Problem 428: Serialize and Deserialize N-ary Tree

Problem Information

Difficulty: Hard
Acceptance Rate: 68.45%
Paid Only: Yes
Tags: String, Tree, Depth-First Search, Breadth-First Search

Problem Description

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize an N-ary tree. An N-ary tree is a rooted tree in which each node has no more than N children. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that an N-ary tree can be serialized to a string and this string can be deserialized to the original tree structure.

For example, you may serialize the following `3-ary` tree

as `[1 [3[5 6] 2 4]]`. Note that this is just an example, you do not necessarily need to follow this format.

Or you can follow LeetCode's level order traversal serialization format, where each group of children is separated by the null value.

For example, the above tree may be serialized as `[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]`.

You do not necessarily need to follow the above-suggested formats, there are many more different formats that work so please be creative and come up with different approaches yourself.

Example 1:

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:** [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]

Example 2:

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

Example 3:

Input: root = [] **Output:** []

Constraints:

* The number of nodes in the tree is in the range `[0, 104]`. * `0 <= Node.val <= 104` * The height of the n-ary tree is less than or equal to `1000` * Do not use class member/global/static variables to store states. Your encode and decode algorithms should be stateless.

Code Snippets

C++:

```
/*
// 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;
}
};
*/

class Codec {
public:
// Encodes a tree to a single string.
string serialize(Node* root) {

}

// Decodes your encoded data to tree.
Node* deserialize(string data) {

}
};

// Your Codec object will be instantiated and called as such:
// Codec codec;
// codec.deserialize(codec.serialize(root));
```

Java:

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

public Node() {}

public Node(int _val) {
val = _val;
}

public Node(int _val, List<Node> _children) {
val = _val;
children = _children;
```

```
}
};
*/

class Codec {
// Encodes a tree to a single string.
public String serialize(Node root) {

}

// Decodes your encoded data to tree.
public Node deserialize(String data) {

}
}

// Your Codec object will be instantiated and called as such:
// Codec codec = new Codec();
// codec.deserialize(codec.serialize(root));
```

Python3:

```
"""
# Definition for a Node.
class Node(object):
def __init__(self, val: Optional[int] = None, children:
Optional[List['Node']] = None):
if children is None:
children = []
self.val = val
self.children = children
"""

class Codec:
def serialize(self, root: 'Node') -> str:
"""Encodes a tree to a single string.

:type root: Node
:rtype: str
"""
```

```
def deserialize(self, data: str) -> 'Node':
"""Decodes your encoded data to tree.

:type data: str
:rtype: Node
"""


# Your Codec object will be instantiated and called as such:
# codec = Codec()
# codec.deserialize(codec.serialize(root))
```