**Tree Hard Questions**

## **297. 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.

class Codec {

public:

// Encodes a tree to a single string.

string serialize(TreeNode\* root) {

string s = "";

queue<TreeNode \*> q;

q.push(root);

while(!q.empty()) {

TreeNode \*node = q.front();

q.pop();

if(!node) s += "n";

else {

s += to\_string(node->val);

q.push(node->left);

q.push(node->right);

}

if(!q.empty()) s += ',';

}

return s;

}

// Decodes your encoded data to tree.

TreeNode\* deserialize(string data) {

char del = ',';

stringstream ss(data);

string word = "";

queue<TreeNode \*> q;

TreeNode \*root = NULL;

while(!ss.eof()) {

if (!root) {

getline(ss, word, del);

if(word != "n") {

root = new TreeNode(stoi(word));

q.push(root);

q.push(root);

}

}

else{

getline(ss, word, del);

TreeNode \*node1 = q.front();

q.pop();

if(word != "n") {

TreeNode \*n = new TreeNode(stoi(word));

node1->left = n;

q.push(n);

q.push(n);

}

getline(ss, word, del);

TreeNode \*node2 = q.front();

q.pop();

if(word != "n") {

TreeNode \*n = new TreeNode(stoi(word));

node2->right = n;

q.push(n);

q.push(n);

}

}

}

return root;

}

};

**New Concept Learned**

How to Split String in C++:

stringstream ss(str);

string word = “”;

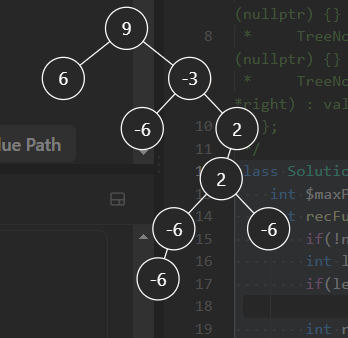
char del = ‘|’;

while(!ss.eof()) {

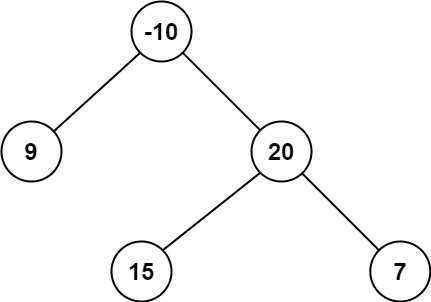
getline(ss, word, del);

}

# **Max Path Sum**



In the longest path sum, values are : [6, 9, -3, 2, 2]



Longest path sum values are : [15, 20, 7]

class Solution {

int $maxPathSum = INT\_MIN;

int recFunc(TreeNode \*node) {

if(!node) return 0;

int leftTreeMaxSum = recFunc(node->left);

if(leftTreeMaxSum < 0) leftTreeMaxSum = 0;

int rightTreeMaxSum = recFunc(node->right);

if(rightTreeMaxSum < 0) rightTreeMaxSum = 0;

int s = node->val + leftTreeMaxSum + rightTreeMaxSum;

$maxPathSum = max($maxPathSum, s);

return node->val + max(leftTreeMaxSum , rightTreeMaxSum);

}

public:

int maxPathSum(TreeNode\* root) {

recFunc(root);

return $maxPathSum;

}

};

Approach

1. If node is null then max path sum is 0.
2. Get max path sum of left and right tree using recursion function.
3. If path sum of trees are neg, then make them 0.
4. Add node val, left path sum and right path sum. Check if you are getting max path sum.
5. Return node val + max of left and right tree path sum. Need to return max because we need path.