# 题目

序列化是将数据结构或对象转换为一系列位的过程，以便它可以存储在文件或内存缓冲区中，或通过网络连接链路传输，以便稍后在同一个或另一个计算机环境中重建。

设计一个算法来序列化和反序列化二叉搜索树。对序列化/反序列化算法的工作方式没有限制。您只需确保二叉搜索树可以序列化为字符串，并且可以将该字符串反序列化为最初的二叉搜索树。

编码的字符串应尽可能紧凑。

示例 1：

输入：root = [2,1,3]

输出：[2,1,3]

示例 2：

输入：root = []

输出：[]

提示：

树中节点数范围是 [0, 104]

0 <= Node.val <= 104

题目数据 保证 输入的树是一棵二叉搜索树。

注意：不要使用类成员/全局/静态变量来存储状态。 你的序列化和反序列化算法应该是无状态的。

# 分析

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

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

 \* };

 \*/

class Codec {

private:

    TreeNode \*next\_node(stringstream &ss)

    {

        string tmp;

        ss >> tmp;

        if(tmp[0] == 'x')

        {

            return nullptr;

        }

        else

        {

            return new TreeNode(stoi(tmp));

        }

    }

public:

    // Encodes a tree to a single string.

    string serialize(TreeNode\* root) {

        string res;

        TreeNode \*cur = root;

        stack<TreeNode \*> stk;

        while(cur || !stk.empty())

        {

            while(cur)

            {

                res += to\_string(cur->val);

                res.push\_back(' ');

                stk.push(cur);

                cur = cur->left;

            }

            res.push\_back('x');

            res.push\_back(' ');

            cur = stk.top();

            stk.pop();

            cur = cur->right;

        }

        res.push\_back('x');

        res.push\_back(' ');

        return res;

    }

    // Decodes your encoded data to tree.

    TreeNode\* deserialize(string data) {

        stringstream ss(data);

        TreeNode \*root = next\_node(ss);

        TreeNode \*cur = root;

        stack<TreeNode\*> stk;

        while(cur || !stk.empty())

        {

            while(cur)

            {

                stk.push(cur);

                cur = cur->left = next\_node(ss);

            }

            cur = stk.top();

            stk.pop();

            cur = cur->right = next\_node(ss);

        }

        return root;

    }

};

// Your Codec object will be instantiated and called as such:

// Codec\* ser = new Codec();

// Codec\* deser = new Codec();

// string tree = ser->serialize(root);

// TreeNode\* ans = deser->deserialize(tree);

// return ans;