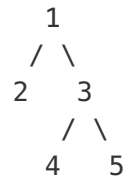
Serialize and Deserialize Binary Tree

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 a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary 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 tree



as `"[1,2,3,null,null,4,5]"`, just the same as how LeetCode OJ serializes a binary tree. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.
Note: Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

Credits:
Special thanks to @Louis1992 for adding this problem and creating all test cases.

Solution 1

The idea is simple: print the tree in pre-order traversal and use "X" to denote null node and split node with ",". We can use a StringBuilder for building the string on the fly. For deserializing, we use a Queue to store the pre-order traversal and since we have "X" as null node, we know exactly how to where to end building subtress.

```
public class Codec {
    private static final String spliter = ",";
    private static final String NN = "X";

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuilder sb = new StringBuilder();
        buildString(root, sb);
        return sb.toString();
    }

    private void buildString(TreeNode node, StringBuilder sb) {
        if (node == null) {
            sb.append(NN).append(spliter);
        } else {
            sb.append(node.val).append(spliter);
            buildString(node.left, sb);
            buildString(node.right,sb);
        }
    }
    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        Deque<String> nodes = new LinkedList<>();
        nodes.addAll(Arrays.asList(data.split(spliter)));
        return buildTree(nodes);
    }

    private TreeNode buildTree(Deque<String> nodes) {
        String val = nodes.remove();
        if (val.equals(NN)) return null;
        else {
            TreeNode node = new TreeNode(Integer.valueOf(val));
            node.left = buildTree(nodes);
            node.right = buildTree(nodes);
            return node;
        }
    }
}
```

written by gavinlinasd original link here

Solution 2

Python

```python
class Codec:

    def serialize(self, root):
        def doit(node):
            if node:
                vals.append(str(node.val))
                doit(node.left)
                doit(node.right)
            else:
                vals.append('#')
        vals = []
        doit(root)
        return ' '.join(vals)

    def deserialize(self, data):
        def doit():
            val = next(vals)
            if val == '#':
                return None
            node = TreeNode(int(val))
            node.left = doit()
            node.right = doit()
            return node
        vals = iter(data.split())
        return doit()
```

C++

```
class Codec {
public:

    string serialize(TreeNode* root) {
        ostringstream out;
        serialize(root, out);
        return out.str();
    }

    TreeNode* deserialize(string data) {
        istringstream in(data);
        return deserialize(in);
    }

private:

    void serialize(TreeNode* root, ostringstream& out) {
        if (root) {
            out << root->val << ' ';
            serialize(root->left, out);
            serialize(root->right, out);
        } else {
            out << "# ";
        }
    }

    TreeNode* deserialize(istringstream& in) {
        string val;
        in >> val;
        if (val == "#")
            return nullptr;
        TreeNode* root = new TreeNode(stoi(val));
        root->left = deserialize(in);
        root->right = deserialize(in);
        return root;
    }
};
```

written by StefanPochmann original link here

Solution 3

```
 public String serialize(TreeNode root) {
    StringBuilder sb = new StringBuilder();
    helperS(root, sb);
    return sb.toString();
}

private void helperS(TreeNode node, StringBuilder sb){
    if(node == null){
        sb.append("null").append(",");
        return;
    }

    sb.append(node.val).append(",");

    helperS(node.left, sb);
    helperS(node.right, sb);
}

// Decodes your encoded data to tree.
public TreeNode deserialize(String data) {
    String[] vals  = data.split("[,]");
    int[]    index = new int[]{0};
    return helperD(vals, index);
}

private TreeNode helperD(String[] vals, int[] index){
    if(index[0] == vals.length){
        return null;
    }

    String visiting = vals[index[0]++];
    if(visiting.equals("null")){
        return null;
    }

    TreeNode node = new TreeNode(Integer.valueOf(visiting));
    node.left     = helperD(vals, index);
    node.right    = helperD(vals, index);

    return node;
}
```

written by larrywang2014 original link here