

## 滑动窗口

```
In [1]: 1 # 和为target的最短子串
2 def minSubArrayLen(target, nums):
3     result = float('inf')
4     i, Sum = 0, 0
5     for j in range(len(nums)):
6         # 放大窗口
7         Sum += nums[j]
8         while Sum >= target:
9             # 缩小窗口的条件
10            # 下标从0开始, 且该值要进窗口, j-i+1表示窗口大小
11            result = min(result, j-i+1)
12            Sum -= nums[i]
13            i += 1
14    return 0 if result > len(nums) else result
15
16
17 #水果成篮, 找只含两个原始的最长子串
18 def totalFruit(tree):
19     if len(set(tree)) == 2: return len(tree) # 太长的情况
20     fruits = [] # 类似Sum
21     i, result = 0, 0
22     for j in range(len(tree)):
23         # 放大窗口
24         fruits.append(tree[j])
25         while len(set(fruits)) > 2:
26             # 不加等号, 因为==2后还能继续再加
27             # 缩小窗口的条件
28             result = max(result, j-i) # 不+1因为使>2的值不能进窗口
29             fruits.pop(0)
30             i += 1
31     return max(result, len(fruits))
32
33 # 最小覆盖子串
34 def minWindow(s, t):
35     need = {i:t.count(i) for i in set(t)}
36     window = {}
37     left, right = 0, 0
38     valid = 0
39     start = 0
40     length = float('inf')
41     for right in range(len(s)):
42         # 增加窗口
43         c = s[right]
44         # 更新窗口内容
45         if c in need:
46             # 需要这个字符
47             # 加进window
48             if c in window: window[c] += 1
49             else: window[c] = 1
50             if window[c] == need[c]: valid += 1 # 窗口中一个字符达到要求了
51     while valid == len(need): # len(need) == len(set(t))
52         if right - left < length:
53             # 最小覆盖的含义
54             # 更新索引, 记录位置
55             start = left
56             length = right - left + 1 # 这一个是要的
57         d = s[left] # 要删除的字符
58         left += 1
59         if d in need:
60             # 做相应的操作
61             if window[d] == need[d]:
62                 valid -= 1
63             window[d] -= 1
64     return '' if length == float('inf') else s[start:start+length]
```

## 字符串

```
In [67]: 1 # 344 双指针实现字符翻转, 不管是不是字符
2 def reverseString(s):
3     """
4     Do not return anything, modify s in-place instead.
5     """
6     left, right = 0, len(s)-1
7     while left < right :
8         # 加不加都一样
9         s[left], s[right] = s[right], s[left]
10        left += 1
11        right -= 1
12    return s
13
14 # 541 前部分翻转
15 def reverseStr(self, s: str, k: int) -> str:
16     # 一段一段的前半段进行反转
17     s = list(s)
18     def reverseString(s):
19         left, right = 0, len(s)-1
20         while left < right :
21             s[left], s[right] = s[right], s[left]
22             left += 1
23             right -= 1
24         return s
25     for i in range(0, len(s), 2*k):
26         # 间隔为2k
27         s[i:(i+k)] = reverseString(s[i:(i+k)])
28         # 列表索引会越界, 切边不会越界。[:len(s)+1]会取到[:len(s)]
29     return ''.join(s) # 连接字符串
30     # return reduce(lambda a, b: a+b, s)
31
32 # 华为面试一, 非字母符号位置不变
33 def A(s):
34     s = list(s)
35     left, right = 0, len(s)-1
36     while left < right:
37         if s[right].isalpha() and s[left].isalpha():
38             s[right], s[left] = s[left], s[right]
39             left += 1; right -= 1
40         elif s[right].isalpha():
41             left += 1
42         else:
43             right -= 1
44     return ''.join(s)
45
46 # 151 翻转单词顺序, 集合顺序改变 剑58, -> 包含 "" 和 " "
47 class Solution:
48     def reverseWords(self, s: str) -> str:
49         # 存在多余空格的情况, 去除
50         # s = list(s.strip()) # 去除前后空格
51         l = self._strip(s)
52         self.reverse_string(l, 0, len(l)-1)
53         self.all_words(l)
54         return ''.join(l)
55     # 指针实现
56     def _strip(self, s):
57         s = list(s)
58         left, right = 0, len(s)-1
59         while left <= right and s[left] == ' ':
60             left += 1
61         while left <= right and s[right] == ' ':
62             right -= 1
63         ##### 也可用数值中删除值的做法
64         tmp = []
65         for i in range(left, (right+1)):
66             if s[i] == s[i-1] and s[i] == ' ':
67                 continue
68             tmp.append(s[i]) #s[i] = s[j] i += 1
69         return tmp # list
70
71 # 翻转nums的一部分
```

```

72     def reverse_string(self, nums, left, right):
73         # 闭区间, 344
74         while left < right:
75             nums[left], nums[right] = nums[right], nums[left]
76             left += 1
77             right -= 1
78         return
79     def all_words(self, nums):
80         # 翻转整个字符串中的单词
81         start, end = 0, 0
82         while start < len(nums):
83             # 原地翻转, 大小不变
84             while end < len(nums) and nums[end] != ' ':
85                 end += 1
86             self.reverse_string(nums, start, end-1) # 闭区间
87             start = end+1
88             end += 1
89         return
90
91 # 剑指Offer58-II. 左旋转字符串
92 class Solution:
93     def reverseLeftWords(self, s: str, n: int) -> str:
94         s = list(s)
95         self.reverse_string(s, 0, n-1) # 闭区间
96         self.reverse_string(s, n, len(s)-1) # 闭区间
97         self.reverse_string(s, 0, len(s)-1) # 闭区间
98         return ''.join(s)
99     def reverse_string(self, nums, left, right):
100         # 闭区间[], 344
101         while left < right:
102             nums[left], nums[right] = nums[right], nums[left]
103             left += 1
104             right -= 1
105         return

```

## KMP

```
In [11]: 1 # 28 KMP
2 def getNext(pattern):
3     # 含该字符的最长公共串长
4     _next = [-1]*len(pattern)
5     j = 0 # 前缀的最后一个字符
6     _next[0] = j
7     for i in range(1, len(pattern)):
8         # i 后缀的最后一个字符
9         while j > 0 and pattern[i] != pattern[j]: # j在i后且不一定走到最后
10             j = _next[j-1] # 前后缀不相同时向前回退
11         if pattern[i] == pattern[j]:
12             j += 1
13         _next[i] = j #将j（前缀的长度）赋给next[i]
14     return _next
15 def strStr(chang, duan):
16     if len(duan) == 0: return 0
17     _next = getNext(duan)
18     j = 0
19     for i in range(len(chang)):
20         while j > 0 and chang[i] != duan[j]:
21             j = _next[j-1]
22         if chang[i] == duan[j]: j += 1
23         if j == len(duan): return i-len(duan)+1
24     return -1
25
26 # 459 重复子字符串
27 def repeatedSubstringPattern(s):
28     def getNext(pattern):
29         # 含该字符的最长公共串长，构建next数组
30         _next = [-1]*len(pattern)
31         j = 0 # 前缀的最后一个字符
32         _next[0] = j
33         for i in range(1, len(pattern)):
34             # i 后缀的最后一个字符
35             while j > 0 and pattern[i] != pattern[j]:
36                 j = _next[j-1] # 前后缀不相同时向前回退，回退到什么位置
37             if pattern[i] == pattern[j]:
38                 j += 1
39             _next[i] = j #将j（前缀的长度）赋给next[i]
40         return _next
41     # 对next数组做简单判断
42     _next = getNext(s)
43     if _next[-1] == 0: return False
44     if len(s) % (len(s)-_next[-1]) == 0: return True # 成立时推出来的
45     print(len(s) % (len(s)-_next[-1]))
46     return False
```

```
In [12]: 1 getNext('abcefgabc')
```

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

```
In [13]: 1 repeatedSubstringPattern('abcefgabc')
```

```
3
```

```
Out[13]: False
```