```
/*
给出一个完全二叉树，求出该树的节点个数。

说明：

完全二叉树的定义如下：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 1~ 2h 个节点。

示例：

输入：
    1
   / \
  2   3
 / \  /
4  5 6

输出：6

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/count-complete-tree-nodes
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:常规方法,直接递归或迭代dfs统计二叉树的节点个数即可.
- 方法二:使用迭代方法,记录下二叉树的层数和空节点数,根据二叉树层数与节点数的关系.

奇怪的是方法二居然比方法一耗时多了.

方法一:C++_dfs递归

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
/**
 * 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:
        int __nodeNum(TreeNode* node)
        {
            if(node == NULL)
            {
                return 0;
            }
            return 1 + __nodeNum(node->left) + __nodeNum(node->right);
        }

    public:
        int countNodes(TreeNode* root)
        {
            if(root!=NULL)
            {
                return 1 + __nodeNum(root->left) + __nodeNum(root->right);
            }
            return 0;
        }
};
/*
执行结果：
通过
显示详情
执行用时 :48 ms, 在所有 C++ 提交中击败了74.73% 的用户
内存消耗 :28.6 MB, 在所有 C++ 提交中击败了86.43%的用户
*/
```

方法一:C++_迭代BFS

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
/**
 * 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:
        int countNodes(TreeNode* root)
        {
            queue<TreeNode*>        qtn                 ;
            TreeNode*               temp    =   NULL    ;
```

```
            int                     ret_val =   0       ;
            int                     num_1   =   0       ;
            int                     num_2   =   0       ;

            if(root)
            {
                qtn.push(root);
                num_2 = 1;
                num_1 = 0;

                while(!qtn.empty())
                {
                    ret_val += num_2;
                    num_1 = num_2;
                    num_2 = 0;
                    for(int i = 0 ; i < num_1;i++)
                    {
                        temp = qtn.front();
                        qtn.pop();
                        if(temp->left)
                        {
                            qtn.push(temp->left);
                            num_2++;
                        }
                        if(temp->right)
                        {
                            qtn.push(temp->right);
                            num_2++;
                        }
                    }
                }
            }
            return ret_val;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :80 ms, 在所有 C++ 提交中击败了15.54% 的用户
内存消耗 :29 MB, 在所有 C++ 提交中击败了17.44%的用户
*/
```

方法二:C++_迭代BFS

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
/**
```

```
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
/**
 * 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:
        int countNodes(TreeNode* root)
        {
            queue<TreeNode*>        qtn                     ;
            TreeNode*               temp        =   NULL    ;
            int                     ret_val     =   0       ;
            int                     numNull     =   0       ;
            int                     numLevel    =   0       ;
            int                     num_1       =   0       ;
            int                     num_2       =   0       ;

            if(root)
            {
                qtn.push(root);
                num_2 = 1;
                num_1 = 0;
                while(1)
                {
                    numLevel++; /*计算层数*/
                    num_1 = num_2;
                    num_2 = 0;
                    for(int i = 0; i < num_1 ; i++)
                    {
                        temp = qtn.front();
                        qtn.pop();
                        if(temp->left!=NULL)
                        {
                            qtn.push(temp->left);
                            num_2++;
                        }
                        else
                        {
                            numNull++;
                        }
                        if(temp->right!=NULL)
                        {
                            qtn.push(temp->right);
                            num_2++;
                        }
```

```
                        else
                        {
                            numNull++;
                        }
                    }
                    /*完全二叉树非满二叉树*/
                    if(numNull!=0)
                    {
                        numLevel++;
                        break;
                    }
                    /*满二叉树*/
                    if(num_2==0)
                    {
                        numNull = 0;
                    }
                }
                return pow(2,numLevel)-1-numNull;
            }
            else
            {
                return 0;
            }
        }
};

/*
执行结果：
通过
显示详情
执行用时 :72 ms, 在所有 C++ 提交中击败了27.59% 的用户
内存消耗 :29.3 MB, 在所有 C++ 提交中击败了6.98%的用户
*/
```

AlimyBreak
2019.09.26