栈、队列、堆

In [ ]:

```python
# 232. 用栈实现队列
class MyQueue:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.stack1 = []  # 入栈
        self.stack2 = []  # 出栈

    def push(self, x: int) -> None:
        """
        Push element x to the back of queue.
        """
        self.stack1.append(x)

    def pop(self) -> int:
        """
        Removes the element from in front of queue and returns that element.
        """
        if self.stack2 == []:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop()


    def peek(self) -> int:
        """
        Get the front element.
        """
        if self.stack2 == []:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2[-1]


    def empty(self) -> bool:
        """
        Returns whether the queue is empty.
        """
        return self.stack2 == [] and self.stack1 == []

# 225. 用队列实现栈
from collections import deque
class MyStack:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.queue1 = deque()
        self.queue2 = deque()  # 辅助删除

    def push(self, x: int) -> None:
        """
        Push element x onto stack.
        """
        self.queue1.append(x)

    def pop(self) -> int:
        """
        Removes the element on top of the stack and returns that element.
        """
        size = len(self.queue1)
        for i in range(size-1):
            self.queue2.append(self.queue1.popleft())
        result = self.queue1.popleft()
        self.queue1,self.queue2 = self.queue2,self.queue1
        return result


```

```
    def top(self) -> int:
        """
        Get the top element.
        """
        return self.queue1[-1]

    def empty(self) -> bool:
        """
        Returns whether the stack is empty.
        """
        return len(self.queue1) == 0

# 20. 有效的括号
class Solution:
    def isValid(self, s: str) -> bool:
        def leftcode(c):
            if c == ']': return '['
            if c == ')': return '('
            if c == '}': return '{'
        stack = []
        for c in s:
            if c in '[{(':  # 输入是左括号
                stack.append(c)
            else:  # 输入是右括号
                if stack and leftcode(c) == stack[-1]:stack.pop()
                else: return False
        return stack == []

# 1047. 删除字符串中的所有相邻重复项
class Solution:
    def removeDuplicates(self, s: str) -> str:
        stack = []
        for c in s:
            if stack and c == stack[-1]: stack.pop()
            else: stack.append(c)
        return ''.join(stack)

# 150. 逆波兰表达式求值
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        result = []
        for token in tokens:
            if token not in '+-*/':
                result.append(token)
            else:
                temp1 = result.pop()
                temp2 = result.pop()
                res = eval(temp2+token+temp1)
                result.append(str(int(res)))
        return int(result[-1])

# 239. 滑动窗口最大值
class MyQueue():
    def __init__(self):
        self.queue = []
    def pop(self,x):
        if self.queue and x == self.queue[0]:
            self.queue.pop(0)
    def push(self,x):
        while self.queue and x > self.queue[-1]:
            self.queue.pop()
        self.queue.append(x)
    def front(self):
        return self.queue[0]
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        # 写一个单调栈，4个功能
        res = []
        que = MyQueue()
        for i in range(k): que.push(nums[i])
        res.append(que.front())
        for i in range(k,len(nums)):
```

```
            que.pop(nums[i-k])
            que.push(nums[i])
            res.append(que.front())
        return res

# 347. 前 K 个高频元素
import heapq
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        # dic = {i:nums.count(i) for i in set(nums)}
        # # 维护一个长为k最小堆
        # que = []
        # for key,frequent in dic.items():
        #     heapq.heappush(que,(frequent,key))
        #     if len(que) > k:
        #         heapq.heappop(que)
        # res = []
        # for _ in range(k):
        #     res.append(heappop(que)[1])
        # return res
        dic = {i:nums.count(i) for i in set(nums)}
        que = []
        for key,fre in dic.items():
            heappush(que,(fre,key))
            if len(que) > k:
                heappop(que)
        return [heappop(que)[1] for _ in range(k)]
```

单调栈

In [ ]:

```python
#### 739 每日温度
def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
    #单调栈
    n = len(temperatures)
    res = [0]*n # 存放结果
    stack = []  # 存放索引而并非元素值stack
    for i in range(n-1,-1,-1):
        while s and temperatures[stack[-1]] <= temperatures[i]:
            stack.pop()
        res[i] = 0 if not stack else stack[-1]-i
        stack.append(i)
    return res

#### 下一个更大的数
def nextGreaterElements(self, nums: List[int]) -> List[int]:
    n = len(nums)
    res = [0]*n
    stack = []
    for i in range(n-1,-1,-1):
        while stack and stack[-1] <= nums[i]:
            stack.pop()
        res[i] = -1 if not stack else stack[-1]
        stack.append(nums[i])
    return res

#### 503 循环数组，下一个更大的元素
def nextGreaterElements(self, nums: List[int]) -> List[int]:
    n = len(nums)
    res = [0]*n
    stack = []
    for i in range(2*n-1,-1,-1):
        while stack and stack[-1] <= nums[i%n]:
            stack.pop()
        res[i%n] = -1 if not stack else stack[-1]
        stack.append(nums[i%n])
    return res


#### 42 接雨水
def trap(self, height: List[int]) -> int:
    # 暴力求解（319/320）O(N^2)
    n = len(height)
    res = 0
    for i in range(1,n-1):
        leftmax,rightmax = 0,0
        for j in range(i,n):
            rightmax = max(rightmax,height[j])
        for j in range(i,-1,-1):
            leftmax = max(leftmax,height[j])
        res += min(leftmax,rightmax)-height[i]
    return res

    # 暴力+备忘录（dp全部算完）O(N)
    n = len(height)
    if n == 0: return 0
    res = 0
    leftmax = [0]*n
```

```
    rightmax = [0]*n
    leftmax[0] = height[0]
    rightmax[-1] = height[-1]
    for i in range(1,n):
        leftmax[i] = max(height[i],leftmax[i-1])
    for i in range(n-2,-1,-1):
        rightmax[i] = max(height[i],rightmax[i+1])
    for i in range(1,n-1):
        res += min(leftmax[i],rightmax[i])-height[i]
    return res

    # 双指针
    if not height: return 0
    n = len(height)
    left,right = 0,n-1
    res = 0
    leftmax,rightmax = height[0],height[-1]
    while left <= right:
        leftmax = max(leftmax,height[left])
        rightmax = max(rightmax,height[right])
        if leftmax < rightmax:
            # 管小不管大，可能存在更大但不在乎
            res += leftmax - height[left]
            left += 1
        else:
            res += rightmax - height[right]
            right -= 1
    return res
```