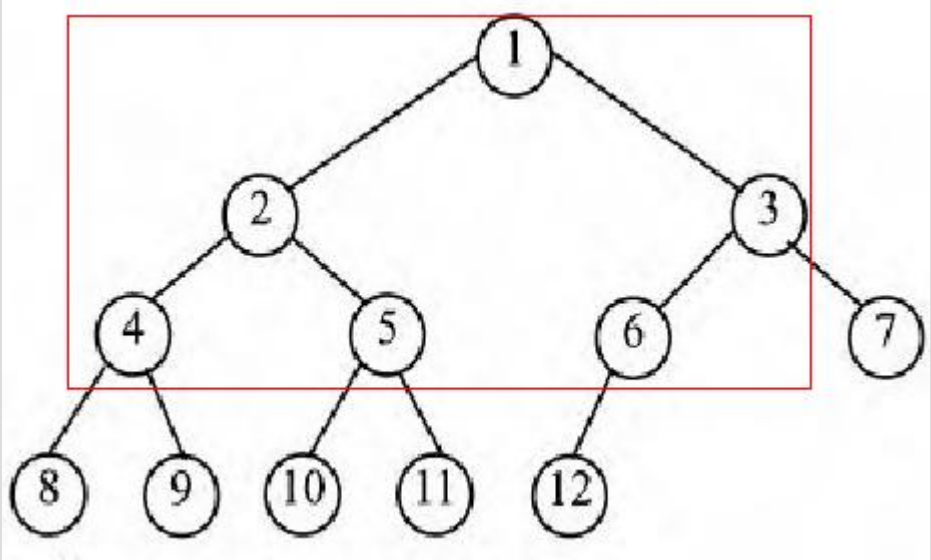
题目描述：

给定一个以顺序储存结构存储整数值的完全二叉树序列（最多 1000 个整数），请找出此完全二叉树的所有非叶子节点部分，然后采用后序遍历方式将此部分树（不包含叶子）输出。

1、只有一个节点的树，此节点认定为根节点（非叶子）。

2、此完全二叉树并非满二叉树，可能存在倒数第二层出现叶子或者无右叶子的情况



其他说明：二叉树的后序遍历是基于根来说的，遍历顺序为：左-右-根

输入描述：

一个通过空格分割的整数序列字符串

输出描述：

非叶子部分树结构的后序遍历结果

补充说明：
输出数字以空格分隔

示例 1

输入：

```
1 2 3 4 5 6 7
```
输出：
```
2 3 1
```
说明：
找到非叶子部分树结构，然后采用后续遍历输出

```
#include <iostream>
#include <sstream>
#include <vector>
#include <queue>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int num) : val(num), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    // Time Complexity: O(n), Space Complexity: O(n)
    TreeNode* buildCmpltTree(vector<int>& levels) { // 根据层次遍历序列建立完全二叉树
        queue<TreeNode* > nodes;
        int i = 0; // 遍历 levels 序列
        int len = levels.size();
        TreeNode* root = new TreeNode(levels[i]); // 树的根节点
        nodes.push(root);
        TreeNode* qfront = NULL;
        TreeNode* p = NULL;
        for (i = 1; i < len; i++) {
            qfront = nodes.front();
            p = new TreeNode(levels[i]);
            nodes.push(p);
            if (!qfront->left) { // 赋左儿子
                qfront->left = p;
            }
            else if(!qfront->right) { // 赋右儿子
                qfront->right = p;
                nodes.pop(); // 先前结点左右儿子均更新完毕，出队
            }
        }
        return root;
    }
```

```
    // Time Complexity: O(n), Space Complexity: O(n)
    void postorder_noleaf(TreeNode* root, vector<int>& post_seq) { // 后序遍历，输出非叶结点，递归
        if (root == NULL || (root->left == NULL && root->right == NULL)) {
            return;
        }
        postorder_noleaf(root->left, post_seq);
        postorder_noleaf(root->right, post_seq);
        post_seq.push_back(root->val);

        // 递归释放左右儿子
        delete root->left;
        delete root->right;
    }
};


int main() {
    vector<int> nodes; // 输入，层次遍历序列
    string input;
    getline(cin, input);
    stringstream ss(input);
    string split;
    while (getline(ss, split, ' ')) {
        nodes.push_back(atoi(split.c_str()));
    }

    if (nodes.size() == 1) { // 只含一个结点（根节点）的树的处理
        cout << nodes[0]; // 根节点看作非叶结点
        return 0;
    }

    Solution sol;
    TreeNode* root = sol.buildCmpltTree(nodes); // 建立完全二叉树
    vector<int> postorder;
    sol.postorder_noleaf(root, postorder); // 取得后根遍历结果

    // 输出后序遍历结果
    int noleaf_len = postorder.size();
    for (int i = 0; i < noleaf_len; i++) {
        cout << postorder[i] << " ";
    }

    return 0;
```

}