题目 栈的压入、弹出序列

考点 举例让抽象具体化 热点指数 60348 通过率 28.69%

具体题目

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可能为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1,2,3,4,5是某栈的压入顺序，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。（注意：这两个序列的长度是相等的）

【思路】借用一个辅助的栈，遍历压栈顺序，先讲第一个放入栈中，这里是1，然后判断栈顶元素是不是出栈顺序的第一个元素，这里是4，很显然1≠4，所以我们继续压栈，直到相等以后开始出栈，出栈一个元素，则将出栈顺序向后移动一位，直到不相等，这样循环等压栈顺序遍历完成，如果辅助栈还不为空，说明弹出序列不是该栈的弹出顺序。

举例：

入栈1,2,3,4,5

出栈4,5,3,2,1

首先1入辅助栈，此时栈顶1≠4，继续入栈2

此时栈顶2≠4，继续入栈3

此时栈顶3≠4，继续入栈4

此时栈顶4＝4，出栈4，弹出序列向后一位，此时为5，,辅助栈里面是1,2,3

此时栈顶3≠5，继续入栈5

此时栈顶5=5，出栈5,弹出序列向后一位，此时为3，,辅助栈里面是1,2,3

….

依次执行，最后辅助栈为空。如果不为空说明弹出序列不是该栈的弹出顺序。

```
 import java.util.ArrayList;
import java.util.Stack;
public class Solution {
    public boolean IsPopOrder(int [] pushA,int [] popA) {
        if(pushA.length == 0 || popA.length == 0)
            return false;
        Stack<Integer> s = new Stack<Integer>();
        //用于标识弹出序列的位置
        int popIndex = 0;
        for(int i = 0; i< pushA.length;i++){
            s.push(pushA[i]);
            //如果栈不为空，且栈顶元素等于弹出序列
            while(!s.empty() &&s.peek() == popA[popIndex]){
                //出栈
                s.pop();
                //弹出序列向后一位
                popIndex++;
            }
        }
        return s.empty();
    }
}
```

```
/*思路：先循环将pushA中的元素入栈，遍历的过程中检索popA可以pop的元素
**如果循环结束后栈还不空，则说明该序列不是pop序列。
**文字有点难说明白，看代码。
*/
import java.util.ArrayList;
import java.util.Stack;
public class Solution {
    public boolean IsPopOrder(ArrayList<Integer> pushA,ArrayList<Integer> popA) {
        Stack stack = new Stack();
        if( pushA.size() == 0 && popA.size() == 0 ) return true;
        for( int i=0,j=0; i < pushA.size(); i++ ){
```

```
            stack.push( pushA.get(i) );
            while( ( !stack.empty() )&& ( stack.peek() == popA.get(j) ) ){
                stack.pop();
                j ++;
            }
        }

        return stack.empty() == true;
    }
}
```

题目描述 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可能为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1,2,3,4,5是某栈的压入顺序，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。（注意：这两个序列的长度是相等的） 解题思路 一开始都看不懂题目... 后来才好像明白是什么个意思... 假设有一串数字要将他们压栈: 1 2 3 4 5 如果这个栈是很大很大，那么一次性全部压进去，再出栈：5 4 3 2 1 但是，如果这个栈高度为4，会发生什么？ 1 2 3 4都顺利入栈，但是满了，那么要先出栈一个，才能入栈，那么就是先出4，然后压入5，随后再全部出栈：4 5 3 2 1 那么我总结了所有可能的出栈情况: 5 4 3 2 1//栈高度为5 4 5 3 2 1//栈高度为4 3 4 5 2 1//栈高度为3 2 3 4 5 1//栈高度为2 1 2 3 4 5//栈高度为1 借助一个辅助的栈，遍历压栈的顺序，依次放进辅助栈中。 对于每一个放进栈中的元素，栈顶元素都与出栈的popIndex对应位置的元素进行比较，是否相等，相等则popIndex++，再判断，直到为空或者不相等为止。 我的答案

```
import java.util.ArrayList;
import java.util.Stack;
public class Solution {
    public boolean IsPopOrder(int [] pushA,int [] popA) {
        //数组为空的情况
        if(pushA.length == 0 || popA.length == 0){
            return false;
        }
        //弹出序列的下表索引
        int popIndex = 0;
        //辅助栈
        Stack<Integer> stack = new Stack<Integer>();
        for(int i=0;i<pushA.length;i++){
            //不停地将pushA中的元素压入栈中，一旦栈顶元素与popA相等了，则开始出栈
            //不相等则继续入栈
            stack.push(pushA[i]);
            while(!stack.isEmpty() && stack.peek()==popA[popIndex]){
                stack.pop();
                popIndex++;
            }
        }
        //栈中没有元素了说明元素全部一致，并且符合弹出顺序，那么返回true
        return stack.isEmpty();
    }
}
```