栈的压入、弹出序列

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1} 是该压栈序列对应的一个弹出序列，但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。

示例 1：

输入：`pushed = [1,2,3,4,5], popped = [4,5,3,2,1]`

输出：`true`

解释：我们可以按以下顺序执行：

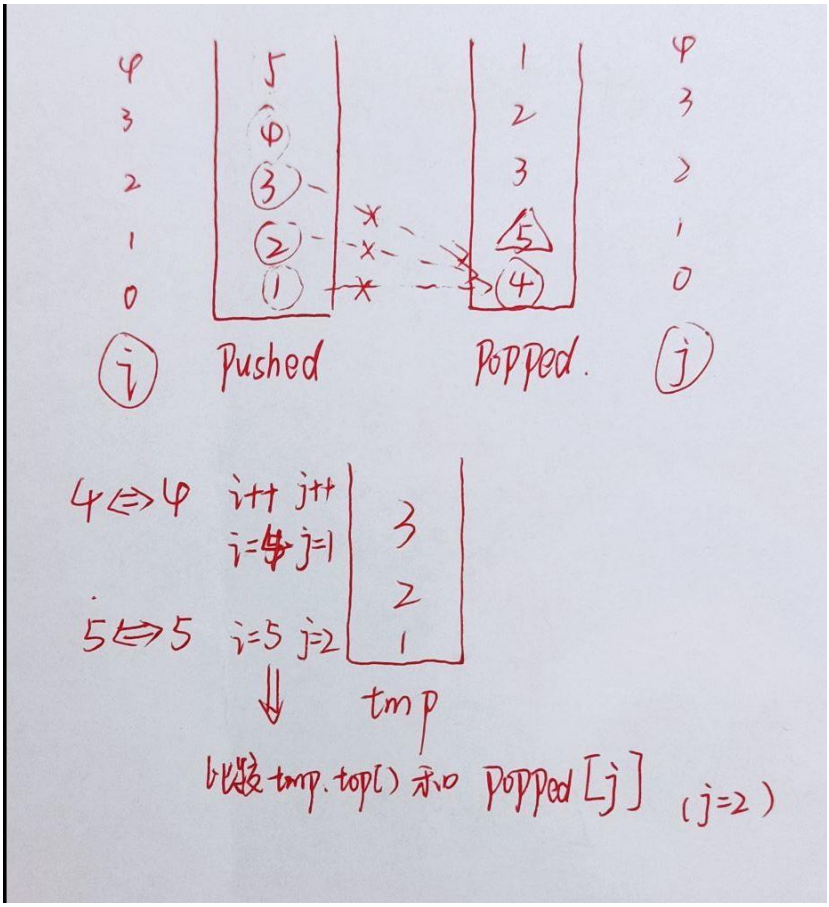
```
push(1), push(2), push(3), push(4), pop() -> 4,

push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
```

示例 2：

输入：`pushed = [1,2,3,4,5], popped = [4,3,5,1,2]`

输出：`false`

解释：1 不能在 2 之前弹出。



```
class Solution {
public:
    bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {
        //需要借助临时栈作为辅助栈，来判断栈压入，弹出序列是否正确
        stack<int> tmp;
        int i=0;
        int j=0;//初始化 i,j 为 0
        while(i<pushed.size())//压入序列中循环查找
        {
            if(pushed[i]!=popped[j])
            {
                tmp.push(pushed[i]);
                i++;//将不匹配的都放入临时栈中
            }
            else//出现相等的情况
            {
                i++;
                j++;//将相等的部分跳过
                while(!tmp.empty()&&tmp.top()==popped[j])//此处需要注意两个逻辑判断先后顺序，需要先对 tmp 栈是否为空进行判断，若为空，则无法进行 top 操作，否则会导致越界
                {
                    tmp.pop();
                    j++;
                }
            }
        }
        return tmp.empty();//最后根据 tmp 是否为空来判断 是否都有所对应的元素与之匹配
    }
};
```