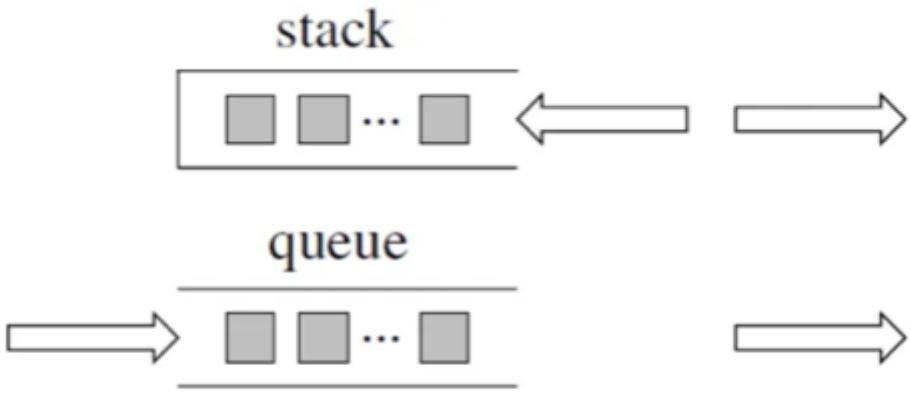
栈与队列



队列是先进先出，栈是先进后出

有效的括号

```python
class Solution:
    def isValid(self, s: str) -> bool:
        if len(s)%2 == 1:
            return False

        pair = {
            ']':'[',
            '}':'{',
            ')':'('
        }

        stk = []
        for c in s:
            if c in pair:
                if not stk or stk.pop()!=pair[c]:
                    return False
            else:
                stk.append(c)

        return not stk
```

删除字符串中的所有相邻重复项

```python
class Solution:
    def removeDuplicates(self, s: str) -> str:
        ls = ['1'] # 防止6行 ls[-1]报错

        for c in s:
            if c == ls[-1]:
                ls.pop()
            else:
                ls.append(c)
        return ''.join(ls[1:])
```

逆波兰表达式求值

```python
from operator import add, sub, mul

class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        mp = {'+': add, '-': sub, '*': mul, '/': lambda x, y: int(x / y)}
        stk = []
        for t in tokens:
            if t in mp:
                a,b=stk.pop(),stk.pop()
                stk.append(mp[t](b,a)) #注意a,b顺序
            else:
                stk.append(int(t))
        return stk.pop()

#关于运算，另一种处理方法
def evaluate(self, num1, num2, op):
    if op == "+":
        return num1 + num2
    elif op == "-":
        return num1 - num2
    elif op == "*":
        return num1 * num2
    elif op == "/":
        return int(num1 / float(num2))
```

前K个高频元素

```python
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        mp = {}
        for n in nums:
            mp[n]=mp.get(n,0)+1

        mp = dict(sorted(mp.items(), key=lambda x: x[1], reverse=True))
                                        #x[1]按值排序，x[0]按键
        ans=[]
        for i,ky in enumerate(mp.keys()):
            print(ky)
            if i==k:
                break
            ans.append(ky)
        return ans
```

滑动窗口最大值

```python
from collections import deque

class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        if not nums or len(nums) < 2:
            return nums

        que = deque()
        res = []

        for i, n in enumerate(nums):
            # 保证从大到小 如果前面数小则需要依次弹出，直至满足要求
            while que and nums[que[-1]] <= n:
                que.pop()

            # nums中元素的下标，加入队列中
            que.append(i)

            # 判断队首值是否有效
            if que[0] <= i - k:
                que.popleft()

            # 当窗口长度为k时 保存当前窗口中最大值
            if i + 1 >= k:
                res.append(nums[que[0]])
        return res
```