431. Encode N-ary Tree to Binary Tree

Leetcode Link

Problem Explanation

Here we're going to design an algorithm to encode an N-ary tree into a binary tree and then be able to decode it back to the original N-ary tree. An N-ary tree is a tree where each node has no more than N children. Similarly, a binary tree is one where each node has no more than 2 children. There is no restriction on how your encoding or decoding should work, but it must be able to accomplish accurately converting between the two tree types.

As an example: Consider an N-ary tree with N=3. It contains a root node and multiple other nodes, each node might have 0 up to 3 children. If we encode this tree as a binary tree, each node of the original N-ary tree could be represented as a binary tree node, with the node's children represented as a linked list of binary tree nodes in the left child of that node. The sibling nodes (other nodes on the same level) are then represented by the right child of each previous node.

For example, an N-ary tree node 1 with children 2, 3, 4, would be encoded into a binary tree like so: Node 1's children are represented as Node 2 (the first child) being the left child of Node 1, Node 3 being the right child of Node 2, and Node 4 being the right child of Node 3.

Solution Explanation

The provided solution encodes N-ary trees as binary trees by taking each node's children and representing them as a linked list in the binary tree. The first child of each N-ary node becomes the left child in the binary tree and other children become the right child of the previous child node.

Decoding is the reverse process. It involves converting the binary tree back to an N-ary tree. The left child of each binary tree node becomes the first child in the N-ary tree, and the right child of each binary tree node becomes the next sibling in the N-ary tree.

The solution uses a queue data structure to traverse both trees in level order and a similar approach using two while loops in both encoding and decoding methods.

Let's write this solution in different programming languages.

```
 1
 2 python
 3 from collections import deque
 4
 5 class Node:
 6     def __init__(self, val=None, children=None):
 7         self.val = val
 8         self.children = children if children else []
 9
10 class TreeNode:
11     def __init__(self, x):
12         self.val = x
13         self.left = None
14         self.right = None
15
16 class Codec:
17     def encode(self, root):
18         if not root: return None
19         newRoot = TreeNode(root.val)
20         queue = deque([(newRoot, root)])
21
22         while queue:
23             parent, curr = queue.popleft()
24             dummy = head = TreeNode(0)
25             for child in curr.children:
26                 newNode = TreeNode(child.val)
27                 dummy.right = newNode
28                 dummy = newNode
29                 queue.append((newNode, child))
30             parent.left = head.right
31         return newRoot
32
33     def decode(self, data):
34         if not data: return None
35         root = Node(data.val)
36         queue = deque([(root, data)])
37
38         while queue:
39             parent, curr = deque.popleft()
40             firstChild = curr.left
41             sibling = firstChild
42             while sibling:
43                 newNode = Node(sibling.val)
44                 parent.children.append(newNode)
45                 queue.append((newNode, sibling))
46                 sibling = sibling.right
47         return root
```

```
 1
 2 java
 3 import java.util.*;
 4
 5 class Node {
 6     public int val;
 7     public List<Node> children;
 8
 9     public Node() {}
10
11     public Node(int _val,List<Node> _children) {
12         val = _val;
13         children = _children;
14     }
15 };
16
17 class TreeNode {
18     int val;
19     TreeNode left;
20     TreeNode right;
21     TreeNode(int x) { val = x; }
22 }
23
24 class Codec {
25     public TreeNode encode(Node root) {
26         if (root == null) return null;
27         TreeNode newRoot = new TreeNode(root.val);
28         if (root.children.size() > 0) {
29             newRoot.left = encode(root.children.get(0));
30         }
31         TreeNode sibling = newRoot.left;
32         for (int i = 1; i < root.children.size(); ++i) {
33             sibling.right = encode(root.children.get(i));
34             sibling = sibling.right;
35         }
36         return newRoot;
37     }
38
39     public Node decode(TreeNode root) {
40         if (root == null) return null;
41         Node newRoot = new Node(root.val, new LinkedList<>());
42         TreeNode sibling = root.left;
43         while (sibling != null) {
44             newRoot.children.add(decode(sibling));
45             sibling = sibling.right;
46         }
47         return newRoot;
48     }
49 }
```

```
 1
 2 javascript
 3 function Node(val,children) {
 4     this.val = val;
 5     this.children = children;
 6 };
 7
 8 function TreeNode(val, left, right) {
 9     this.val = (val===undefined ? 0 : val)
10     this.left = (left===undefined ? null : left)
11     this.right = (right===undefined ? null : right)
12 };
13
14 class Codec {
15     encode(root) {
16         if (!root) return null;
17         let newRoot = new TreeNode(root.val);
18         if (root.children.length > 0) {
19             newRoot.left = this.encode(root.children[0]);
20         }
21         let sibling = newRoot.left;
22         for (let i = 1; i < root.children.length; ++i) {
23             sibling.right = this.encode(root.children[i]);
24             sibling = sibling.right;
25         }
26         return newRoot;
27     }
28
29     decode(root) {
30         if (!root) return null;
31         let newRoot = new Node(root.val, []);
32         let sibling = root.left;
33         while (sibling !== null) {
34             newRoot.children.push(this.decode(sibling));
35             sibling = sibling.right;
36         }
37         return newRoot;
38     }
39 }
```

```
 1
 2 c++
 3 #include <vector>
 4 #include <queue>
 5
 6 using namespace std;
 7
 8 class Node {
 9 public:
10     int val;
11     vector<Node*> children;
12
13     Node() {}
14
15     Node(int _val) {
16         val = _val;
17     }
18
19     Node(int _val, vector<Node*> _children) {
20         val = _val;
21         children = _children;
22     }
23 };
24
25 class TreeNode {
26 public:
27     int val;
28     TreeNode *left;
29     TreeNode *right;
30     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
31 };
32
33 class Codec {
34 public:
35     TreeNode* encode(Node* root) {
36         if (!root) return NULL;
37         TreeNode* newRoot = new TreeNode(root -> val);
38         if (!root -> children.empty()) newRoot -> left = encode(root -> children[0]);
39         TreeNode* sibling = newRoot -> left;
40         for (int i=1; i<root->children.size(); i++) {
41             sibling -> right = encode(root -> children[i]);
42             sibling = sibling -> right;
43         }
44         return newRoot;
45     }
46
47     Node* decode(TreeNode* root) {
48         if (!root) return NULL;
49         Node* newRoot = new Node(root -> val, {});
50         TreeNode* sibling = root -> left;
51         while (sibling != NULL) {
52             newRoot -> children.push_back(decode(sibling));
53             sibling = sibling -> right;
54         }
55         return newRoot;
56     }
57 };
```

The above solutions in Python, Java, and JavaScript, and C++ effectively demonstrate how to solve the problem of encoding and decoding an N-ary tree into and from a binary tree. While each language has its unique syntax and some methods might not be directly transferrable from one language to another, all these solutions hold the same fundamental logic and structure that makes them valid solutions.

In Python, Java, JavaScript, and C++, classes are defined to represent the nodes in the N-ary and binary trees. They also include a Codec class that has encode and decode methods that perform the conversion of trees, and utilize a queue data structure to ease the traversal of nodes.

The encode method starts by checking if the root node exists. If it does not exist, the method returns null, otherwise it creates a new node in the binary tree. It then goes on to convert the children of the current node, if any, into a linked list of binary tree nodes. The first child becomes the left child of the current binary tree node, while the remaining children become the right child of the preceding binary node.

The decode method reverts the encoding process by transforming the linked list of binary tree nodes back into the original N-ary tree. The left child of each binary tree node becomes the first child in the N-ary tree, and all the right children become the next sibling in the N-ary tree.

Despite the differences in syntax and built-in methods among Python, JavaScript, Java, and C++, they all employ the same logic and approach to solving the problem, thus linking these solutions together in their common goal of accurately converting between N-ary and binary trees.