Leetcode(python版) 栈简单题

20有效的括号

题目：给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。
2. 左括号必须以正确的顺序闭合。

注意空字符串可被认为是有效字符串。

**示例 1:**

**输入:** "()"

**输出:** true

**示例 2:**

**输入:** "()[]{}"

**输出:** true

**示例 3:**

**输入:** "(]"

**输出:** false

**示例 4:**

**输入:** "([)]"

**输出:** false

**示例 5:**

**输入:** "{[]}"

**输出:** true

思路：

1)空和奇数个直接排除   
2)剩下偶数个，若第一个就是右括号直接排除，不是右括号再入栈

代码：

1. **class** Solution(object):
2. **def** isValid(self, s):
3. """
4. :type s: str
5. :rtype: bool
6. """
7. d={'(':')','{':'}','[':']'}
8. stack=[]
9. **if** len(s)%2==1:
10. **return** False
11. **if** len(s)==0:
12. **return** True
13. **for** i **in** s:
14. **if** i **in** d:
15. stack.append(i)
16. **else**:
17. **if** **not** stack **or** d[stack.pop()]!=i:
18. **return** False
19. **return** stack==[]

155最小栈

题目：

设计一个支持 push，pop，top 操作，并能在常数时间内检索到最小元素的栈。

* push(x) -- 将元素 x 推入栈中。
* pop() -- 删除栈顶的元素。
* top() -- 获取栈顶元素。
* getMin() -- 检索栈中的最小元素。

**示例:**

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); --> 返回 -3.

minStack.pop();

minStack.top(); --> 返回 0.

minStack.getMin(); --> 返回 -2.

思路：

1. 代码：**class** MinStack(object):
3. **def** \_\_init\_\_(self):
4. """
5. initialize your data structure here.
6. """
7. self.stack=[]
8. self.min=None
10. **def** push(self, x):
11. """
12. :type x: int
13. :rtype: void
14. """
15. self.stack.append(x)
16. **if** self.min==None **or** x<self.min:
17. self.min=x

20. **def** pop(self):
21. """
22. :rtype: void
23. """
24. popItem=self.stack.pop()
25. **if** len(self.stack)==0:
26. self.min=None
27. **return** popItem
28. **if** popItem==self.min:
29. self.min=self.stack[0]
30. **for** i **in** self.stack:
31. **if** i<self.min:
32. self.min=i
33. **return** popItem
35. **def** top(self):
36. """
37. :rtype: int
38. """
39. **return** self.stack[-1]
41. **def** getMin(self):
42. """
43. :rtype: int
44. """
45. **return** self.min

48. # Your MinStack object will be instantiated and called as such:
49. # obj = MinStack()
50. # obj.push(x)
51. # obj.pop()
52. # param\_3 = obj.top()
53. # param\_4 = obj.getMin()

225用队列实现栈

题目：

使用队列实现栈的下列操作：

* push(x) -- 元素 x 入栈
* pop() -- 移除栈顶元素
* top() -- 获取栈顶元素
* empty() -- 返回栈是否为空

**注意:**

* 你只能使用队列的基本操作-- 也就是 push to back, peek/pop from front, size, 和 is empty 这些操作是合法的。
* 你所使用的语言也许不支持队列。 你可以使用 list 或者 deque（双端队列）来模拟一个队列 , 只要是标准的队列操作即可。
* 你可以假设所有操作都是有效的（例如, 对一个空的栈不会调用 pop 或者 top 操作）。

思路：

代码：

1. **from** collections **import** deque
2. **class** MyStack(object):
4. **def** \_\_init\_\_(self):
5. """
6. Initialize your data structure here.
7. """
8. self.queue=deque([])
10. **def** push(self, x):
11. """
12. Push element x onto stack.
13. :type x: int
14. :rtype: void
15. """
16. self.queue.append(x)
17. **for** \_ **in** range(0,len(self.queue)-1):
18. self.queue.append(self.queue.popleft())
20. **def** pop(self):
21. """
22. Removes the element on top of the stack and returns that element.
23. :rtype: int
24. """
25. **return** self.queue.popleft()
27. **def** top(self):
28. """
29. Get the top element.
30. :rtype: int
31. """
32. **return** self.queue[0]
34. **def** empty(self):
35. """
36. Returns whether the stack is empty.
37. :rtype: bool
38. """
39. **return** **not** self.queue


43. # Your MyStack object will be instantiated and called as such:
44. # obj = MyStack()
45. # obj.push(x)
46. # param\_2 = obj.pop()
47. # param\_3 = obj.top()
48. # param\_4 = obj.empty()

232用栈实现队列

题目：

使用栈实现队列的下列操作：

* push(x) -- 将一个元素放入队列的尾部。
* pop() -- 从队列首部移除元素。
* peek() -- 返回队列首部的元素。
* empty() -- 返回队列是否为空。

**示例:**

MyQueue queue = new MyQueue();

queue.push(1);

queue.push(2);

queue.peek(); // 返回 1

queue.pop(); // 返回 1

queue.empty(); // 返回 false

**说明:**

* 你只能使用标准的栈操作 -- 也就是只有 push to top, peek/pop from top, size, 和 is empty 操作是合法的。
* 你所使用的语言也许不支持栈。你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。
* 假设所有操作都是有效的 （例如，一个空的队列不会调用 pop 或者 peek 操作）。

思路:列表（list）的功能与栈一样,stack1主要进出队列，stack2作中转

代码：

1. **class** MyQueue(object):
3. **def** \_\_init\_\_(self):
4. """
5. Initialize your data structure here.
6. """
7. self.stack1=[]
8. self.stack2=[]

11. **def** push(self, x):
12. """
13. Push element x to the back of queue.
14. :type x: int
15. :rtype: void
16. """
17. self.stack1.append(x)
19. **def** pop(self):
20. """
21. Removes the element from in front of queue and returns that element.
22. :rtype: int
23. """
24. **while** self.stack1:
25. self.stack2.append(self.stack1.pop())
26. res=self.stack2.pop()
27. **while** self.stack2:
28. self.stack1.append(self.stack2.pop())
29. **return** res
31. **def** peek(self):
32. """
33. Get the front element.
34. :rtype: int
35. """
36. **return** self.stack1[0]

39. **def** empty(self):
40. """
41. Returns whether the queue is empty.
42. :rtype: bool
43. """
44. **return** **not** self.stack1

47. # Your MyQueue object will be instantiated and called as such:
48. # obj = MyQueue()
49. # obj.push(x)
50. # param\_2 = obj.pop()
51. # param\_3 = obj.peek()
52. # param\_4 = obj.empty()

496下一个更大元素I

题目：

给定两个**没有重复元素**的数组 nums1 和 nums2 ，其中nums1 是 nums2 的子集。找到 nums1 中每个元素在 nums2 中的下一个比其大的值。

nums1 中数字 **x** 的下一个更大元素是指 **x** 在 nums2 中对应位置的右边的第一个比 **x**大的元素。如果不存在，对应位置输出-1。

**示例 1:**

**输入:** **nums1** = [4,1,2], **nums2** = [1,3,4,2].

**输出:** [-1,3,-1]

**解释:**

对于num1中的数字4，你无法在第二个数组中找到下一个更大的数字，因此输出 -1。

对于num1中的数字1，第二个数组中数字1右边的下一个较大数字是 3。

对于num1中的数字2，第二个数组中没有下一个更大的数字，因此输出 -1。

**示例 2:**

**输入:** **nums1** = [2,4], **nums2** = [1,2,3,4].

**输出:** [3,-1]

**解释:**

  对于num1中的数字2，第二个数组中的下一个较大数字是3。

对于num1中的数字4，第二个数组中没有下一个更大的数字，因此输出 -1。

**注意:**

1. nums1和nums2中所有元素是唯一的。
2. nums1和nums2 的数组大小不超过1000

思路：遍历，但是这边用到index,这个方法还是值得记住的

代码：

1. **class** Solution(object):
2. **def** nextGreaterElement(self, findNums, nums):
3. """
4. :type findNums: List[int]
5. :type nums: List[int]
6. :rtype: List[int]
7. """
8. res=[]
9. n=len(nums)
10. flag=False
11. **for** e **in** findNums:
12. indexofe=nums.index(e)
13. **for** e2 **in** nums[indexofe+1:]:
14. **if** e2>e:
15. res.append(e2)
16. flag=True
17. **break**
18. **if** **not** flag:
19. res.append(-1)
20. **else**:
21. flag=False
22. **return** res

682棒球比赛

题目：

你现在是棒球比赛记录员。  
给定一个字符串列表，每个字符串可以是以下四种类型之一：  
1.整数（一轮的得分）：直接表示您在本轮中获得的积分数。  
2. "+"（一轮的得分）：表示本轮获得的得分是前两轮有效 回合得分的总和。  
3. "D"（一轮的得分）：表示本轮获得的得分是前一轮有效 回合得分的两倍。  
4. "C"（一个操作，这不是一个回合的分数）：表示您获得的最后一个有效 回合的分数是无效的，应该被移除。  
每一轮的操作都是永久性的，可能会对前一轮和后一轮产生影响。  
你需要返回你在所有回合中得分的总和。

**示例 1:**

**输入:** ["5","2","C","D","+"]

**输出:** 30

**解释:**

第1轮：你可以得到5分。总和是：5。

第2轮：你可以得到2分。总和是：7。

操作1：第2轮的数据无效。总和是：5。

第3轮：你可以得到10分（第2轮的数据已被删除）。总数是：15。

第4轮：你可以得到5 + 10 = 15分。总数是：30。

**示例 2:**

**输入:** ["5","-2","4","C","D","9","+","+"]

**输出:** 27

**解释:**

第1轮：你可以得到5分。总和是：5。

第2轮：你可以得到-2分。总数是：3。

第3轮：你可以得到4分。总和是：7。

操作1：第3轮的数据无效。总数是：3。

第4轮：你可以得到-4分（第三轮的数据已被删除）。总和是：-1。

第5轮：你可以得到9分。总数是：8。

第6轮：你可以得到-4 + 9 = 5分。总数是13。

第7轮：你可以得到9 + 5 = 14分。总数是27。

思路：无非就是各种条件去做判断

代码：

1. **class** Solution(object):
2. **def** calPoints(self, ops):
3. """
4. :type ops: List[str]
5. :rtype: int
6. """
7. point\_list=[]
8. **for** op **in** ops:
9. **if** op=='+':
10. point\_list.append(sum(point\_list[-2:]))
11. **elif** op=='C':
12. **if** point\_list:
13. **del** point\_list[-1]
14. **elif** op=='D':
15. **if** point\_list:
16. point\_list.append(point\_list[-1]\*2)
17. **else**:
18. point\_list.append(int(op))
19. **return** sum(point\_list)

844比较含退格的字符串

题目：

给定 S 和 T 两个字符串，当它们分别被输入到空白的文本编辑器后，判断二者是否相等，并返回结果。 # 代表退格字符。

**示例 1：**

**输入：**S = "ab#c", T = "ad#c"

**输出：**true

**解释：**S 和 T 都会变成 “ac”。

**示例 2：**

**输入：**S = "ab##", T = "c#d#"

**输出：**true

**解释：**S 和 T 都会变成 “”。

**示例 3：**

**输入：**S = "a##c", T = "#a#c"

**输出：**true

**解释：**S 和 T 都会变成 “c”。

**示例 4：**

**输入：**S = "a#c", T = "b"

**输出：**false

**解释：**S 会变成 “c”，但 T 仍然是 “b”。

**提示：**

1. 1 <= S.length <= 200
2. 1 <= T.length <= 200
3. S 和 T 只含有小写字母以及字符 '#'。

思路：遇到#，字符串不为空，就删除最后一个字符。如果不是#号，就拼接到字符串的最后。把两个字符串都求出来，然后比较就好。

代码：

1. **class** Solution(object):
2. **def** backspaceCompare(self, S, T):
3. """
4. :type S: str
5. :type T: str
6. :rtype: bool
7. """
8. ans\_s=""
9. ans\_t=""
10. **for** s **in** S:
11. **if** s=='#':
12. **if** ans\_s:
13. ans\_s=ans\_s[:-1]
14. **else**:
15. ans\_s+=s
16. **for** t **in** T:
17. **if** t=='#':
18. **if** ans\_t:
19. ans\_t=ans\_t[:-1]
20. **else**:
21. ans\_t+=t
22. **return** ans\_s==ans\_t