JavaScript N-ary Tree Postorder Traversal

Challenge

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

N-ary tree input serialization is represented in their level order traversal. Each group of children is separated by the null value.

1st Example

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

2nd Example

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

Constraints

- `0 <= Node.val <= 10⁴`

- The number of nodes in the tree is in the range `[0, 10⁴]`.
- The height of the n-ary tree is less than or equal to `1000`.

Solution

```
const postorder = (root) => {
    if (!root) return [];

    let res   = [],
        stack = [root];

    while (stack.length > 0) {
        root = stack.pop();

        res.push(root.val);

        for (let node of root.children) {
            stack.push(node);
        }
    }

    return res.reverse();
};
```

Explanation

I've defined a function called `postorder` that takes in a parameter called `root`. The purpose of this function is to perform a postorder traversal on a tree structure represented by the `root` parameter and return an array of the values of the nodes in the tree in reverse postorder.

Inside the function, there is an initial check to see if the `root` parameter is falsy (null or undefined). If it is, the function immediately returns an empty array.

If the `root` parameter is truthy, the function initializes an empty array called `res` to store the node values in postorder and a stack array with the `root` as its only element.

The function then enters a while loop that continues as long as the stack is not empty. Inside the loop, the `root` variable is updated by popping the last element from the stack using the `pop()` method. This represents the current node being processed.

The value of the current node is then pushed into the `res` array using the `push()` method. This ensures that the node values are stored in postorder.

A for loop is used to iterate over each child node of the current node. For each child node, it is pushed into the stack using the `push()` method. This allows for the traversal of the tree structure.

Once all the child nodes have been processed, the while loop repeats until the stack is empty. This ensures that all nodes in the tree are traversed.

Finally, the function returns the `res` array in reverse order using the `reverse()` method. This provides the node values in reverse postorder as the output of the function.

In summary, this function performs a postorder traversal on a tree structure by using a stack to keep track of the nodes. It stores the node values in postorder and returns them in reverse order as the output of the function.