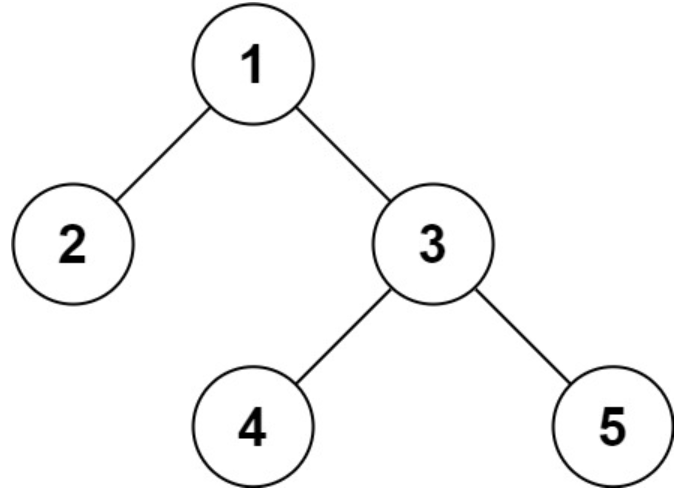
# 题目

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。

请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列/反序列化算法执行逻辑，只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

示例 1：



输入：root = [1,2,3,null,null,4,5]

输出：[1,2,3,null,null,4,5]

示例 2：

输入：root = []

输出：[]

示例 3：

输入：root = [1]

输出：[1]

示例 4：

输入：root = [1,2]

输出：[1,2]

提示：

输入输出格式与LeetCode目前使用的方式一致，详情请参阅LeetCode序列化二叉树的格式。你并非必须采取这种方式，也可以采用其他的方法解决这个问题。

树中结点数在范围 [0, 104] 内

-1000 <= Node.val <= 1000

注意：本题与主站 297 题相同：

https://leetcode-cn.com/problems/serialize-and-deserialize-binary-tree/

# 分析

## 方法一：栈+前序遍历

/\*\*

 \* 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, deser;

// TreeNode\* ans = deser.deserialize(ser.serialize(root));

## 方法二：层次遍历

/\*\*

\* 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 {

public:

// Encodes a tree to a single string.

string serialize(TreeNode\* root) {

queue<TreeNode\*> q;

q.push(root);

string str;

while(q.size()){

TreeNode\* t=q.front();

q.pop();

if(t)str+=to\_string(t->val)+" ";

else{

str+="\* ";

continue;

}

q.push(t->left);

q.push(t->right);

}

return str;

}

// Decodes your encoded data to tree.

TreeNode\* deserialize(string data) {

stringstream ss;

ss<<data;

string str1,str2;

ss>>str1;

if(str1=="\*")return NULL;

TreeNode\* root=new TreeNode(stoi(str1));

queue<TreeNode\*> q;

q.push(root);

while(ss>>str1){

if(!(ss>>str2))break;

TreeNode\* t=q.front();

q.pop();

if(str1=="\*")t->left=NULL;

else{

TreeNode\* node=new TreeNode(stoi(str1));

t->left=node;

q.push(node);

}

if(str2=="\*")t->right=NULL;

else{

TreeNode\* node=new TreeNode(stoi(str2));

t->right=node;

q.push(node);

}

}

return root;

}

};