**Problem: Serialize and Deserialize Binary Tree**

**Introduction**

The "Serialize and Deserialize Binary Tree" challenge on LeetCode is a hard-level problem that requires transforming a binary tree into a string representation and vice versa. The goal is to create an algorithm that can serialize a binary tree into a string and then deserialize the string back into the binary tree.

**Important Insights needed to solve this problem**

To solve this problem, we need to explore how to efficiently encode a binary tree as a string and rebuild the tree from the string. One critical insight required to tackle this challenge is the selection of an appropriate format for serializing the binary tree. The preorder traversal of the tree is a frequent format. The preorder traversal visits the binary tree nodes in the following order: root, left subtree, right subtree. We can simply recreate the tree by following the same order after serializing it using this traversal.

**Well-known algorithm**

Depth-first search known as DFS is a well-known method that may be used to solve this problem. We can traverse the binary tree depth-first while serializing and deserializing it. We can recursively visit each node during serialization, attaching its value to the serialized string. We may iteratively recreate the tree during deserialization by extracting values from the serialized string and constructing nodes accordingly.

**Choice of different data-structures for the problem**

The data structures used to solve this challenge are determined by the implementation. In general, a stack may be used to provide depth-first navigation during serialization and deserialization. During deserialization, we may also utilize a queue or an array to hold the serialized text or values.

**Tradeoffs**

The serialization and deserialization techniques for the binary tree issue need a trade-off between memory and readability in terms of trade-offs between CPU, memory, and readability. For big trees, the serialized string can be memory intensive. Because of its recursive structure, the DFS method used for traversal is efficient but may be difficult to implement for those who are not so used to this method.

**Solution in code with someone else’s solution that I built on it like the assignment description states:**

**struct TreeNode {**

**int val;**

**TreeNode\* left;**

**TreeNode\* right;**

**TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}**

**};**

**class Codec {**

**public:**

**// Encodes a tree to a single string.**

**std::string serialize(TreeNode\* root) {**

**std::string serialized;**

**serializeHelper(root, serialized);**

**return serialized;**

**}**

**// Decodes your encoded data to tree.**

**TreeNode\* deserialize(std::string data) {**

**std::istringstream iss(data);**

**return deserializeHelper(iss);**

**}**

**private:**

**// Helper function to serialize the tree**

**void serializeHelper(TreeNode\* root, std::string& serialized) {**

**if (root == nullptr) {**

**serialized += "null ";**

**return;**

**}**

**serialized += std::to\_string(root->val) + " ";**

**serializeHelper(root->left, serialized);**

**serializeHelper(root->right, serialized);**

**}**

**// Helper function to deserialize the tree**

**TreeNode\* deserializeHelper(std::istringstream& iss) {**

**std::string val;**

**iss >> val;**

**if (val == "null")**

**return nullptr;**

**TreeNode\* node = new TreeNode(std::stoi(val));**

**node->left = deserializeHelper(iss);**

**node->right = deserializeHelper(iss);**

**return node;**

**}**

**};**