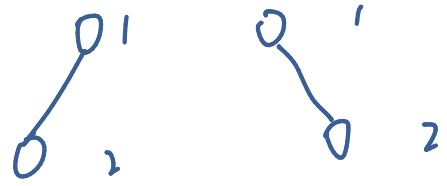
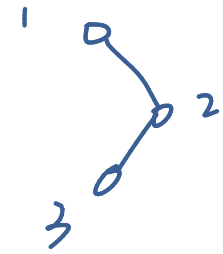
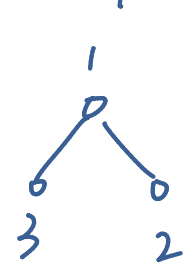
LC 100. Same tree







pre-order: [1, 2]

[1, 2, null]

可区分开来

[1, 2]

[1, null, 2]

[1, 2, 3]

[1, null, 2, 3, null]

[1, 2, 3]

in-order:

[2, 1]

[1, 2]

[1, 3, 2]

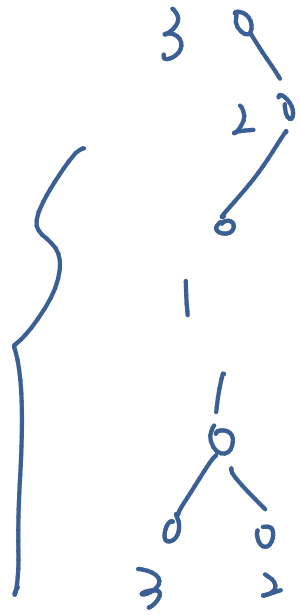
[3, 1, 2]

post-order: [2, 1]

[2, 1]

[3, 2, 1]

[3, 2, 1]



← in-order: [3, 1, 2]

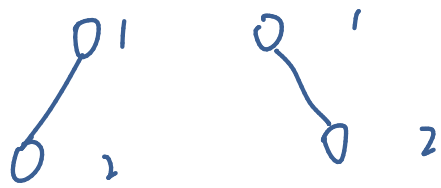
← in-order: [3, 1, 2]

∴ only use read-order won't help.

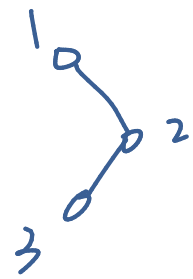
use "null" might help.

结论：和traversal的方式无关，关键是对空结点如何处理：要print，而不是跳过

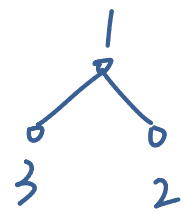
more example :



pre: [1, 2, null]

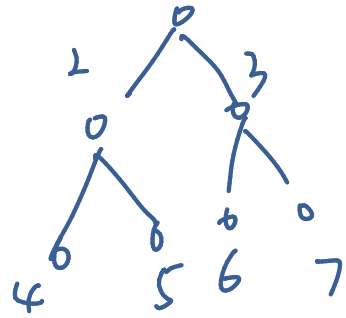


[1, null, 2]



[1, null, 2, 3, null]

or [1, null, 2, 3, null, null, null].



pre: [1, 2, 4, 5, 3, 6, 7]

代码思路：

1. 判断根结点是否为 None：

A、都为 None： return True

B、只有一个为None： return False

2. 用 queue 来进行 BFS （用stack进行DFS也OK.

如前文所说，traversal 的方式无所谓）：while 两个queue都不为空： blabla…

3. 两个queue，同时 pop()，

A、都为 None： 继续

B、只有一个为None： return False

C：没有一个为None：

1) val 相同： 继续

2) val 不同： return False

4. 同时 append 当前结点的左右子结点

5. while 结束后，判断两个queue是否都空了

A：都空了（长度为0）：return True

B： return False.

如果不都为0，"while"结束了，但两个树头有前一部分相同.

```python
class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        if p is None:
            if q is None:
                return True
            else:
                return False
        if p is None:
            if q is None:
                return True
            else:
                return False
        que_p = [p]
        que_q = [q]
        while que_p and que_q:
            q_cur = que_q.pop(0)
            p_cur = que_p.pop(0)
            if (q_cur and not p_cur) or (p_cur and not q_cur):
                return False
            if not q_cur and not p_cur:
                continue
            if q_cur.val != p_cur.val:
                return False
            que_p.append(p_cur.left)
            que_p.append(p_cur.right)
            que_q.append(q_cur.left)
            que_q.append(q_cur.right)
        if len(que_p) == len(que_q):
            return True
        else:
            return False
```