知识总结

- 字典具有相同的键和值，即使键的顺序不同，也将返回True

```
dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'b': 2, 'c': 3, 'a': 1}

print(dict1 == dict2)  # True
```

- 字典排序

```
#x[0]按key降序
mp = dict(sorted(mp.items(), key=lambda x: x[0], reverse=True))
#x[1]按val降序
mp = dict(sorted(mp.items(), key=lambda x: x[1], reverse=True))
```

-

```
ord('a') = 97
ord('A') = 65
```

-

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}

# 并集（Union）
set1.union(set2)
set1 | set2

# 交集（Intersection）
set1.intersection(set2)
set1 & set2

# 补集（Complement）
set1.difference(set2)
set1 - set2
```

-

```
float(inf)
```

-

```
" ".join(reversed(s.split()))
```

-

```
'/': lambda x, y: int(x / y)
```

-

```
#counts是list，mp的key不能是list，要转换tuple
mp[tuple(counts)].append(st)
```

数组

有序数组

二分查找

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        l, r = 0, len(nums)-1
        while l <= r:
            m = l+(r-l)//2
            t = nums[m]  #不要写错成nums(m)
            if t == target:
                return m
            elif t > target:
                r = m - 1
            else:
                l = m + 1
        return -1
```

搜索插入位置

```python
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        l, r = 0, len(nums)-1
        while l <= r:
            m = l+(r-l)//2
            t = nums[m]
            if t >= target:  #区别
                r = m - 1
            else:
                l = m + 1
        return l
```

在排序数组中查找元素的第一个和最后一个位置

```python
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        def sb(nums, target): #与上一题代码相同
            l, r = 0, len(nums)-1
            while l <= r:
                m = l+(r-l)//2
                t = nums[m]
                if t >= target:
                    r = m - 1
                else:
                    l = m + 1
```

```
            return l

        l = sb(nums, target)
        r = sb(nums, target+1) - 1

        if l == len(nums) or nums[l] != target: #条件顺序不能写反
            return [-1, -1]
        return [l, r]
```

移除元素

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        f = s = 0
        ln = len(nums)

        while f < ln:
            if nums[f] != val:    #和下一题的区别
                nums[s] = nums[f]
                s += 1
            f += 1
        return s #s代表移除后的数组长度
```

删除有序数组中的重复项

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        f=s=1 #和上一题的区别
        ln = len(nums)

        while f < ln:
            if nums[f] != nums[f-1]:  #和上一题的区别
                nums[s] = nums[f]
                s += 1
            f += 1
        return s #s代表去重后的数组长度
```

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        for i, x in enumerate(sorted(set(nums))):
            nums[i] = x
        return i + 1
```

移动零

```python
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        f = s = 0
        ln = len(nums)

        while f < ln:
            if nums[f] != 0:  #和上一题的区别
                nums[s] = nums[f]
                s += 1
            f += 1

        for i in range(ln-s):    #最后补0
            nums[-1-i] = 0

```

比较含退格的字符串

题目明确，注意：如果对空文本输入退格字符，文本继续为空。

```python
class Solution:
    def backspaceCompare(self, s: str, t: str) -> bool:
        def build(s):
            ret = []
            for ch in s:
                if ch != "#":
                    ret.append(ch)
                elif ret:    #注意！！！！
                    ret.pop()
            return ret

        return build(s) == build(t)
```

长度最小的子数组

长度最小的子数组

示例 1：

> **输入：** target = 7, nums = [2,3,1,2,4,3]
> **输出：** 2
> **解释：** 子数组 [4,3] 是该条件下的长度最小的子数组。

```python
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        f=s=0
        ln = len(nums)

```

```
        ret = ln + 1
        total = 0

        while f < ln:
            total += nums[f]
            while total >= target:
                ret = min(ret, f-s+1)
                total -= nums[s]
                s += 1
            f += 1

        return 0 if ret == ln+1 else ret
```

滑窗模板

```
# 最小滑窗模板
while j < len(nums):
    判断[i, j]是否满足条件
    while 满足条件:
        不断更新结果  #(注意在while内更新！)
        i += 1 （最大程度的压缩i，使得滑窗尽可能的小）
    j += 1

# 最大滑窗模板
while j < len(nums):
    判断[i, j]是否满足条件
    while 不满足条件:
        i += 1 （最保守的压缩i，一旦满足条件了就退出压缩i的过程，使得滑窗尽可能的大）
    不断更新结果  #（注意在while外更新！）
    j += 1
```

最小覆盖子串

```
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        lns,lnt = len(s),len(t)
        if lns < lnt:return ""

        mp={}
        for c in t:
            mp[c] = mp.get(c,0)+1
        lg = len(mp)

        ans = ""
        l=r=0

```

```
        while r<lns:
            while r<lns and lg:
                if s[r] in mp:
                    mp[s[r]] -= 1
                    if mp[s[r]] == 0:lg-=1
                r += 1
            if r == lns and lg:break

            while l<r and not lg:
                if s[l] in mp:
                    mp[s[l]]+=1
                    if mp[s[l]] > 0:lg+=1

                l += 1
            if not ans or len(ans)>r-l+1: #or
                ans = s[l-1:r]
        return ans
```

水果成篮

示例 3：

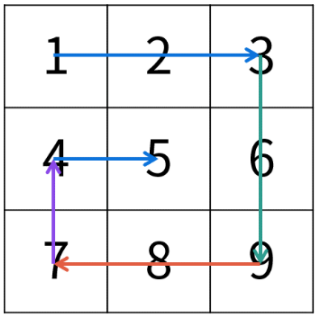
> **输入：** fruits = [1,**2,3,2,2**]
> **输出：** 4
> **解释：** 可以采摘 [2,3,2,2] 这四棵树。
> 如果从第一棵树开始采摘，则只能采摘 [1,2] 这两棵树。

```
class Solution:
    def totalFruit(self, fruits: List[int]) -> int:
        mp = {}
        lg=0
        ans =0
        l=r=0

        while r<len(fruits):
            mp[fruits[r]]=mp.get(fruits[r],0)+1
            if mp[fruits[r]]==1:
                lg+=1

            while lg>2:
                mp[fruits[l]]-=1
                if mp[fruits[l]]==0:
                    lg-=1
                l+=1
            ans = max(ans, r-l+1)
            r+=1
        return ans
```

螺旋矩阵



螺旋遍历二维数组

```python
class Solution:
    def spiralArray(self, matrix: List[List[int]]) -> List[int]:
        if not matrix: return []

        n = len(matrix)
        m = len(matrix[0])
        top, bottom = 0, n - 1
        left, right = 0, m - 1

        res = []
        while True:
            # 左 -> 右
            for i in range(left, right + 1):
                res.append(matrix[top][i])
            top += 1
            if top > bottom: break

            # 上 -> 下
            for i in range(top, bottom + 1):
                res.append(matrix[i][right])
            right -= 1
            if left > right: break

            # 右 -> 左
            for i in range(right, left - 1, -1):
                res.append(matrix[bottom][i])
            bottom -= 1
            if top > bottom: break

            # 下 -> 上
            for i in range(bottom, top - 1, -1):
                res.append(matrix[i][left])
            left += 1
            if left > right: break
        return res
```

螺旋矩阵 II

```python
class Solution:
    def generateMatrix(self, n: int) -> [[int]]:
        l=t=0
        r=b=n-1

        mat = [[0 for _ in range(n)] for _ in range(n)]
        num, tar = 1, n * n

        while num <= tar:
            for i in range(l, r + 1): # left to right
                mat[t][i] = num
                num += 1
            t += 1
            for i in range(t, b + 1): # top to bottom
                mat[i][r] = num
                num += 1
            r -= 1
            for i in range(r, l - 1, -1): # right to left
                mat[b][i] = num
                num += 1
            b -= 1
            for i in range(b, t - 1, -1): # bottom to top
                mat[i][l] = num
                num += 1
            l += 1
        return mat
```