不包含 1 的子树”。
14 右图为返回的答案。
15
16
17 示例2:
18 输入: [1,0,1,0,0,0,1]
19 输出: [1,null,1,null,1]
20
21
22
23 示例3:
24 输入: [1,1,0,1,1,0,1,0]
25 输出: [1,1,0,1,1,null,1]
26
27
28
29 说明:
30
31     给定的二叉树最多有 100 个节点。
32     每个节点的值只会为 0 或 1 。
33
34 来源：力扣（LeetCode）
35 链接：https://leetcode-cn.com/problems/binary-tree-pruning
36 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。
37 */

```

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

```

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* __dfs(TreeNode* node)
14         {
15             if(node==NULL)
16             {
17                 return NULL;
18             }
19             node->left = __dfs(node->left);
20             node->right = __dfs(node->right);
21             if( ( node->left == NULL )
22                &&( node->right == NULL )
23                &&( node->val == 0 )
24             )
25             {
26                 return NULL;
27             }
28             return node;
29         }
30
31     public:
32         TreeNode* pruneTree(TreeNode* root)
33         {
34             root = __dfs(root);
35             return root;
36         }
37 };
38
39
40 /*
41 执行结果:
42 通过
43 显示详情
44 执行用时 :4 ms, 在所有 C++ 提交中击败了88.46% 的用户
45 内存消耗 :9.7 MB, 在所有 C++ 提交中击败了82.14%的用户
46 */

```

方法二:C++_递归且需要释放

```

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* __dfs(TreeNode* node)
14          {
15              TreeNode* left_bak ;
16              TreeNode* right_bak;
17              if(node==NULL)
18              {
19                  return NULL;
20              }
21              left_bak  = node->left;
22              right_bak = node->right;
23              node->left  = __dfs(node->left);
24              node->right = __dfs(node->right);
25
26              if(left_bak!=node->left)
27              {
28                  free(left_bak);
29              }
30              if(right_bak!=node->right)
31              {
32                  free(right_bak);
33              }
34
35              if( ( node->left  == NULL )
36                  &&( node->right == NULL )
37                  &&( node->val  == 0    )
38              )
39              {
40                  return NULL;
41              }
42              return node;
43          }
44
45      public:
46          TreeNode* pruneTree(TreeNode* root)
47          {
48              root = __dfs(root);
49              return root;
50          }
51  };
52
53
54  /*
55  运行报错,默认不需要释放.
56  */

```