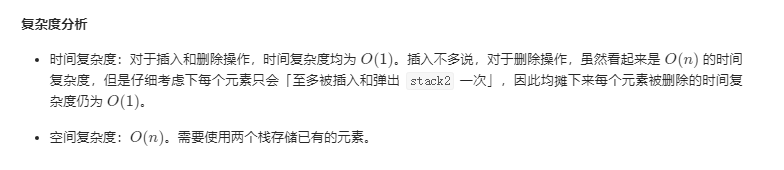
栈

1. 用栈实现队列（232）【用两个栈实现队列（剑指9）】





232可运行版本

import java.util.Stack;  
public class MyQueue {  
 Stack<Integer> s1, s2;  
 */\*\* Initialize your data structure here. \*/* public MyQueue() {  
 s1 = new Stack<>();  
 s2 = new Stack<>();  
 }  
 public void push(int x) {  
 s1.push(x);  
 }  
 */\*\* Removes the element from in front of queue and returns that element. \*/* public int pop() {  
 if (s2.isEmpty()) {  
 while (!s1.isEmpty()){  
 s2.push(s1.pop());  
 }  
 }  
 if (!s2.isEmpty()) return s2.pop();  
 throw new RuntimeException("There are no elements in the queue!");  
 }  
 */\*\* Get the front element. \*/* public int peek() {  
 if (s2.isEmpty()) {  
 while (!s1.isEmpty()){  
 s2.push(s1.pop());  
 }  
 }  
 if (!s2.isEmpty()) return s2.peek();  
 throw new RuntimeException("There are no elements in the queue!");  
 }  
 */\*\* Returns whether the queue is empty. \*/* public boolean empty() {  
 return s1.isEmpty() && s2.isEmpty();  
 }  
 public static void main(String[] args){  
 MyQueue queue = new MyQueue();  
 queue.push(1);  
 queue.push(2);  
 System.*out*.println(queue.peek()); *// 返回 1* System.*out*.println(queue.pop()); *// 返回 1* System.*out*.println(queue.empty()); *// 返回 false* }  
}

剑指9版本

**class** CQueue {

    Deque<Integer> stack1;

    Deque<Integer> stack2;

**public** CQueue() {

        stack1 = **new** LinkedList<Integer>();

        stack2 = **new** LinkedList<Integer>();

    }

**public** **void** appendTail(**int** value) {

        stack1.push(value);

    }

**public** **int** deleteHead() {

*// 如果第二个栈为空*

**if** (stack2.isEmpty()) {

**while** (!stack1.isEmpty()) {

                stack2.push(stack1.pop());

            }

        }

**if** (stack2.isEmpty()) **return** -1;

**else** {

**int** deleteItem = stack2.pop();

**return** deleteItem;

        }

    }

}

*/\*\**

*\* Your CQueue object will be instantiated and called as such:*

*\* CQueue obj = new CQueue();*

*\* obj.appendTail(value);*

*\* int param\_2 = obj.deleteHead();*

*\*/*

1. 用队列实现栈（225）import java.util.LinkedList;  
   import java.util.Queue;  
   public class MyStack {  
    Queue<Integer> que;  
    */\*\* Initialize your data structure here. \*/* public MyStack() {  
    que = new LinkedList<>();  
    }  
    */\*\* Push element x onto stack. push方法：先进来一个1，此时queue：[1] 在进来一个2，此时queue：[1,2] 因为queue的性质是头（左边）出尾（右边）进，如果此时queue.poll()的话，取出的是1，此时queue：[2]，然而这不是我们想要的结果， 我们想要的是取出2剩下1（因为模拟的是栈stack，后进先出）=》取出2得queue：[1] 所以利用循环，把queue：[1,2] 里面的2不动，把1调到后面（队列尾部），此时queue：[2，1]，在调用 queue.poll()的话，取出的就是2，此时的效果刚好满足用队列模拟栈的后进先出的特性\*/* public void push(int x) {  
    que.add(x);  
    for (int i = 1; i < que.size(); i++){  
    que.add(que.remove());  
    }  
    }  
    */\*\* Removes the element on top of the stack and returns that element. \*/* public int pop() {  
    return que.poll();  
    }  
     
    */\*\* Get the top element. \*/* public int top() {  
    return que.peek();  
    }  
     
    */\*\* Returns whether the stack is empty. \*/* public boolean empty() {  
    return que.size() == 0;  
    }  
    public static void main(String[] args){  
    MyStack sta = new MyStack();  
    sta.push(1);  
    sta.push(2);  
    System.*out*.println(sta.top());  
    System.*out*.println(sta.pop());  
    System.*out*.println(sta.empty());  
    }  
   }
2. 最小栈（155）【包含min函数的栈（剑指30）】

甜姨总结：

<https://leetcode-cn.com/problems/min-stack/solution/3-chong-fang-fa-shi-xian-bi-xu-miao-dong-by-sweeti/>

思路1，用辅助栈（7ms）

首先定义一个 「数据栈」来支持常规的 push、pop、top 操作；

再定义一个「辅助栈」，其栈顶一直保持为当前的最小值，以支持常数时间复杂度的 getMin 操作。

**class** MinStack {

**private** Stack<Integer> dataStack;

**private** Stack<Integer> minStack;

*/\*\* initialize your data structure here. \*/*

**public** MinStack() {

        dataStack = **new** Stack<>();

        minStack = **new** Stack<>();

    }

**public** **void** push(**int** x) {

        dataStack.push(x);

**if** (minStack.isEmpty() || x <= minStack.peek() ) {

            minStack.push(x);

        }

    }

**public** **void** pop() {

**int** x = dataStack.pop();

**if** (x == minStack.peek()) {

            minStack.pop();

        }

    }

**public** **int** top() {

**return** dataStack.peek();

    }

**public** **int** getMin() {

**return** minStack.peek();

    }

}

*/\*\**

*\* Your MinStack object will be instantiated and called as such:*

*\* MinStack obj = new MinStack();*

*\* obj.push(x);*

*\* obj.pop();*

*\* int param\_3 = obj.top();*

*\* int param\_4 = obj.getMin();*

*\*/*

思路2：不利用辅助栈，自建Node（5ms）

**class** MinStack {

**private** Node head;

**public** MinStack() {

    }

**public** **void** push(**int** x) {

*// 新建节点插入链表头部，作为新的头节点，存储当前的元素值和最小值，并且指向之前的头节点。*

**if** (head == null) {

            head = **new** Node(x, x);

        } **else** {

            head = **new** Node(x, Math.min(x, head.min), head);

        }

    }

**public** **void** pop() {

*// 删除链表头节点*

        head = head.next;

    }

**public** **int** top() {

*// 返回头节点中存储的元素值*

**return** head.val;

    }

**public** **int** getMin() {

*// 返回头节点中存储的最小值*

**return** head.min;

    }

**private** **static** **class** Node {

**int** val; *// 元素值*

**int** min; *// 最小值*

        Node next; *// 后继节点*

**public** Node(**int** val, **int** min) {

**this**(val, min, null);

        }

**public** Node(**int** val, **int** min, Node next) {

**this**.val = val;

**this**.min = min;

**this**.next = next;

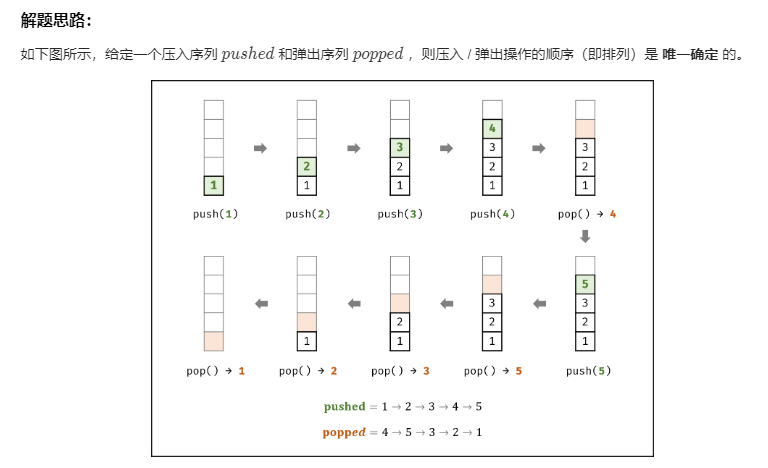
        }

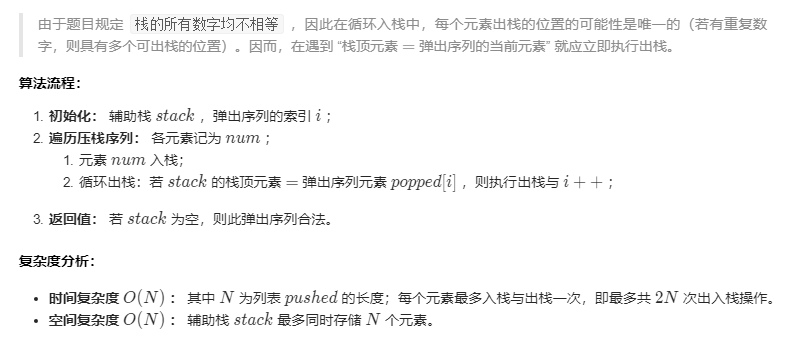
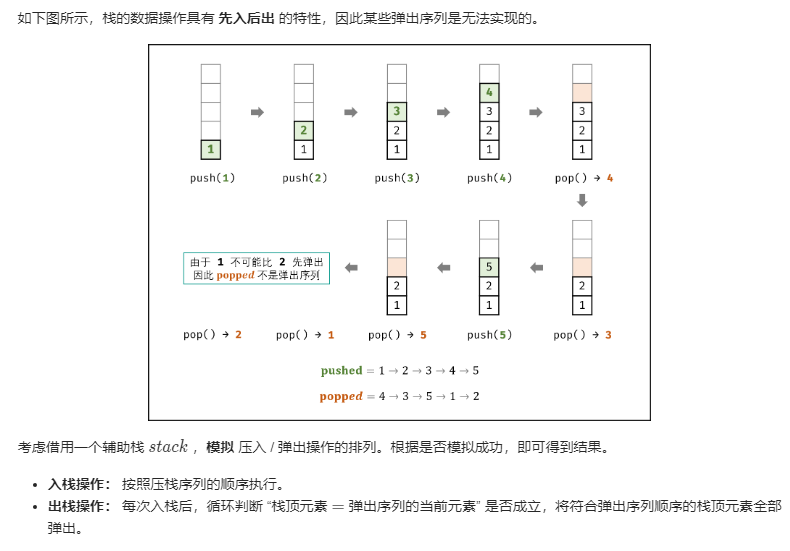
    }

}

1. 验证栈序列（946）【栈的压入、弹出序列（剑指31）】

<https://leetcode-cn.com/problems/zhan-de-ya-ru-dan-chu-xu-lie-lcof/solution/mian-shi-ti-31-zhan-de-ya-ru-dan-chu-xu-lie-mo-n-2/>





class Solution {

    public boolean validateStackSequences(int[] pushed, int[] popped) {

        Stack<Integer> sta = new Stack<>();

        int i = 0;

        for (int num : pushed) {

            sta.push(num);

            while (!sta.isEmpty() && sta.peek() == popped[i]){

                sta.pop();

                i++;

            }

        }

        return sta.isEmpty();

    }

}