4.

string serialize(TreeNode\* root) {

*// Check if the tree is empty*

if (!root) {

return "";

}

*// Initialize an empty string*

*// to store the serialized data*

string s = "";

*// Use a queue for*

*// level-order traversal*

queue<TreeNode\*> q;

*// Start with the root node*

q.push(root);

*// Perform level-order traversal*

while (!q.empty()) {

*// Get the front node in the queue*

TreeNode\* curNode = q.front();

q.pop();

*// Check if the current node is*

*// null and append "#" to the string*

if (curNode == nullptr) {

s += "#,";

} else {

*// Append the value of the*

*// current node to the string*

s += to\_string(curNode->val) + ",";

*// Push the left and right children*

*// to the queue for further traversal*

q.push(curNode->left);

q.push(curNode->right);

}

}

return s;

}

*// Decode the encoded*

*// data to a tree*

TreeNode\* deserialize(string data) {

*// Check if the*

*// serialized data is empty*

if (data.empty()) {

return nullptr;

}

*// Use a stringstream to*

*// tokenize the serialized data*

stringstream s(data);

string str;

*// Read the root value*

*// from the serialized data*

getline(s, str, ',');

TreeNode\* root = new TreeNode(stoi(str));

*// Use a queue for*

*// level-order traversal*

queue<TreeNode\*> q;

*// Start with the root node*

q.push(root);

*// Perform level-order traversal*

*// to reconstruct the tree*

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

*// Read the value of the left*

*// child from the serialized data*

getline(s, str, ',');

*// If the value is not "#", create a new*

*// left child and push it to the queue*

if (str != "#") {

TreeNode\* leftNode = new TreeNode(stoi(str));

node->left = leftNode;

q.push(leftNode);

}

*// Read the value of the right child*

*// from the serialized data*

getline(s, str, ',');

*// If the value is not "#", create a*

*// new right child and push it to the queue*

if (str != "#") {

TreeNode\* rightNode = new TreeNode(stoi(str));

node->right = rightNode;

q.push(rightNode);

}

}

return root;

}

void inorder(TreeNode\* root){

if(!root){

return;

}

inorder(root->left);

cout << root->val << " ";

inorder(root->right);

}

A screenshot of a test results

Description automatically generated