

栈、队列、堆

```
In [ ]: 1 # 232. 用栈实现队列
2 class MyQueue:
3
4     def __init__(self):
5         """
6         Initialize your data structure here.
7         """
8         self.stack1 = [] # 入栈
9         self.stack2 = [] # 出栈
10
11     def push(self, x: int) -> None:
12         """
13         Push element x to the back of queue.
14         """
15         self.stack1.append(x)
16
17     def pop(self) -> int:
18         """
19         Removes the element from in front of queue and returns that element.
20         """
21         if self.stack2 == []:
22             while self.stack1:
23                 self.stack2.append(self.stack1.pop())
24         return self.stack2.pop()
25
26
27     def peek(self) -> int:
28         """
29         Get the front element.
30         """
31         if self.stack2 == []:
32             while self.stack1:
33                 self.stack2.append(self.stack1.pop())
34         return self.stack2[-1]
35
36
37     def empty(self) -> bool:
38         """
39         Returns whether the queue is empty.
40         """
41         return self.stack2 == [] and self.stack1 == []
42
43 # 225. 用队列实现栈
44 from collections import deque
45 class MyStack:
46
47     def __init__(self):
48         """
49         Initialize your data structure here.
50         """
51         self.queue1 = deque()
52         self.queue2 = deque() # 辅助删除
53
54     def push(self, x: int) -> None:
55         """
56         Push element x onto stack.
57         """
58         self.queue1.append(x)
59
60     def pop(self) -> int:
61         """
62         Removes the element on top of the stack and returns that element.
63         """
64         size = len(self.queue1)
65         for i in range(size-1):
66             self.queue2.append(self.queue1.popleft())
67         result = self.queue1.popleft()
68         self.queue1, self.queue2 = self.queue2, self.queue1
69         return result
70
71
```

```

72     def top(self) -> int:
73         """
74         Get the top element.
75         """
76         return self.queue1[-1]
77
78     def empty(self) -> bool:
79         """
80         Returns whether the stack is empty.
81         """
82         return len(self.queue1) == 0
83
84 # 20. 有效的括号
85 class Solution:
86     def isValid(self, s: str) -> bool:
87         def leftcode(c):
88             if c == ']': return '['
89             if c == ')': return '('
90             if c == '}': return '{'
91         stack = []
92         for c in s:
93             if c in '[((': # 输入是左括号
94                 stack.append(c)
95             else: # 输入是右括号
96                 if stack and leftcode(c) == stack[-1]: stack.pop()
97                 else: return False
98         return stack == []
99
100 # 1047. 删除字符串中的所有相邻重复项
101 class Solution:
102     def removeDuplicates(self, s: str) -> str:
103         stack = []
104         for c in s:
105             if stack and c == stack[-1]: stack.pop()
106             else: stack.append(c)
107         return ''.join(stack)
108
109 # 150. 逆波兰表达式求值
110 class Solution:
111     def evalRPN(self, tokens: List[str]) -> int:
112         result = []
113         for token in tokens:
114             if token not in '+-*/':
115                 result.append(token)
116             else:
117                 temp1 = result.pop()
118                 temp2 = result.pop()
119                 res = eval(temp2+token+temp1)
120                 result.append(str(int(res)))
121         return int(result[-1])
122
123 # 239. 滑动窗口最大值
124 class MyQueue():
125     def __init__(self):
126         self.queue = []
127     def pop(self, x):
128         if self.queue and x == self.queue[0]:
129             self.queue.pop(0)
130     def push(self, x):
131         while self.queue and x > self.queue[-1]:
132             self.queue.pop()
133         self.queue.append(x)
134     def front(self):
135         return self.queue[0]
136 class Solution:
137     def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
138         # 写一个单调栈，4个功能
139         res = []
140         que = MyQueue()
141         for i in range(k): que.push(nums[i])
142         res.append(que.front())
143         for i in range(k, len(nums)):

```

```
144         que.pop(nums[i-k])
145         que.push(nums[i])
146         res.append(que.front())
147     return res
148
149 # 347. 前 K 个高频元素
150 import heapq
151 class Solution:
152     def topKFrequent(self, nums: List[int], k: int) -> List[int]:
153         # dic = {i:nums.count(i) for i in set(nums)}
154         # # 维护一个长为k最小堆
155         # que = []
156         # for key,frequent in dic.items():
157         #     heapq.heappush(que, (frequent, key))
158         #     if len(que) > k:
159         #         heapq.heappop(que)
160         # res = []
161         # for _ in range(k):
162         #     res.append(heapq.heappop(que)[1])
163         # return res
164         dic = {i:nums.count(i) for i in set(nums)}
165         que = []
166         for key, fre in dic.items():
167             heapq.heappush(que, (fre, key))
168             if len(que) > k:
169                 heapq.heappop(que)
170         return [heapq.heappop(que)[1] for _ in range(k)]
```

单调栈

In []:

```

1  ##### 739 每日温度
2  def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
3      #单调栈
4      n = len(temperatures)
5      res = [0]*n # 存放结果
6      stack = [] # 存放索引而非元素值stack
7      for i in range(n-1,-1,-1):
8          while s and temperatures[stack[-1]] <= temperatures[i]:
9              stack.pop()
10         res[i] = 0 if not stack else stack[-1]-i
11         stack.append(i)
12     return res
13
14 ##### 下一个更大的数
15 def nextGreaterElements(self, nums: List[int]) -> List[int]:
16     n = len(nums)
17     res = [0]*n
18     stack = []
19     for i in range(n-1,-1,-1):
20         while stack and stack[-1] <= nums[i]:
21             stack.pop()
22         res[i] = -1 if not stack else stack[-1]
23         stack.append(nums[i])
24     return res
25
26 ##### 503 循环数组，下一个更大的元素
27 def nextGreaterElements(self, nums: List[int]) -> List[int]:
28     n = len(nums)
29     res = [0]*n
30     stack = []
31     for i in range(2*n-1,-1,-1):
32         while stack and stack[-1] <= nums[i%n]:
33             stack.pop()
34         res[i%n] = -1 if not stack else stack[-1]
35         stack.append(nums[i%n])
36     return res
37
38
39 ##### 42 接雨水
40 def trap(self, height: List[int]) -> int:
41     # 暴力求解 (319/320) O(N^2)
42     n = len(height)
43     res = 0
44     for i in range(1,n-1):
45         leftmax, rightmax = 0,0
46         for j in range(i,n):
47             rightmax = max(rightmax, height[j])
48         for j in range(i,-1,-1):
49             leftmax = max(leftmax, height[j])
50         res += min(leftmax, rightmax)-height[i]
51     return res
52
53     # 暴力+备忘录 (dp全部算完) O(N)
54     n = len(height)
55     if n == 0: return 0
56     res = 0
57     leftmax = [0]*n

```

```
58     rightmax = [0]*n
59     leftmax[0] = height[0]
60     rightmax[-1] = height[-1]
61     for i in range(1,n):
62         leftmax[i] = max(height[i], leftmax[i-1])
63     for i in range(n-2,-1,-1):
64         rightmax[i] = max(height[i], rightmax[i+1])
65     for i in range(1,n-1):
66         res += min(leftmax[i], rightmax[i]) - height[i]
67     return res
68
69     # 双指针
70     if not height: return 0
71     n = len(height)
72     left, right = 0, n-1
73     res = 0
74     leftmax, rightmax = height[0], height[-1]
75     while left <= right:
76         leftmax = max(leftmax, height[left])
77         rightmax = max(rightmax, height[right])
78         if leftmax < rightmax:
79             # 管小不管大，可能存在更大但不在乎
80             res += leftmax - height[left]
81             left += 1
82         else:
83             res += rightmax - height[right]
84             right -= 1
85     return res
```