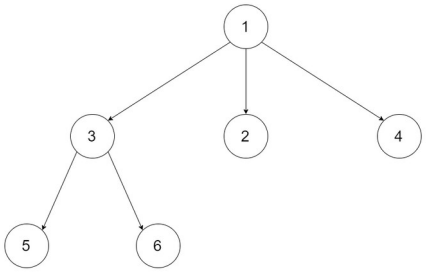
590. N-ary Tree Postorder Traversal

Given the `root` of an n-ary tree, return *the postorder traversal of its nodes' values*.

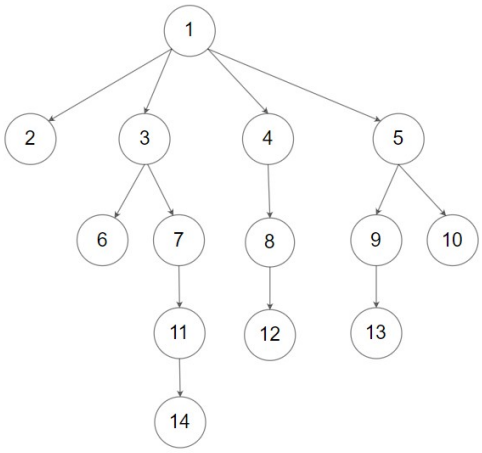
Nary-Tree input serialization is represented in their level order traversal. Each group of children is separated by the null value (See examples)

Example 1:




```
Input: root = [1,null,3,2,4,null,5,6]
Output: [5,6,3,2,4,1]
```

Example 2:




```
Input: root =
[1,null,2,3,4,5,null,null,6,7,null,8,null
,9,10,null,null,11,null,12,null,13,null,n
ull,14]
Output:
[2,6,14,11,7,3,12,8,4,13,9,10,5,1]
```

Constraints:

- The number of nodes in the tree is in the range `[0, 10^4]`.
- `0 <= Node.val <= 10^4`
- The height of the n-ary tree is less than or equal to `1000`.

Follow up: Recursive solution is trivial, could you do it iteratively?

590. N-ary Tree Postorder Traversal

Node definition

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
```

590. N-ary Tree Postorder Traversal

```
/*
    Recursive DFS
    Time complexity: O(n)
    space complexity: O(n)
    n=#nodes in the N-ary tree
*/
typedef std::vector<int> vi;
class Solution {
    public:
        vi ans;
    public:
        vi postorder(Node* root) {
            if (!root) return {};
            for(auto& child: root->children){
                postorder(child);
            }
            ans.push_back(root->val);
            return ans;
        }
};
```

590. N-ary Tree Postorder Traversal

```
/*
    Iterative DFS
    Time complexity: O(n)
    space complexity: O(n)
    n=#nodes in the N-ary tree
*/
typedef std::vector<int> vi;
class Solution {
    public:
        vi ans;
    public:
        vi postorder(Node* root) {
           std::stack<Node*> st;
           st.push(root);
           while(!st.empty()){
                Node* cur=st.top();
                st.pop();
                if(cur){
                    ans.push_back(cur->val);
                    for(auto& child: cur->children) st.push(child);
                }
           }
           std::reverse(ans.begin(),ans.end());
           return ans;
        }
};
```