滑动窗口

In [1]:

```python
# 和为target的最短子串
def minSubArrayLen(target, nums):
    result = float('inf')
    i,Sum = 0,0
    for j in range(len(nums)):
        # 放大窗口
        Sum += nums[j]
        while Sum >= target:
            # 缩小窗口的条件
            # 下标从0开始，且该值要进窗口,j-i+1表示窗口大小
            result = min(result,j-i+1)
            Sum -= nums[i]
            i += 1
    return 0 if result > len(nums) else result


#水果成蓝，找只含两个原始的最长子串
def totalFruit(tree):
    if len(set(tree)) == 2 : return len(tree)  # 太长的情况
    fruits = []  # 类似Sum
    i,result = 0,0
    for j in range(len(tree)):
        # 放大窗口
        fruits.append(tree[j])
        while len(set(fruits)) > 2:
            # 不加等号，因为==2后还能继续再加
            # 缩小窗口的条件
            result = max(result,j-i)  # 不+1因为使>2的值不能进窗口
            fruits.pop(0)
            i += 1
    return max(result,len(fruits))

# 最小覆盖子串
def minWindow(s,t):
    need = {i:t.count(i) for i in set(t)}
    window = {}
    left,right = 0,0
    valid = 0
    start = 0
    length = float('inf')
    for right in range(len(s)):
        # 增加窗口
        c = s[right]
        # 更新窗口内容
        if c in need:
            # 需要这个字符
            # 加进window
            if c in window: window[c]+=1
            else: window[c] = 1
            if window[c] == need[c]: valid += 1  # 窗口中一个字符达到要求了
        while valid == len(need):  # len(need) == len(set(t))
            if right - left < length:
                # 最小覆盖的含义
                # 更新索引，记录位置
                start = left
                length = right - left + 1  # 这一个是需要的
            d = s[left]  # 要删除的字符
            left += 1
            if d in need:
                # 做相应的操作
                if window[d] == need[d]:
                    valid -= 1
                window[d] -= 1
    return '' if length == float('inf') else s[start:start+length]
```

字符串

In [67]:

```python
# 344 双指针实现字符翻转，不管是不是字符
def reverseString(s):
        """
        Do not return anything, modify s in-place instead.
        """
        left,right = 0,len(s)-1
        while left < right :
            # 加不加=都一样
            s[left],s[right] = s[right],s[left]
            left += 1
            right -= 1
        return s

# 541 前部分翻转
def reverseStr(self, s: str, k: int) -> str:
        # 一段一段的前半段进行反转
        s = list(s)
        def reverseString(s):
            left,right = 0,len(s)-1
            while left < right :
                s[left],s[right] = s[right],s[left]
                left += 1
                right -= 1
            return s
        for i in range(0,len(s),2*k):
            # 间隔为2k
            s[i:(i+k)] = reverseString(s[i:(i+k)])
            # 列表索引会越界，切边不会越界。[:len(s)+1]会取到[:len(s)]
        return ''.join(s)  # 连接字符串
        # return reduce(lambda a,b:a+b,s)

# 华为面试一，非字母符号位置不变
def A(s):
    s = list(s)
    left,right = 0,len(s)-1
    while left < right:
        if s[right].isalpha() and s[left].isalpha():
            s[right],s[left] = s[left],s[right]
            left += 1; right -= 1
        elif s[right].isalpha():
            left += 1
        else:
            right -= 1
    return ''.join(s)

# 151 翻转单词顺序，集合顺序改变  剑58，-> 包含 "" 和"  "
class Solution:
    def reverseWords(self, s: str) -> str:
        # 存在多余空格的情况,去除
        # s = list(s.strip())  # 去除前后空格
        l = self._strip(s)
        self.reverse_string(l,0,len(l)-1)
        self.all_words(l)
        return ''.join(l)
    # 指针实现
    def _strip(self,s):
        s = list(s)
        left,right = 0,len(s)-1
        while left <= right and s[left]==' ':
            left += 1
        while left <= right and s[right]==' ':
            right -= 1
        ##### 也可用数值中删除值的做法
        tmp = []
        for i in range(left,(right+1)):
            if s[i] == s[i-1] and s[i] == ' ':
                continue
            tmp.append(s[i])  #s[i] = s[j] i += 1
        return tmp  # list

    # 翻转nums的一部分
```

```python
    def reverse_string(self,nums,left,right):
        # 闭区间，344
        while left < right:
            nums[left],nums[right] = nums[right],nums[left]
            left += 1
            right -= 1
        return
    def all_words(self,nums):
        # 翻转整个字符串中的单词
        start,end = 0,0
        while start < len(nums):
            # 原地翻转，大小不变
            while end < len(nums) and nums[end] != ' ':
                end += 1
            self.reverse_string(nums,start,end-1)  # 闭区间
            start = end+1
            end += 1
        return

# 剑指Offer58-II.左旋转字符串
class Solution:
    def reverseLeftWords(self, s: str, n: int) -> str:
        s = list(s)
        self.reverse_string(s,0,n-1)  # 闭区间
        self.reverse_string(s,n,len(s)-1) # 闭区间
        self.reverse_string(s,0,len(s)-1) # 闭区间
        return ''.join(s)
    def reverse_string(self,nums,left,right):
        # 闭区间[]，344
        while left < right:
            nums[left],nums[right] = nums[right],nums[left]
            left += 1
            right -= 1
        return
```

KMP

In [11]:
```python
# 28 KMP
def getNext(pattern):
    # 含该字符的最长公共串长
    _next = [-1]*len(pattern)
    j = 0 # 前缀的最后一个字符
    _next[0] = j
    for i in range(1,len(pattern)):
        # i 后缀的最后一个字符
        while j > 0 and pattern[i] != pattern[j]:  # j在i后且不一定会走到最后
            j = _next[j-1]  # 前后缀不相同时向前回退
        if pattern[i] == pattern[j]:
            j += 1
        _next[i] = j  #将j（前缀的长度）赋给next[i]
    return _next
def strStr(chang,duan):
    if len(duan) == 0: return 0
    _next  = getNext(duan)
    j = 0
    for i in range(len(chang)):
        while j >0 and chang[i] != duan[j]:
            j = _next[j-1]
        if chang[i] == duan[j]: j += 1
        if j == len(duan): return i-len(duan)+1
    return -1

# 459 重复子字符串
def repeatedSubstringPattern(s):
    def getNext(pattern):
        # 含该字符的最长公共串长，构建next数组
        _next = [-1]*len(pattern)
        j = 0 # 前缀的最后一个字符
        _next[0] = j
        for i in range(1,len(pattern)):
            # i 后缀的最后一个字符
            while j > 0 and pattern[i] != pattern[j]:
                j = _next[j-1]  # 前后缀不相同时向前回退，回退到什么位置
            if pattern[i] == pattern[j]:
                j += 1
            _next[i] = j  #将j（前缀的长度）赋给next[i]
        return _next
    # 对next数组做简单判断
    _next = getNext(s)
    if _next[-1] == 0: return False
    if len(s) % (len(s)-_next[-1]) == 0: return True  # 成立时推出来的
    print(len(s) % (len(s)-_next[-1]))
    return False
```

In [12]:
```python
getNext('abcefgabc')
```

Out[12]: [0, 0, 0, 0, 0, 0, 1, 2, 3]

In [13]:
```python
repeatedSubstringPattern('abcefgabc')
```

```
3
```

Out[13]: False