# 题目

序列化二叉树的一种方法是使用前序遍历。当我们遇到一个非空节点时，我们可以记录下这个节点的值。如果它是一个空节点，我们可以使用一个标记值记录，例如#。

\_9\_

/ \

3 2

/ \ / \

4 1 # 6

/ \ / \ / \

# # # # # #

例如，上面的二叉树可以被序列化为字符串 "9,3,4,#,#,1,#,#,2,#,6,#,#"，其中 # 代表一个空节点。

给定一串以逗号分隔的序列，验证它是否是正确的二叉树的前序序列化。编写一个在不重构树的条件下的可行算法。

每个以逗号分隔的字符或为一个整数或为一个表示 null 指针的 '#' 。

你可以认为输入格式总是有效的，例如它永远不会包含两个连续的逗号，比如 "1,,3" 。

**示例 1:**

输入: "9,3,4,#,#,1,#,#,2,#,6,#,#"

输出: true

**示例 2:**

输入: "1,#"

输出: false

**示例 3:**

输入: "9,#,#,1"

输出: false

# 分析

## 方法一：栈

**思路：**

关于序列之类的可以使用栈。

通过简单观察可以发现，最后#的数量肯定比数字多一个。

所以考虑到前序遍历先根后左右的特性，思路就是，每次遇到数字就直接push，遇到#就pop。

当字符是#时，判断栈是否为空，如果是空的，再判断是否已经是字符串最后了，不是最后的话说明#的位置错了，反之说明是正确的前序遍历。

栈不为空就直接pop，i++是为了跳过分隔符。

类型选了bool是因为选啥都行，只是占位符而已，所以尽量选个简单的。

**代码：**

class Solution {

public:

bool isValidSerialization(string preorder) {

if (preorder.empty()) return false;

stack<bool> s;

for (int i = 0; i < preorder.size(); ++i) {

if (preorder[i] == '#') {

if (s.empty()) return i == preorder.size() - 1;

else {

s.pop();

i++;

}

}

else {

while (i < preorder.size() && preorder[i] != ',') i++;

s.push(0);

}

}

return false;

}

};