Array/3SumWithMultiplicity.py

```
"""
Given an integer array A, and an integer target, return the number of tuples i,
j, k  such that i < j < k and A[i] + A[j] + A[k] == target.
As the answer can be very large, return it modulo 10^9 + 7.
Example 1:
Input: A = [1,1,2,2,3,3,4,4,5,5], target = 8
Output: 20
Explanation:
Enumerating by the values (A[i], A[j], A[k]):
(1, 2, 5) occurs 8 times;
(1, 3, 4) occurs 8 times;
(2, 2, 4) occurs 2 times;
(2, 3, 3) occurs 2 times.
Example 2:
Input: A = [1,1,2,2,2,2], target = 5
Output: 12
Explanation:
A[i] = 1, A[j] = A[k] = 2 occurs 12 times:
We choose one 1 from [1,1] in 2 ways,
and two 2s from [2,2,2,2] in 6 ways.
Note:
3 <= A.length <= 3000
0 <= A[i] <= 100
0 <= target <= 300
思路：
首先排序，然后按照正常的3sum。
这样做有个问题：
重复的太多会超级耗时，我的做法是重新生成列表，若超过三个，则按3个加入。
继续之前的思路所要生成的重复的次数：
[1,1,2,2,3,3,4,4,5,5]
1 2 5
1出现了两次，2出现了两次，5出现了两次，那么总次数为 2*2*2 = 8
(2, 2, 4)
2出现了两次，4出现了两次，不过 2 在这个子问题中也出现了两次，所以总数应为 2的总数-1的阶加 * 4
出现的总数。
sum(range(2)) * 2
如果是
0 0 0
0
这样子问题是
0 0 0，出现三次的情况。
若原列表中出现了 3 次，那么只有 1种情况。
                  4 次，则是 4 种。
                  5 次，则是 10 种。
                  6 次，则是 20 种。
这个次数其实是 3 次则是 sum(range(2)) + sum(range(1))
               4 次则是 sum(range(3)) + sum(range(2)) + sum(range(1))
               ...
按照这个思路我的做法是首先生成最大次数的阶加结果。
可以 passed. 320ms。 运行测试的话每次都要 生成阶加列表，如果可以只生成一次到是可以减少很大运行
时间，可能还有其他方法？
测试地址：
```

```
https://leetcode.com/problems/3sum-with-multiplicity/description/
"""
class Solution(object):
    def threeSumMulti(self, A, target):
        """
        :type A: List[int]
        :type target: int
        :rtype: int
        """
        mod = 10**9 + 7
        def get_multiple(time, all_times):
            if time==1:
                return all_times
            elif time==2:
                return sum(range(all_times))
            elif time==3:
                return sum(plus[:all_times])
        x = {}
        for j in set(A):
            x[j] = A.count(j)
        maxes = max(x, key=lambda t: x[t])
        plus = [sum(range(i)) for i in range(x[maxes])]
        A = []
        for i in sorted(x.keys()):
            if x[i] > 3:
                A.extend([i]*3)
            else:
                A.extend([i]*x[i])
        result = 0
        length = len(A) - 1
        sets = set()
        for i in range(length):
            t = target - A[i]
            start = i+1
            end = length
            while start < end:
                if A[start] + A[end] == t:
                    # pass
                    _ = [A[i], A[start], A[end]]
                    y = {e:_.count(e) for e in set(_)}
                    _ = "{}{}{}".format(*_)
                    if _ in sets:
                        start += 1
                        continue
                    c = 1
                    for g in y:
                        c *= get_multiple(y[g], x[g])
                    result += c
                    sets.add(_)
                    start += 1
                if A[start] + A[end] > t:
                    end -= 1
                else:
                    start += 1
        return result % mod
```

Array/AddTowNumbersII.py

```
"""
You are given two non-empty linked lists representing two non-negative integers.
The most significant digit comes first and each of their nodes contain a single
digit. Add the two numbers and return it as a linked list.
You may assume the two numbers do not contain any leading zero, except the number
0 itself.
Follow up:
What if you cannot modify the input lists? In other words, reversing the lists is
not allowed.
Example:
Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 8 -> 0 -> 7
这次给定的正序排列的，而且不允许翻转链表。
思路：
先遍历一遍，为短的一个链表填充0。
(7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)
               ↓
(7 -> 2 -> 4 -> 3) + (0 -> 5 -> 6 -> 4)
之后用递归：
def x(l1, l2):
    # 退出的条件是 l1 和 l2 均为 None.
    # 当然因为事先做了长度相同的处理这里判断一个即可。
    # 返回的是 (rest, ListNode)
    if l1 is None:
        return (0, None)
    nextValue = x(l1.next, l2.next)
    get_value = getValue(l1, l2, nextValue[0])
    myNode = ListNode(get_value[0])
    myNode.next = nextValue[1]
    return (get_value[1], myNode)
之后在最外层做一次rest判断。
Ok，一遍过，beat 89%.
还有一种方法是不断 * 10，相加后不断 % 10， // 10。
效率都是一样的。
测试地址：
https://leetcode.com/problems/add-two-numbers-ii/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        l1, l2 = self.getEqualNodes(l1, l2)
        # return l1, l2
        result = self._addTwoNumbers(l1, l2)
        if result[0]:
            node = ListNode(result[0])
```

```
            node.next = result[1]
            return node
        return result[1]
    def _addTwoNumbers(self, l1, l2):
        if l1 is None:
            return (0, None)
        nextValue = self._addTwoNumbers(l1.next, l2.next)
        get_value = self.getRest(l1, l2, nextValue[0])
        myNode = ListNode(get_value[0])
        myNode.next = nextValue[1]
        return (get_value[1], myNode)
        # get_value = self.getRest()
    def getEqualNodes(self, l1, l2):
        # Get equal length.
        b_l1 = l1
        b_l2 = l2
        while b_l1 is not None and b_l2 is not None:
            b_l1 = b_l1.next
            b_l2 = b_l2.next
        if b_l1:
            # l1 is longer than l2
            t_lx = b_l1
            fix_lx = l2
            raw_lx = l1
        else:
            t_lx = b_l2
            fix_lx = l1
            raw_lx = l2
        if t_lx:
            root = ListNode(0)
            b_root = root
            t_lx = t_lx.next
            while t_lx:
                node = ListNode(0)
                root.next = node
                root = node
                t_lx = t_lx.next
            root.next = fix_lx
            return (raw_lx, b_root)
        return (l1, l2)
    def getRest(self, l1, l2, rest=0):
        # return (val, rest)
        # 9+8 (7, 1)
        l1_val = l1.val if l1 else 0
        l2_val = l2.val if l2 else 0
        return ((l1_val + l2_val + rest) % 10, (l1_val + l2_val + rest) // 10)
```

Array/AddTwoNumbers.py

```
"""
You are given two non-empty linked lists representing two non-negative integers.
The digits are stored in reverse order and each of their nodes contain a single
digit. Add the two numbers and return it as a linked list.
You may assume the two numbers do not contain any leading zero, except the number
0 itself.
Example:
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
Explanation: 342 + 465 = 807.
给两个非空的链表，每个节点包含一个整数。数字是以倒序排列的，现在输出两个链表相加得出的新链表。
长度问题：
1. 同长度不需要考虑。
2. 不同长度下，正序意味着
 3421 + 465 = 3421
               465
这种情况最好的方法应该是从后向前，但此题目中给出的就是从后向前，所以也不必考虑这个，
直接给0即可。
思路：
头到尾，对于每一个位来说，最大是 9+9，进位最大是1.
Ok，一遍过，O(n)。
beat 67%.
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        get_value = self.getRest(l1, l2)
        root = ListNode(get_value[0])
        rest_value = get_value[1]
        l1 = l1.next
        l2 = l2.next
        backup_root = root
        while l1 is not None or l2 is not None:
            get_value = self.getRest(l1, l2, rest_value)
            new_node = ListNode(get_value[0])
            rest_value = get_value[1]
            root.next = new_node
            root = new_node
            l1 = l1.next if l1 else None
            l2 = l2.next if l2 else None
        if rest_value:
            root.next = ListNode(rest_value)
        return backup_root
    def getRest(self, l1, l2, rest=0):
        # return (val, rest)
```

```
        # 9+8 (7, 1)
        l1_val = l1.val if l1 else 0
        l2_val = l2.val if l2 else 0
        return ((l1_val + l2_val + rest) % 10, (l1_val + l2_val + rest) // 10)
```

Array/BestTimeToBuyAndSellStockI_II.py

```
"""
I
Say you have an array for which the ith element is the price of a given stock on
day i.
If you were only permitted to complete at most one transaction (i.e., buy one and
sell one share of the stock), design an algorithm to find the maximum profit.
Note that you cannot sell a stock before you buy one.
Example 1:
Input: [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1
= 5.
             Not 7-1 = 6, as selling price needs to be larger than buying price.
Example 2:
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
II
Say you have an array for which the ith element is the price of a given stock on
day i.
Design an algorithm to find the maximum profit. You may complete as many
transactions as you like (i.e., buy one and sell one share of the stock multiple
times).
Note: You may not engage in multiple transactions at the same time (i.e., you
must sell the stock before you buy again).
Example 1:
Input: [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1
= 4.
             Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit
= 6-3 = 3.
Example 2:
Input: [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1
= 4.
             Note that you cannot buy on day 1, buy on day 2 and sell them later,
as you are
             engaging multiple transactions at the same time. You must sell
before buying again.
Example 3:
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
I 比较好理解，找到此前中最小的一个与当前相减，最后取最大的一个即可。
II 有点绕：
1 2 3 这样一个递增段，在 1 买入， 3卖出时是最大的。不过 1 买入，2卖出 2在买入，3卖出 是同样的
效果。
如果 1 - 3 中出现一个比 1 小的，那么替换即可，若出现大的用大的减去即可。
I测试地址：
https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/
II测试地址：
https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/description/
```

```
"""
# I
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if not prices:
            return 0
        mins = prices[0]
        maxes = 0
        for i in range(1, len(prices)):
            mins = min(prices[i], mins)
            maxes = max(prices[i]-mins, maxes)
        return maxes
# II
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        sums = 0
        for i in range(len(prices)-1):
            if prices[i+1] > prices[i]:
                sums += prices[i+1] - prices[i]
        return sums
```

Array/BinarySubarraysWithSum.py

```
"""
In an array A of 0s and 1s, how many non-empty subarrays have sum S?
Example 1:
Input: A = [1,0,1,0,1], S = 2
Output: 4
Explanation:
The 4 subarrays are bolded below:
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
Note:
A.length <= 30000
0 <= S <= A.length
A[i] is either 0 or 1.
pre result + hash 有一个思路一模一样，代码也一模一样的。
测试地址：
https://leetcode.com/contest/weekly-contest-108/problems/binary-subarrays-with-sum/
"""
class Solution(object):
    def numSubarraysWithSum(self, A, S):
        """
        :type A: List[int]
        :type S: int
        :rtype: int
        """
        dicts = {0:1}
        pre = 0
        result = 0
        for i in A:
            pre += i
            result += dicts.get(pre-S, 0)
            dicts[pre] = dicts.get(pre, 0) + 1
        return result
```

Array/ContainerWithMostWater.py

```
"""
Given n non-negative integers a1, a2, ..., an , where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.
Note: You may not slant the container and n is at least 2.
The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.
Example:
Input: [1,8,6,2,5,4,8,3,7]
Output: 49
给一组数据，返回两组边界中可盛放水最多的一组。
图数据看链接：
https://leetcode.com/problems/container-with-most-water/description/
思路：
1. 一开始是个Dp, O(n²)。当然是TLE...
Dp时的主要难点在于：
无法把当前点作为子问题答案，只能是记录从之前的所有点到此点最大的一组作为子问题。
分析
盛放水有两个决定因素：一个是点到点的距离，另一个是两点中最短的那个的高度。
一开始可能会想的很简单：
记录出最高最远的一个点不就是了？
答案是错误的。
例子中所选的点是：
 0 1 2 3 4 5 6 7 8
[1,8,6,2,5,4,8,3,7]
   ↑               ↑
若是修改一下，把第一个从 1 变为 7。那么应该选 0 和 8 索引处的数。
 0 1 2 3 4 5 6 7 8
[7,8,6,2,5,4,8,3,7]
 ↑                 ↑
当记录 7,0 和 8,1 相遇，是记录 7,0 好还是记录 8,1 好呀？
肯定会说当然是记录 7,0 它能提供 7 个呢，8从高度上只多了 1 而已。
这样在部分情况下当然没问题。
还有一种情况：
[7, 15, 13]
这种情况15 比 7 多了 8 点，是否应该记录 15,1 而不是 7,0 呢？
如果记录了 15,1  在 13 处所记录的最大值应为 13 * (2-1) = 13
但是 用 7,0 却可以有 7 * (2-0) = 14 呢。
这种情况下 15 这个点应该用 7,0
但稍加一位：
[7, 15, 13, 12]
这个时候，15 这个点用 15,1 又是最合适的。
所以明朗了，
当前点用哪个取决于它后面的数。
在判断到 15 这个点的时候，如果后面也有一个数，它与 7 的差值*与 15 的距离可以抵消掉 7 的话就可以了。
但是这个写法写起来不光麻烦...而且从效率上好像并无改进，甚至更差，因为不光要把剩下的所有元素迭代一遍，
有时还要判断之前的一些。
2.
通过上面的分析，这个算法肯定不能只从一边想办法。
由此，想到了 Two sum 的一种O(n) 算法：
先排序，
```

有两个指针从两端开始走，若它们相加大于目标，那么就把右边的指针向前挪一位，反之左边的挪一位。
同样的思路用在这。
用两个指针从两端走，记录下当前的最大值，然后让其中较小的一边移动一位。
为什么移动较小的一个呢：
先来看如果移动较大的一位：
将较大的一位记为 X.
  较小的记为 Y.
那么剩下的数对于Y的那个来说有三种情况：
1. 比X大。 这种情况肯定是不如之前的可放的水多，高度不变的情况下距离变短了。
2. 与X相等。同上。
3. 比X要小。
   如果这个数比X要小，那么对应的也有三种情况：
   1. 比 Y 要小，如果比 Y 小，那么距离和高度同时不如 X, Y。
   2. 与 Y 相等，距离不如 X, Y。
   3. 比 Y 要大，距离不如 X, Y。
所以如果移动较大的一位，剩下的结果只能是要小。没有比它们更大的情况。
所以移动较小的一位，可以找出潜在的上面分析到的，
像 [7, 15, 13, 12]
是否有可能有 15 - 12 这种情况。

最后，再写的时候又发现一个问题：
如果两个数相同的要怎么办呢？
最后得出的结论是：
凉拌！

拿错剧本了。
应该是 走哪一边无所谓。
因为只有以下几种情况：
1. 比它们小的，这种情况全部都会小，静静的等着迭代结束就好了。
2. 有一个比它们大的，这种情况同上，分析过了。
3. 有多个比它们大的，只有这种情况才有可能比这一对相同的大。
   而这种情况，先走哪一个都不会对结果有影响，如果有这种情况先走的那个一定会在这个点上等着后走的追上来。
OK:
思路清晰了：
1. 头尾两个指针。
2. 小的要先走。
3. 同样的无所谓。
4. 记录每一步 min(X, Y) * distance.
5. 在头尾相遇时结束，也就是 O(n) 时间复杂度与 O(1) 空间复杂度。
测试链接：
https://leetcode.com/problems/container-with-most-water/description/
beat 95% 40ms。

哇塞，又一个自己想到的好方法，测试通过后在 Discuss 里找到同样的思路的算法。
好像，我还是有点东西的喔。哎嘿嘿。
"""

```python
class Solution(object):
    def maxArea(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        l, r = 0, len(height)-1
        currentMax = 0
        while l != r:
            currentMax = max(min(height[r], height[l]) * (r-l), currentMax)
            if height[r] > height[l]:
```

```
                l += 1
            else:
                r -= 1
        return currentMax
```

Array/CountOfSmallerNumbersAfterSelf.py

```
"""
You are given an integer array nums and you have to return a new counts array.
The counts array has the property where counts[i] is the number of smaller
elements to the right of nums[i].
Example:
Input: [5,2,6,1]
Output: [2,1,1,0]
Explanation:
To the right of 5 there are 2 smaller elements (2 and 1).
To the right of 2 there is only 1 smaller element (1).
To the right of 6 there is 1 smaller element (1).
To the right of 1 there is 0 smaller element.
思路:
二分。
瓶颈依然在于插入列表中的时候需要的时间复杂度为 O(n)。
beat
82%
测试地址:
https://leetcode.com/problems/count-of-smaller-numbers-after-self/description/
"""
import bisect
class Solution(object):
    def countSmaller(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        x = []
        result = []
        for i in nums[::-1]:
            result.append(bisect.bisect_left(x, i))
            bisect.insort(x,i)
        return result[::-1]
```

Array/FairCandySwap.py

```
"""
Alice and Bob have candy bars of different sizes: A[i] is the size of the i-th
bar of candy that Alice has, and B[j] is the size of the j-th bar of candy that
Bob has.
Since they are friends, they would like to exchange one candy bar each so that
after the exchange, they both have the same total amount of candy.  (The total
amount of candy a person has is the sum of the sizes of candy bars they have.)
Return an integer array ans where ans[0] is the size of the candy bar that Alice
must exchange, and ans[1] is the size of the candy bar that Bob must exchange.
If there are multiple answers, you may return any one of them.  It is guaranteed
an answer exists.
Example 1:
Input: A = [1,1], B = [2,2]
Output: [1,2]
Example 2:
Input: A = [1,2], B = [2,3]
Output: [1,2]
Example 3:
Input: A = [2], B = [1,3]
Output: [2,3]
Example 4:
Input: A = [1,2,5], B = [2,4]
Output: [5,4]
Note:
1 <= A.length <= 10000
1 <= B.length <= 10000
1 <= A[i] <= 100000
1 <= B[i] <= 100000
It is guaranteed that Alice and Bob have different total amounts of candy.
It is guaranteed there exists an answer.
开胃菜。
有两个好盆友，要交换糖果使得他们两个的糖果一样。
输出的结果也是交换的糖果数量而不是下标，所以非常容易写。
设爱丽丝给 x 颗糖，得到 y颗。那鲍勃就给 y 得到 x。
最终是
1? - x + y = 2? - y + x
2x - 2y = 1? - 2?
x - y = (1? - 2?) / 2
1?即 爱丽丝的初始糖，
2?即 鲍勃的初始糖。
那么首先把 爱丽丝和鲍勃的糖加起来 / 2，然后迭代 爱丽丝的糖，若迭代的这个糖 - 前面计算出来的糖鲍勃有的话返回即可。
要写的话首先把鲍勃的糖哈希一下，这样查找的时间复杂度为 O(1)。
测试地址：
https://leetcode.com/contest/weekly-contest-98/problems/fair-candy-swap/
beat:
100% 44 ms
"""
class Solution(object):
    def fairCandySwap(self, A, B):
        """
        :type A: List[int]
        :type B: List[int]
        :rtype: List[int]
```

```
        """
        x = (sum(A) - sum(B)) // 2
        b = set(B)
        for i in A:
            if i - x in b:
                return (i, i-x)
```

Array/FindFirstAndLastPositionOfElementInSortedArray.py

```
"""
Given an array of integers nums sorted in ascending order, find the starting and
ending position of a given target value.
Your algorithm's runtime complexity must be in the order of O(log n).
If the target is not found in the array, return [-1, -1].
Example 1:
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
Example 2:
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
要求： 时间复杂度必须是 O(log n)
看到log n 立马先想到二分，排好序的数组，刚好可以利用二分。
正好 Python 内置模块 bisect 有写二分法，稍加改进即可使用。
一般可能会想到用递归实现，下面是 Python 官方的迭代实现：
```

初始定义两个变量：
lo (lower)
hi (higher)
lo 初始为首位 0.
hi 则为 len(list).
每次都获取 list[mid] mid = (lo + hi) // 2.
list[mid] 有三种情况：

1. 与target相等。
2. 大于target.
3. 小于target.
   lo mid hi
   1 2 3 4 5 6
   若mid大于target则表示要找的目标存在于 lo 与 mid之间。
   小于则表示要找的目标存在于 mid 与 hi之间。
   要改进的地方在于，处理相等的情况：
   在不相等时先移动lo还是hi都无所谓。
   在相等时若先移动 lo 则尽可能找到的是最右边的一个。
   hi 则是最左边的一个。
   lo hi
   1 2 2 2 3
   先移动 lo
   lo hi
   1 2 2 2 3
   ↑
   没有可能在找到它。
   先移动 hi
   lo hi
   1 2 2 2 3
   ↑
   没有可能在找到它。
   所以一左一右，两次二分即可。
   测试地址：
   https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/descri

ption/
beat
94%

```
"""
class Solution(object):
    def find_right(self, nums, target):
        lo = 0
        hi = len(nums)
        equals = []
        while lo < hi:
            mid = (lo + hi) // 2
            if target == nums[mid]:
                equals.append(mid)
            if target < nums[mid]:
                hi = mid
            else:
                lo = mid + 1
        return equals[-1] if equals else -1
    def find_left(self, nums, target):
        lo = 0
        hi = len(nums)
        equals = []
        while lo < hi:
            mid = (lo + hi) // 2
            if target == nums[mid]:
                equals.append(mid)
            if target > nums[mid]:
                lo = mid + 1
            else:
                hi = mid
        return equals[-1] if equals else -1
    def searchRange(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        left = self.find_left(nums, target)
        if left == -1:
            return [-1, -1]
        return [left, self.find_right(nums, target)]
```

Array/FindMinimumInRotatedSortedArray.py

```
"""
Suppose an array sorted in ascending order is rotated at some pivot unknown to
you beforehand.
(i.e.,  [0,1,2,4,5,6,7] might become  [4,5,6,7,0,1,2]).
Find the minimum element.
You may assume no duplicate exists in the array.
Example 1:
Input: [3,4,5,1,2]
Output: 1
Example 2:
Input: [4,5,6,7,0,1,2]
Output: 0
找旋转过的排序数组中最小的数。
有旋转则旋转点最小，无则 0 最小。
beat: 100% 20ms
      48%  24ms
测试地址:
https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/
"""
class Solution(object):
    def find_rotate(self, nums):
        target = nums[0]
        lo = 1
        hi = len(nums)
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] > target:
                lo = mid + 1
            else:
                hi = mid
        return lo
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        rotate = self.find_rotate(nums)
        if rotate == len(nums):
            return nums[0]
        return nums[rotate]
```

Array/FindMinimumInRotatedSortedArrayII.py

```
"""
Suppose an array sorted in ascending order is rotated at some pivot unknown to
you beforehand.
(i.e.,  [0,1,2,4,5,6,7] might become  [4,5,6,7,0,1,2]).
Find the minimum element.
The array may contain duplicates.
Example 1:
Input: [1,3,5]
Output: 1
Example 2:
Input: [2,2,2,0,1]
Output: 0
Note:
This is a follow up problem to Find Minimum in Rotated Sorted Array.
Would allow duplicates affect the run-time complexity? How and why?
与Search in Rotated Sorted Array II 中讨论的一样，主要就是重复的数。用同样的方法即可。
解释请看 SearchInRotatedSortedArrayII.py.
beat 100% 20ms~24ms都有可能。
测试地址：
https://leetcode.com/problems/find-minimum-in-rotated-sorted-array-
ii/description/
"""
class Solution(object):
    def find_rotate(self, nums):
        target = nums[0]
        lo = 1
        for i in range(1, len(nums)):
            if nums[i] == target:
                lo += 1
            else:
                break
        hi = len(nums)
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] > target:
                lo = mid + 1
            else:
                hi = mid
        return lo
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        rotate = self.find_rotate(nums)
        if rotate == len(nums):
            return nums[0]
        return nums[rotate]
```

Array/FindPeakElement.py

```
"""
A peak element is an element that is greater than its neighbors.
Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and
return its index.
The array may contain multiple peaks, in that case return the index to any one of
the peaks is fine.
You may imagine that nums[-1] = nums[n] = -∞.
Example 1:
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number
2.
Example 2:
Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak
element is 2,
             or index number 5 where the peak element is 6.
Note:
Your solution should be in logarithmic complexity.
log n待续。
先O(n)。
看来测试数据不多，O(n) 的可以 beat 100%.
测试地址：
https://leetcode.com/problems/find-peak-element/description/
"""
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        length = len(nums)
        for i in range(1, length-1):
            if nums[i-1] < nums[i] > nums[i+1]:
                return i
        if length <= 2:
            return nums.index(max(nums))
        else:
            if nums[0] > nums[1]:
                return 0
            elif nums[-1] > nums[-2]:
                return length-1
            return None
```

Array/FizzBuzz.py

```
"""
Write a program that outputs the string representation of numbers from 1 to n.
But for multiples of three it should output “Fizz” instead of the number and for
the multiples of five output “Buzz”. For numbers which are multiples of both
three and five output “FizzBuzz”.
Example:
n = 15,
Return:
[
    "1",
    "2",
    "Fizz",
    "4",
    "Buzz",
    "Fizz",
    "7",
    "8",
    "Fizz",
    "Buzz",
    "11",
    "Fizz",
    "13",
    "14",
    "FizzBuzz"
]
1 - n
3的倍数加 Fizz
5的倍数加 Buzz
3和5的倍数加 FizzBuzz
比较简单，今日的零启动。
用取模%也可以，一开始觉得取模可能效率不高就没用。
不过看结果是一样的：
beat 94%
测试链接：
https://leetcode.com/problems/fizz-buzz/description/
"""
class Solution(object):
    def fizzBuzz(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        result = []
        fizz_time = 0
        buzz_time = 0
        fizzbuzz_time = 0
        for i in range(1, n+1):
            fizz_time += 1
            buzz_time += 1
            fizzbuzz_time += 1
            if fizzbuzz_time == 15:
                result.append("FizzBuzz")
                fizzbuzz_time = 0
                fizz_time = 0
```

```
                buzz_time = 0
                continue
            if fizz_time == 3:
                fizz_time = 0
                result.append("Fizz")
                continue
            if buzz_time == 5:
                buzz_time = 0
                result.append("Buzz")
                continue
            result.append(str(i))
        return result
```

Array/FootballFans.py

```
"""
好像是今日头条2018 秋招算法题？
球场C 可容纳 M*N个球迷。统计有几组球迷群体：
1. 每组球迷群体会选择相邻的作为（8个方位。）
2. 不同球迷群体不会相邻。
3. 给一个 M*N 的二维球场，0为没人，1为有人，求出群体个数P以及最大的群体人数Q.
OK, bug free.
类似算法的 leetcode：
https://leetcode.com/problems/number-of-islands/description/  解答请看
NumberOfIslands.py
https://leetcode.com/problems/max-area-of-island/description/ 解答请看
MaxAreaOfIsland.py
"""
test = [
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 1, 0, 1, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 1, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 1, 1],
[0, 0, 0, 1, 1, 1, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
[0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
]
# 6 8
def makeAroundXY(x, y):
    # ((x, y)....)
    return ((x, y-1),
            (x, y+1),
            (x-1, y),
            (x+1, y),
            (x-1, y-1),
            (x-1, y+1),
            (x+1, y+1),
            (x+1, y-1))
def getFootballFans(court):
    """
        [[0, 0, 0].....]
        从 0,0 开始，如果遇到1则进入递归：
        递归结束条件：
            四周都是 0或边界。
            结束时将搜索到的人数添加。
        未结束时根据四周的情况进入相同的递归。
    """
    fans_groups = []
    x = 0
    y = 0
    x_length = len(court[0])
    y_length = len(court)
    def helper(x, y, result=0):
        nonlocal court
        Xy = makeAroundXY(x, y)
        for i in Xy:
```

```
            try:
                if i[0] < 0 or i[1] < 0:
                    continue
                if court[i[1]][i[0]] == 1:
                    court[i[1]][i[0]] = 0
                    result += 1
                    t = helper(i[0], i[1], 0)
                    result += t
            except IndexError:
                continue
        else:
            return result
    for y in range(y_length):
        for x in range(x_length):
            if court[y][x] == 1:
                court[y][x] = 0
                fans_groups.append(helper(x, y, 1))
    if not fans_groups:
        return (0, 0)
    return (len(fans_groups, max(fans_groups)))
getFootballFans(test)
```

Array/FruitIntoBaskets.py

```
"""
In a row of trees, the i-th tree produces fruit with type tree[i].
You start at any tree of your choice, then repeatedly perform the following
steps:
1. Add one piece of fruit from this tree to your baskets.  If you cannot, stop.
2. Move to the next tree to the right of the current tree.  If there is no tree
to the right, stop.
Note that you do not have any choice after the initial choice of starting tree:
you must perform step 1, then step 2, then back to step 1, then step 2, and so on
until you stop.
You have two baskets, and each basket can carry any quantity of fruit, but you
want each basket to only carry one type of fruit each.
What is the total amount of fruit you can collect with this procedure?
Example 1:
Input: [1,2,1]
Output: 3
Explanation: We can collect [1,2,1].
Example 2:
Input: [0,1,2,2]
Output: 3
Explanation: We can collect [1,2,2].
If we started at the first tree, we would only collect [0, 1].
Example 3:
Input: [1,2,3,2,2]
Output: 4
Explanation: We can collect [2,3,2,2].
If we started at the first tree, we would only collect [1, 2].
Example 4:
Input: [3,3,3,1,2,1,1,2,3,3,4]
Output: 5
Explanation: We can collect [1,2,1,1,2].
If we started at the first tree or the eighth tree, we would only collect 4
fruits.
对于每一个 i，都会产生 tree[i] 类型的水果。有两个篮子，每个篮子只能放一种类型，但同类型的不限次数。
问最多能摘的水果数量。
思路:
1. 一开始用的回溯法:
用两个变量表示篮子，都有水果时就追加。
第三种类型的出现时就进行回溯，回到上一个水果的点再次进行判断。
效率上最差就算 O(n²) 吧。
反正没passed就是了，90个里过了80个..
 1.2. 有个要注意的点：回溯的点选择:
     [1,0,6,6,4,6]
在 tree[2] (6) 这个点，出现了 1,0,6 三种类型，开始回溯，回溯的点是 0, 6 (1, 2) 。
在 tree[4] (4) 这个点，出现了 0,6,4 三种类型，开始回溯，回溯的点需要是 6, 4 (2, 4) 这个6是相邻的第一次出现的点。
---
2. O(n) 的进阶:
对于每一个点来说可以存储一些属性来取消回溯:
     [1,0,6,6,4,6]
count: 这个点可采集到的两种类型的水果数量。
repeat_count: 相邻的同类型水果数量。
capacity: 篮子里的水果类型。
```

self-value: 这个点可以采集的水果类型。
那么对于下一个点，只需要判断：
1. 是不是同类型：
    同类型 repeat_count 和 count 都 + 1.
   不是看2.
2. 是不是在篮子里：
    是则只把 count + 1 ，同时 repeat_count 和 self-value 更新为1与此点的类型。
   不是看3.。
 2.1 篮子没满，没满就 count + 1 重置 self-value repeat_count 并在 capacity 加上 这个点。
3. 这一步是出现了不在篮子里的第三种水果类型，出现之后：
   count repeat_count capacity 和 self-value 全部重置。
   count 为上一个点的 repeat_count + 1
   repeat_count 为 1
   capacity 重置为 上一个 点的 self-value + 现在的 self-value
   self_value 就此点的值。
最后输出 max count即可。
这个passed. 784ms.
测试链接：
https://leetcode.com/contest/weekly-contest-102/problems/fruit-into-baskets/
"""
```
class Solution(object):
    def totalFruit(self, tree):
        """
        :type tree: List[int]
        :rtype: int
        """
        # [{count, repeat_count, capacity, self-value}, ]
        tree_seed = [
                        {'count': 1, 'repeat_count': 1, 'capacity':
set([tree[0]]), 'self_value': tree[0]}
                    ]
        for i in range(1, len(tree)):
            prev = tree_seed[i-1]
            # repeat self_value
            if tree[i] == prev.get('self_value'):
                tree_seed.append({'count': prev.get('count') + 1,
                                  'repeat_count': prev.get('repeat_count') + 1,
                                  'capacity': prev.get('capacity'),
                                  'self_value': tree[i]})
                continue
            # already into basket but cannot beside each other.
            if tree[i] in prev.get('capacity'):
                tree_seed.append({'count': prev.get('count') + 1,
                                  'repeat_count': 1,
                                  'capacity': prev.get('capacity'),
                                  'self_value': tree[i]})
                continue
            # != but can taken into basket.
            if len(tree_seed[i-1].get('capacity')) == 1:
                new = prev.get('capacity')
                new.add(tree[i])
                tree_seed.append({'count': prev.get('count') + 1,
                                  'repeat_count': 1,
                                  'capacity': new,
                                  'self_value': tree[i]})
                continue
            # Found Third-fruit.
```

```
            new = set()
            new.add(tree[i])
            new.add(tree[i-1])
            tree_seed.append({'count': tree_seed[i-1].get('repeat_count') + 1,
                              'repeat_count': 1,
                              'capacity': new,
                              'self_value': tree[i]})
        return max(tree_seed, key=lambda x: x['count']).get('count')
#         maxes = 0
#         one = None
#         two = None
#         count = 0
#         # for index in range(len(tree)):
#         index = 0
#         length = len(tree)
#         while index < length:
#             i = tree[index]
#             if one is None:
#                 one = i
#                 count += 1
#                 index += 1
#                 continue
#             if i == one:
#                 count += 1
#                 index += 1
#                 continue
#             if two is None:
#                 two = i
#                 count += 1
#                 index += 1
#                 continue
#             if i == two:
#                 count += 1
#                 index += 1
#                 continue
#             maxes = max(maxes, count)
#             count = 1
#             one = tree[index-1]
#             two = None
#             for t in range(index-2, -1, -1):
#                 if tree[t] == one:
#                     index = t+1
#                 else:
#                     break
#         else:
#             maxes = max(maxes, count)
        # return maxes
```

Array/GasStation.py

```
"""
There are N gas stations along a circular route, where the amount of gas at
station i is gas[i].
You have a car with an unlimited gas tank and it costs cost[i] of gas to travel
from station i to its next station (i+1). You begin the journey with an empty
tank at one of the gas stations.
Return the starting gas station's index if you can travel around the circuit once
in the clockwise direction, otherwise return -1.
Note:
If there exists a solution, it is guaranteed to be unique.
Both input arrays are non-empty and have the same length.
Each element in the input arrays is a non-negative integer.
Example 1:
Input:
gas  = [1,2,3,4,5]
cost = [3,4,5,1,2]
Output: 3
Explanation:
Start at station 3 (index 3) and fill up with 4 unit of gas. Your tank = 0 + 4 =
4
Travel to station 4. Your tank = 4 - 1 + 5 = 8
Travel to station 0. Your tank = 8 - 2 + 1 = 7
Travel to station 1. Your tank = 7 - 3 + 2 = 6
Travel to station 2. Your tank = 6 - 4 + 3 = 5
Travel to station 3. The cost is 5. Your gas is just enough to travel back to
station 3.
Therefore, return 3 as the starting index.
Example 2:
Input:
gas  = [2,3,4]
cost = [3,4,3]
Output: -1
Explanation:
You can't start at station 0 or 1, as there is not enough gas to travel to the
next station.
Let's start at station 2 and fill up with 4 unit of gas. Your tank = 0 + 4 = 4
Travel to station 0. Your tank = 4 - 3 + 2 = 3
Travel to station 1. Your tank = 3 - 3 + 3 = 3
You cannot travel back to station 2, as it requires 4 unit of gas but you only
have 3.
Therefore, you can't travel around the circuit once no matter where you start.
给两个数组：
一个 gas 一个 cost，gas 是所包含的气，cost 则是要花费的气。
问如果能从某个站点开始行使完一圈，那么开始的点在哪。
思路：
如果 sum(gas) < sum(cost)，那么一定是不可能跑完全程的。
注意事项里说过只存在一个解决方法。
那么既然全程的气是足够的，那么一定有一个点可以开始。
1. 从头到尾，若 gas - cost 有剩余，那么是一个潜在的点，此时继续向下进行并记录 rest gas，若之
后某站 gas 不够了，那之前记录的点就不是。
2. 此时重新开始，重复 1. 即可。
beat 66%。
测试地址：
https://leetcode.com/problems/gas-station/description/
```

```
"""
class Solution(object):
    def canCompleteCircuit(self, gas, cost):
        """
        :type gas: List[int]
        :type cost: List[int]
        :rtype: int
        """
        if sum(gas) < sum(cost):
            return -1
        rest = 0
        station = None
        for i in range(len(gas)):
            if rest + gas[i] - cost[i] >= 0:
                rest = rest + gas[i] - cost[i]
                if station is None:
                    station = i
            else:
                rest = 0
                station = None
        return station
```

Array/GlobalAndLocalInversions.py

```
"""
We have some permutation A of [0, 1, ..., N - 1], where N is the length of A.
The number of (global) inversions is the number of i < j with 0 <= i < j < N and
A[i] > A[j].
The number of local inversions is the number of i with 0 <= i < N and A[i] >
A[i+1].
Return true if and only if the number of global inversions is equal to the number
of local inversions.
Example 1:
Input: A = [1,0,2]
Output: true
Explanation: There is 1 global inversion, and 1 local inversion.
Example 2:
Input: A = [1,2,0]
Output: false
Explanation: There are 2 global inversions, and 1 local inversion.
Note:
A will be a permutation of [0, 1, ..., A.length - 1].
A will have length in range [1, 5000].
The time limit for this problem has been reduced.
思路:
 二分法。
测试地址:
https://leetcode.com/contest/weekly-contest-69/problems/global-and-local-
inversions/
"""
import bisect
class Solution(object):
    def isIdealPermutation(self, A):
        """
        :type A: List[int]
        :rtype: bool
        """
        global_inversions = []
        _g = 0
        for i in A[::-1]:
            _g += bisect.bisect_left(global_inversions, i)
            bisect.insort_left(global_inversions, i)
        _l = 0
        for i in range(len(A)-1):
            if A[i] > A[i+1]:
                _l += 1
        return _g == _l
```

Array/GroupAnagrams.py

```
"""
Given an array of strings, group anagrams together.
Example:
Input: ["eat", "tea", "tan", "ate", "nat", "bat"],
Output:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
Note:
All inputs will be in lowercase.
The order of your output does not matter.
给一组带有字符串的数组，进行变位词分类。
思路是排序的思路，排序后为同一个的放到一起。
效率就测试来看尚可。
beat 70%
测试地址：
https://leetcode.com/problems/group-anagrams/description/
"""
class Solution(object):
    def groupAnagrams(self, strs):
        """
        :type strs: List[str]
        :rtype: List[List[str]]
        """
        result = {}
        for i in strs:
            key = ''.join(sorted(i))
            try:
                result[key].append(i)
            except:
                result[key] = [i]
        return result.values()
```

Array/IncreasingTripletSubsequence.py

```
"""
Given an unsorted array return whether an increasing subsequence of length 3
exists or not in the array.
Formally the function should:
Return true if there exists i, j, k
such that arr[i] < arr[j] < arr[k] given 0 ≤ i < j < k ≤ n-1 else return false.
Note: Your algorithm should run in O(n) time complexity and O(1) space
complexity.
Example 1:
Input: [1,2,3,4,5]
Output: true
Example 2:
Input: [5,4,3,2,1]
Output: false
给一个数组，若里面存在 arr[i] < arr[j] < arr[k] 且 0 <= i < j < k <= n-1，则返回True。
否则False.
要求 O(n)时间以及O(1) 空间。
对于 i 的规则是尽量小。首先确定为第一个，之后若出现比它小的，则确定为这个否则确定为 j。
对于 j 的规则依然是尽量小，但要大于i。
思考时有可能出现的陷阱：
1. 有比 i 小的就替换，那要是 arr[100] < i，那前面的99个确定没有符合条件的吗？
   还要基于 j，i的判断条件是，若比 i 小则替换为 i，若比 i 大则与 j 比较，与 j 比较遵循同样的
规则。
2. 有无可能出现在 j 之前有 3 个小于 j 的递增序列呢？
   不可能：
      一个数在 j 之前有两种状态：
         比 j 大或与 j 相等。
         若比 j 小则替换为 j 了。
    在 j 之后也不可能，若这个数比 j 小则会替换为 j。
3. 那么有没有可能出现第一个 j 与第二个 j 之间存在 3 个这样的递增序列呢？
   也是不可能的：
       若比 j 大则直接返回了。
       若比 j 小或等则替换。
所以按此规则：
1. i 之前不可能存在 3 个递增序列。
2. i - j 这段也不可能存在 3 个递增序列。
3. j - k 这段也不可能存在 3 个递增序列。
o(n) o(1)
     只有 4 个变量。
beat 100%
测试地址：
https://leetcode.com/problems/increasing-triplet-subsequence/description/
"""
class Solution(object):
    def increasingTriplet(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        if len(nums) < 3:
            return False
        one = None
        two = None
        for i, d in enumerate(nums):
```

```
            if one:
                if d <= one[1]:
                    one = (i, d)
                    continue
                if two:
                    if d <= two[1]:
                        two = (i, d)
                    else:
                        return True
                else:
                    two = (i, d)
            else:
                one = (i, d)
        return False
```

Array/InsertInterval.py

```
"""
Given a set of non-overlapping intervals, insert a new interval into the
intervals (merge if necessary).
You may assume that the intervals were initially sorted according to their start
times.
Example 1:
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]
Example 2:
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].
与 Merge Array 思路上是一样的，本以为用同样的代码会导致 TLE，不过也 Pass 了，beat 74%.
C扩展的排序就是快，不用排序的思路：
这个只会部分重叠，所以目标是找到head和end的点：
[[1,2],[3,5],[6,7],[8,10],[12,16]]  [4, 8]
head 找最后一个大于的。
end 则找第一个小于的。
比如 4 对比 1，3，6后，那么确定 head 为 3. [3, 5]
    8  对比 2，5，7 后找到 10。  [8, 10]
测试地址：
https://leetcode.com/problems/insert-interval/description/
"""
# Definition for an interval.
# class Interval(object):
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e
class Solution(object):
    def insert(self, _sentences, newInterval):
        """
        :type intervals: List[Interval]
        :type newInterval: Interval
        :rtype: List[Interval]
        """
        _sentences.append(newInterval)
        _sentences = sorted(_sentences, key=lambda x: x.start)
        if not _sentences:
            return []
        result = []
        head = _sentences[0].start
        tail = _sentences[0].end
        length = len(_sentences)
        for x in range(1, length):
            i = _sentences[x]
            if tail >= i.start:
                tail = max(tail, i.end)
            else:
                result.append([head, tail])
                head = i.start
                tail = i.end
        result.append([head, tail])
        return result
```

Array/IntersectionOfTwoArrays.py

```
"""
Given two arrays, write a function to compute their intersection.
Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]
Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
Note:
Each element in the result must be unique.
The result can be in any order.
找两个数组里的重复数据。
唯一，任意顺序。
集合，取交集即可。
零启动。
beat 100% 20ms.
测试地址：
https://leetcode.com/problems/intersection-of-two-arrays/description/
"""
class Solution(object):
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        return list(set(nums1) & set(nums2))
```

Array/IntersectionOfTwoArraysII.py

```
"""
Given two arrays, write a function to compute their intersection.
Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]
Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
Note:
Each element in the result should appear as many times as it shows in both
arrays.
The result can be in any order.
Follow up:
What if the given array is already sorted? How would you optimize your
algorithm?
What if nums1's size is small compared to nums2's size? Which algorithm is
better?
What if elements of nums2 are stored on disk, and the memory is limited such that
you cannot load all elements into the memory at once?
与 1 差不多，不过这次有重复。
进阶条件里问：
1. 如果是已排序的可以怎么优化？
2. 如果 数组1 比 数组2 要小，哪个算法好些？
3. 如果 数组2 在硬盘里排过序，但内存不足以一次性全部读取该怎么做？
我的思路是：
1. 先排序。
2. 之后设置两个指针。
若 1与2相同，则将结果添加到最终结果的列表中，1和2各+1。
若 1比2要大，那么表示2向后走还有可能遇到和1相同的数字，所以2 +1。
否则 1 +1。直到有一个到了末尾。
这个思路的话，进阶的1和3直接可以包含进去。
2的话，Discuss 里的其他方法基本上是用 哈希表。 1和2哈希，然后取个数较小的交集，这种方法在数组
较小的时候要比上面提到的思路快。
排序后的思路：
beat 100% 24ms.
测试地址：
https://leetcode.com/problems/intersection-of-two-arrays-ii/description/
"""
class Solution(object):
    def intersect(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        result = []
        nums1.sort()
        nums2.sort()
        _n1 = 0
        _n2 = 0
        _n1_length = len(nums1)
        _n2_length = len(nums2)
        while _n1 < _n1_length and _n2 < _n2_length:
            if nums1[_n1] == nums2[_n2]:
```

```
                result.append(nums1[_n1])
                _n1 += 1
                _n2 += 1
            elif nums1[_n1] < nums2[_n2]:
                _n1 += 1
            else:
                _n2 += 1
        return result
```

Array/IntersectionOfTwoLinkedLists.py

```
"""
Write a program to find the node at which the intersection of two singly linked
lists begins.
For example, the following two linked lists:
A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:     b1 → b2 → b3
begin to intersect at node c1.
Notes:
If the two linked lists have no intersection at all, return null.
The linked lists must retain their original structure after the function
returns.
You may assume there are no cycles anywhere in the entire linked structure.
Your code should preferably run in O(n) time and use only O(1) memory.
给定两个链表，判断是否有交叉部分。
分析：
那么，就有了以下4种情况：
 1. 长度相同，有交叉部分。
 2. 长度不同，有交叉部分。
 3. 长度相同，无交叉部分。
 4. 长度不同，无交叉部分。
1. 两个链表，若存在交叉部分则最后至交叉点一定是相同的。
那么倒序判断可以说应该是最高效的，从两个链表的尾部开始，直至找到不同部分或一方为None表示无交叉。
由给定的链表节点可知，这是一只单向链表，所以此思路已经无法在继续进行。 O(n)
2. 另一个思路是根据上面的信息，顺序进行判断，
让长链表一方先走，然后与短的一起走，直至找到相同部分或一方为None表示无交叉，
但我们也不知道长度，只能先遍历一遍找到长度。O(2(m+n))
3. 用的 Python 可以直接利用set()，一个哈希表，来达到O(1)的查找...
所以 原本的做法是，遍历b，然后判断b中的每一个是否在a中存在，存在则返回。这种做法简单粗暴..
但相应的复杂度是O(mn).
1. 不可行，3.有点无脑。用2.来做一下
此做法参考了 Discuss 里的高票回答：
反正是要遍历两遍，直接让两个一起走，
要么一起结束：
 1. 有相同返回相同.
 2. 无相同，返回None.
要么一长一短：
 1. 短的肯定是先走完的，然后让短的变成长的。
 2. 短的变成长的之后原本的长的因为走了一段所以变成了较短的，所以会先走完。走的这段距离就是原本我
们要求的差值。
 3. 原本的长的再变成短的，一起走完即可。
例：
a = [1, 2, 3, 4]
b = [5, 6, 7]
t = 1
f = 5
t != f
t = 2
f = 6
t != f
t = 3
f = 7
```

```
t != f
此时f已经走到的尽头，f替换为a.
t = 4
f = a.head = 1
t != f
此时t已经走到了尽头，t替换为b.
t = b.head = 5
f = 2
此时已经与我们预想的一致了。
a = [1, 2, 3, 4]
b =    [5, 6, 7]
"""
class ListNode(object):
    __slots__ = ('val', 'next')
    def __init__(self, x):
        self.val = x
        self.next = None
class Solution(object):
    def getIntersectionNode(self, headA, headB):
        """
        :type head1, head1: ListNode
        :rtype: ListNode
        """
        a = headA
        b = headB
        if not a or not b:
            return None
        while a != b:
            a = headB if a is None else a.next
            b = headA if b is None else b.next
        return a
```

Array/JumpGame.py

```
"""
Given an array of non-negative integers, you are initially positioned at the
first index of the array.
Each element in the array represents your maximum jump length at that position.
Determine if you are able to reach the last index.
Example 1:
Input: [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
Example 2:
Input: [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum
             jump length is 0, which makes it impossible to reach the last
index.
给一组数组，返回能不能到达最后的位置。每个位置包含的是从此点出发可走的最远距离，在哪里落脚随意。
思路：
从 0 开始，找到剩下的里面 下标 + 能走多远 最大的一个。然后一直重复即可。
下面的这个代码并不是一遍的 O(n) ，用此思路也可以优化成 O(n)，不优化了，没什么优化难点。
关键字：
current_index 变更为上次range的末尾，并相应的减去。
beat 66%
测试地址：
https://leetcode.com/problems/jump-game/description/
"""
class Solution(object):
    def canJump(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        dp = [nums[0]]
        current_index = 0
        while 1:
            if current_index + dp[-1] >= len(nums)-1:
                return True
            base = 0
            index = current_index
            for i in range(current_index, current_index+dp[-1]+1):
                if nums[i] + i > base:
                    base = nums[i] + i
                    index = i
            if current_index == index:
                return False
            current_index = index
            dp.append(base-index)
```

Array/JumpGameII.py

```
"""
Given an array of non-negative integers, you are initially positioned at the
first index of the array.
Each element in the array represents your maximum jump length at that position.
Your goal is to reach the last index in the minimum number of jumps.
Example:
Input: [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2.
    Jump 1 step from index 0 to 1, then 3 steps to the last index.
Note:
You can assume that you can always reach the last index.
与 1 差不多，这个需要返回的是最少的可用步数。
因为 1 中直接返回的是最大的，所以可以顺便把 2 也给完成。
beat 66%.
测试地址：
https://leetcode.com/problems/jump-game-ii/description/
"""
class Solution(object):
    def jump(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums) == 1:
            return 0
        # dp = [nums[0]]
        maxes = nums[0]
        count = 1
        current_index = 0
        length = len(nums) - 1
        while 1:
            if current_index + maxes >= length:
                return count
            base = 0
            index = current_index
            for i in xrange(current_index, current_index+maxes+1):
                if nums[i] + i > base:
                    base = nums[i] + i
                    index = i
            current_index = index
            maxes = base-index
            count += 1
```

Array/KthLargestElementInAnArray.py

```
"""
Find the kth largest element in an unsorted array. Note that it is the kth
largest element in the sorted order, not the kth distinct element.
Example 1:
Input: [3,2,1,5,6,4] and k = 2
Output: 5
Example 2:
Input: [3,2,3,1,2,4,5,5,6] and k = 4
Output: 4
Note:
You may assume k is always valid, 1 ≤ k ≤ array's length.
返回第 k 大个数，与 Third maximum numbers 思路一致：
找到第 k 大个数一般的思路有：
1. 排序后放到数组中，排序算法使用归并和快排在理想情况下都是O(nlogn)，归并比较稳定一些。之后的索
引是O(1)。
    这种的适合并不需要插入的情况，因为每次插入的时间复杂度为 O(n)。
2. 建立二叉搜索树，进阶的话红黑树或AVL树。
    这种情况下搜索和插入在理想情况下都是O(logn)。
由于不会有插入，所以选用排序。
测试用例：
https://leetcode.com/problems/kth-largest-element-in-an-array/description/
内置排序beat 99%，重新写一下归并和快排比比试试。
自己写的归并排序，beat 40% .
快速排序由于基准点问题第一次直接 TLE.
好吧，修改一下，选取基准点的方法基本是：
1. 取左端或右端。
2. 随机取。
3. 取左右中三者中中间的一个。
三者取中间的一个直接用了排序，做了个对比：
内置的sorted在用key的时候是最快的，自己的merge sort 用 key 会比较慢。
差距很明显，如果用的内置sorted，时间基本与merge sort一致，若用的merge sort，时间慢了100倍。
多次测试后：
内置 < merge sort < quick sort.
merge sort 和 quick sort 差距较小。
"""
class Solution(object):
    def quickSort(self, unsorted_list):
        """
            每次都选一个基准点，大的放在右边，小的放在左边，等于的随便归到一个地方，不断拆分拆
分。
            这里直接选用[0]，当然这种情况下往往会发生不理想的情况，
            不理想的情况表示每次恰好都是最小或最大，这样的结果会直接导致算法变为O(n^2).
        """
        if len(unsorted_list) <= 1:
            return unsorted_list
        left = []
        right = []
        meta = self._getMiddle(unsorted_list)
        for i in unsorted_list[:meta] + unsorted_list[meta+1:]:
            if i <= unsorted_list[meta]:
                left.append(i)
                continue
            right.append(i)
```

```
        return self.quickSort(left) + [unsorted_list[meta]] +
self.quickSort(right)
    def _getMiddle(self, unsorted_list):
        """
            返回快排所需的基准点，
            左右中中间选择一个。
            若不足3位，选左。
        """
        if len(unsorted_list) < 3:
            return 0
        left = unsorted_list[0]
        right = unsorted_list[-1]
        middle = unsorted_list[len(unsorted_list) // 2]
        l, r, m = [(0, left), (len(unsorted_list) - 1, right),
(len(unsorted_list) // 2, middle)]
        # 这里对比了自己写的merge sort 与内置的差距，
        # 在有key的情况下差距非常大。
        return sorted([l, r, m], key=lambda x: x[1])[1][0]
    def mergeSort(self, unsorted_list, key=None):
        """
            归并排序的基本思路是分治，把一个大问题分解成小问题。逐个解决小问题。
            以长度的一半为基准点，将一个大列表分为两个小列表，一直分一直分，然后合并。
            所以排序分为两步：
               第一步是分解，第二步是合并。
        """
        if len(unsorted_list) <= 1:
            return unsorted_list
        break_point = len(unsorted_list) // 2
        left = self.mergeSort(unsorted_list[:break_point])
        right = self.mergeSort(unsorted_list[break_point:])
        return self._decomposeAndMerge(left, right, key=key)
    def _decomposeAndMerge(self, unsorted_list_A, unsorted_list_B, key):
        sorted_list = []
        a = 0
        b = 0
        length_A = len(unsorted_list_A)
        length_B = len(unsorted_list_B)
        while a < length_A and b < length_B:
            if unsorted_list_A[a] <= unsorted_list_B[b]:
                sorted_list.append(unsorted_list_A[a])
                a += 1
            else:
                sorted_list.append(unsorted_list_B[b])
                b += 1
        if a < length_A:
            sorted_list.extend(unsorted_list_A[a:])
        else:
            sorted_list.extend(unsorted_list_B[b:])
        return sorted_list
    def findKthLargest(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        return sorted(nums, reverse=True)[k-1]
s = Solution()
a = list(range(50000))
```

```
print(s.quickSort(a))
# 1
```

Array/LargestNumber.py

```
"""
Given a list of non negative integers, arrange them such that they form the
largest number.
Example 1:
Input: [10,2]
Output: "210"
Example 2:
Input: [3,30,34,5,9]
Output: "9534330"
Note: The result may be very large, so you need to return a string instead of an
integer.
思路是补位:
9 > 34
位数不够的补齐。
一开始的补位用的是最后一位。测试时发现一个错误:
8247
824
按照一开始的补位规则:
824 会补成 8244
e...虽然调整后通过了测试，不过最终结果是缺少了一些测试例子。
我做的调整是，从补位补最后一个变为补最大的一位。但是:
284
2847 这种情况下，会以
284 2847 排，但应该是:
2847 284
----
所以还是有问题，在Discuss里提个Issue.
---
Python2的话可以用 soted的 cmp参数，不过3中已经不存在了。
暂时不搞了...
测试地址:
https://leetcode.com/problems/largest-number/description/
beat 38%~77%.
"""
class Solution(object):
    def largestNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: str
        """
        if not any(nums):
            return '0'
        max_nums = len(str(max(nums)))
        # 2
        def mycmp(x, y):
            if x + y > y + x:
                return 1
            else:
                return -1
        # 测试用下面的可以跑过 2 & 3。
        def makeEqual(s, length=max_nums):
            if len(s) == length:
                return s
            # 这种补位会通过测试，但是 Leetcode 的测试并没有包含所有的情况。
```

```
            x = max(s) * (length - len(s))
            return s+x
        # 2
        return ''.join(sorted(map(str, nums), cmd=mycmp, reverse=True))
        # 3
        return ''.join(sorted(map(str, nums), key=makeEqual, reverse=True))
        # print(sorted(map(str, nums), key=makeEqual, reverse=True))
```

Array/LinkedListCycle.py

```
"""
Given a linked list, determine if it has a cycle in it.
Follow up:
Can you solve it without using extra space?
基本思路：
形成环就是后面的节点中的next指向了前面出现过的节点。
下面这个用了额外空间。
改进：
使用 O(1) 空间的解决方法：
思路是两个指针：
一个每次走一步，另一个每次走两步，若有一个环，那么走两步的与走一步的会在走过这个环的长度后相遇。
相当于两个人跑步，一个每秒跑两米，一个跑一米，绕着100米的圆形跑，100秒过后，一米的这个跑了一圈，二米的这个跑了两圈，但它们相遇了。
测试地址：
https://leetcode.com/problems/linked-list-cycle/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        # while head:
        #     if hasattr(head, 'hasVisited'):
        #         return True
        #     head.hasVisited = True
        #     head = head.next
        # return False
        if not head:
            return False
        two_head = head.next
        if not two_head:
            return False
        while head != None and two_head != None:
            if head == two_head:
                return True
            head = head.next
            try:
                two_head = two_head.next.next
            except:
                return False
        return False
```

Array/LongestConsecutiveSequence.py

```
"""
Given an unsorted array of integers, find the length of the longest consecutive
elements sequence.
Your algorithm should run in O(n) complexity.
Example:
Input: [100, 4, 200, 1, 3, 2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore
its length is 4.
给定一个数组，返回里面最长的连续子序列。
[100, 4, 200, 1, 3, 2]
需要 O(n) 的时间复杂度。
第一种思路：
排序 + 遍历。
虽然很明显时间复杂度不是 O(n)。Python 的底层用的应该是一个优化过的归并排序，
所以排序时间复杂度基本为 O(nlogn)。
不过很容易写出来，对于Python 来说c++内置的排序效率无疑非常高效。
这种方法的话，平均 28ms 在 Python 里 beat 65%。
第二种思路：
用堆代替排序。构建堆的时间复杂度是 O(n)，这个就算了，底层堆构建自己暂时不会写，等能自己写出稳定
的堆再来补上。
第三种思路：
在 Discuss 里找到的：
首先set一遍nums，也就是对应的哈希一遍。
之后迭代nums，然后在set里找n+1,n+2,n+3...直到不存在的一个，记下最大长度。
不过只有这样是不行，对于在一条递增链上的数个数，就会寻找数次。
[100, 200, 1, 4, 3, 2]
100, 200 都没什么事。
1, 4, 3, 2
1 会查询4次。
3 会查询1次。
2 会查询2次。
但除了 1 之外，都没必要查询。
要减少这些没必要的查询，要找到的是 *最小* 这个点。既然是最小，那么n-1肯定是不在这个里面的。
所以可以利用这个条件来减少不必要的查询数量。
不要小看这个操作，如果最长的有10000个数。那么无用的子查询会有 10000 的阶加 50005000 次。
写一下。
ok，这个更高效，提高了4 ms。
第四种思路：
也是在 Discuss 里找到的：
来日再战。
We will use HashMap. The key thing is to keep track of the sequence length and
store that in the boundary points of the sequence. For example, as a result, for
sequence {1, 2, 3, 4, 5}, map.get(1) and map.get(5) should both return 5.
Whenever a new element n is inserted into the map, do two things:
See if n - 1 and n + 1 exist in the map, and if so, it means there is an existing
sequence next to n. Variables left and right will be the length of those two
sequences, while 0 means there is no sequence and n will be the boundary point
later. Store (left + right + 1) as the associated value to key n into the map.
Use left and right to locate the other end of the sequences to the left and right
of n respectively, and replace the value with the new length.
Everything inside the for loop is O(1) so the total time is O(n). Please comment
if you see something wrong. Thanks.
public int longestConsecutive(int[] num) {
```

```
    int res = 0;
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    for (int n : num) {
        if (!map.containsKey(n)) {
            int left = (map.containsKey(n - 1)) ? map.get(n - 1) : 0;
            int right = (map.containsKey(n + 1)) ? map.get(n + 1) : 0;
            // sum: length of the sequence n is in
            int sum = left + right + 1;
            map.put(n, sum);
            // keep track of the max length
            res = Math.max(res, sum);
            // extend the length to the boundary(s)
            // of the sequence
            // will do nothing if n has no neighbors
            map.put(n - left, sum);
            map.put(n + right, sum);
        }
        else {
            // duplicates
            continue;
        }
    }
    return res;
}
"""
class Solution(object):
    def longestConsecutive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        numx = set(nums)
        maxes = 0
        for i in nums:
            if i-1 not in numx:
                x = i
                currentMaxes = 1
                while 1:
                    if x+1 in numx:
                        x += 1
                        currentMaxes += 1
                        continue
                    else:
                        maxes = max(maxes, currentMaxes)
                        break
        return maxes
        # sorted.
        # if not nums:
        #     return 0
        # nums.sort()
        # currentNums = nums[0]
        # result = []
        # currentMax = 1
        # for i in nums[1:]:
        #     if i == currentNums + 1:
        #         currentMax += 1
        #         currentNums = i
        #     elif i == currentNums:
```

```
        #         pass
        #     else:
        #         result.append(currentMax)
        #         currentMax = 1
        #         currentNums = i
        # result.append(currentMax)
        # return max(result)
```

Array/MaxAreaOfIsland.py

```
"""
与今日头条秋招第一题最相似的一道题，只是方向少了四个。
Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's
(representing land) connected 4-directionally (horizontal or vertical.) You may
assume all four edges of the grid are surrounded by water.
Find the maximum area of an island in the given 2D array. (If there is no island,
the maximum area is 0.)
Example 1:
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,1,1,0,1,0,0,0,0,0,0,0,0],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
Given the above grid, return 6. Note the answer is not 11, because the island
must be connected 4-directionally.
Example 2:
[[0,0,0,0,0,0,0,0]]
Given the above grid, return 0.
Note: The length of each dimension in the given grid does not exceed 50.
passed:
beat 54%.
测试地址：
https://leetcode.com/problems/max-area-of-island/description/
"""
class Solution(object):
    def makeAroundXY(self, x, y):
        return ((x, y-1),
                (x, y+1),
                (x-1, y),
                (x+1, y))
    def maxAreaOfIsland(self, court):
        """
        :type grid: List[List[str]]
        :rtype: int
        """
        fans_groups = []
        x = 0
        y = 0
        if not court:
            return 0
        x_length = len(court[0])
        y_length = len(court)
        def helper(x, y, result=0):
            Xy = self.makeAroundXY(x, y)
            for i in Xy:
                try:
                    if i[0] < 0 or i[1] < 0:
                        continue
                    if court[i[1]][i[0]] == 1:
                        court[i[1]][i[0]] = 0
                        result += 1
```

```
                    t = helper(i[0], i[1], 0)
                    result += t
            except IndexError:
                continue
        else:
            return result
    for y in range(y_length):
        for x in range(x_length):
            if court[y][x] == 1:
                court[y][x] = 0
                fans_groups.append(helper(x, y, 1))
    if not fans_groups:
        return 0
    return max(fans_groups)
```

Array/MaximizeDistanceToClosestPerson.py

```
"""
In a row of seats, 1 represents a person sitting in that seat, and 0 represents
that the seat is empty.
There is at least one empty seat, and at least one person sitting.
Alex wants to sit in the seat such that the distance between him and the closest
person to him is maximized.
Return that maximum distance to closest person.
Example 1:
Input: [1,0,0,0,1,0,1]
Output: 2
Explanation:
If Alex sits in the second open seat (seats[2]), then the closest person has
distance 2.
If Alex sits in any other open seat, the closest person has distance 1.
Thus, the maximum distance to the closest person is 2.
Example 2:
Input: [1,0,0,0]
Output: 3
Explanation:
If Alex sits in the last seat, the closest person is 3 seats away.
This is the maximum distance possible, so the answer is 3.
Note:
1 <= seats.length <= 20000
seats contains only 0s or 1s, at least one 0, and at least one 1.
Alex 要坐在任意一个 0 上，问坐在哪个点上距离最近的 1是最远的。
思路：
1. 处理好边界
2. 两两相加，用 相加//2 减去前一个，不用后一个减去 相加//2 是因为如果是奇数的话会偏向前一个，
偶数无所谓，所以这样做可以少一步判断。
beat :
75%
测试地址：
https://leetcode.com/contest/weekly-contest-88/problems/maximize-distance-to-
closest-person/
"""
class Solution(object):
    def maxDistToClosest(self, seats):
        """
        :type seats: List[int]
        :rtype: int
        """
        indexes = [i for i in range(len(seats)) if seats[i] == 1]
        x = indexes[0]
        if indexes[0] != 0:
            maxes = indexes[0]
        else:
            maxes = 0
        for i in range(1, len(indexes)):
            maxes = max((x + indexes[i]) // 2 - x, maxes)
            x = indexes[i]
        if indexes[-1] != len(seats):
            maxes = max(len(seats)-indexes[-1]-1, maxes)
        return maxes
```

Array/MaximumProductSubarray.py

```
"""
Given an integer array nums, find the contiguous subarray within an array
(containing at least one number) which has the largest product.
Example 1:
Input: [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
Example 2:
Input: [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
给一个数组，找出其中连续的子数组中，相乘最大的一组。
这个算法和 maximum subarray 很相似，那个每个点的状态是 负数的话就可以断开，这个的话不行。
但也是一样的，一开始的思路是遇到负数不断开，遇到负数会乘原来的数，然后把自己加进去也作为因子。
1.
看下面注释掉的写法。这种写法对于负数并不多的可以很有效。但对付负数较多的则力不从心。
184个测试跑到183个就TLE了。
2.
基于 1. 的改进，1. 中主要是会一直加一直加，导致最差将时间复杂度升高到 O(n²)。
但对于每个点来说其实只需要保留两种状态：
第一种个状态是其中的最大值，还有一个是最小值。
[2, -5, -2, -4, 3]
这个例子中，-2 这个点若按 1. 中的写法积累下来应为 [20, 10, -2]。但10是没有必要的，-5*-2显然是没有 2*-5*-2大，而且即使后面无论是什么数都不可能出现正数
-5*-2 * x > 2*-5*-2*x的情况。
x在正数的情况下不可能大于 2x。
当x是负数时也基本不会用到它，因为求的是最大值，后面只有出现另一个负数才会相乘得正，x*-y 也不可能大于 2x*-y 的。
所以2算是给1做了剪枝。
这个算法还不错，40ms，前面的也都是基于同样的思路，只保存最大最小。
效率 O（n）。
看了 Discuss 好像也都是这个写法。
测试地址：
https://leetcode.com/problems/maximum-product-subarray/discuss/
"""
class Solution(object):
    def maxProduct(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        maxes = nums[0]
        currentNums = [[nums[0]]]
        for i in range(1, len(nums)):
            x = nums[i]
            temp = [x*j for j in currentNums[i-1]]
            minx = min(x, min(temp))
            maxx = max(x, max(temp))
            currentNums.append([minx, maxx])
            maxes = max(maxes, maxx)
        return maxes
        # maxes = nums[0]
        # currentNums = [[nums[0]]]
#         for i in nums[1:]:
```

```
#             if not currentNums:
#                 currentNums.append(i)
#                 maxes = max(maxes, i)
#                 continue
#             if i == 0:
#                 currentNums = []
#                 maxes = max(maxes, i)
#                 continue
#             for j in range(len(currentNums[:])):
#                 t = currentNums[j]
#                 if t != 0:
#                     currentNums[j] = t*i
#                     if i < 0 or t*i < 0:
#                         currentNums.append(i)
#                 else:
#                     currentNums.append(i)
#             maxes = max(maxes, max(currentNums))
#         return maxes
```

Array/MaximumSubarray.py

```
"""
Given an integer array nums, find the contiguous subarray (containing at least
one number) which has the largest sum and return its sum.
Example:
Input: [-2,1,-3,4,-1,2,1,-5,4],
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
Follow up:
If you have figured out the O(n) solution, try coding another solution using the
divide and conquer approach, which is more subtle.
思路：
O(n)
这个与https://leetcode.com/problems/binary-tree-maximum-path-sum/description/
二叉树中最大的路径和相似。
到了数组中显得更加清晰：
每个点都有两个向上返回的状态：
大于0就向上，否则不向上。
不大于0时还要进行一次取最大值。
第一次写的时候把路径也带上了，后面发现没有必要。
带上路径比较慢 500ms - 700ms.
不带的话可以跑 28 ms.
测试地址：
https://leetcode.com/problems/maximum-subarray/description/
"""
class Solution(object):
    def maxSubArray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        maxes = nums[0]
        current = nums[0]
        for i in nums[1:]:
            maxes = max(maxes, current + i,  i)
            if current + i < i:
                current = i
            else:
                current = current + i
        return maxes
        # if not nums:
        #     return 0
        # maxes = ([nums[0]], nums[0])
        # current = ([nums[0]], nums[0])
        # for i in nums[1:]:
        #     maxes = max(maxes, (current[0]+[i], current[1] + i), ([i], i),
key=lambda x: x[1])
        #     if current[1] + i < i:
        #         current = ([i], i)
        #     else:
        #         current = (current[0]+[i], current[1] + i)
        # return maxes[1]
```

Array/MaximumSumCircularSubarray.py

```
"""
Given a circular array C of integers represented by A, find the maximum possible
sum of a non-empty subarray of C.
Here, a circular array means the end of the array connects to the beginning of
the array.  (Formally, C[i] = A[i] when 0 <= i < A.length, and C[i+A.length] =
C[i] when i >= 0.)
Also, a subarray may only include each element of the fixed buffer A at most
once.  (Formally, for a subarray C[i], C[i+1], ..., C[j], there does not exist i
<= k1, k2 <= j with k1 % A.length = k2 % A.length.)
Example 1:
Input: [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3
Example 2:
Input: [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10
Example 3:
Input: [3,-1,2,-1]
Output: 4
Explanation: Subarray [2,-1,3] has maximum sum 2 + (-1) + 3 = 4
Example 4:
Input: [3,-2,2,-3]
Output: 3
Explanation: Subarray [3] and [3,-2,2] both have maximum sum 3
Example 5:
Input: [-2,-3,-1]
Output: -1
Explanation: Subarray [-1] has maximum sum -1
Note:
-30000 <= A[i] <= 30000
1 <= A.length <= 30000
给一个前后连通的数组，求里面和最大的子数组。
不连通的之前已经做过了，每个点的状态有两个，一个是加上之前的 > 0，一个是加上之前的小于 0，此点保存的值为 加上与不加较大的一个。
连通的话呢...一开始致力于用O(n)就找出可以开始的点...走了些弯路：
1. 尝试从 0 开始找第一个非负，没有覆盖全部情况。比如这样 8 -1 -3 8 -5 -5 -5 8.
2. 尝试把点设在从后向前的负数的前面的最大正数。 比如 8 -1 -3 8 -5 -5 -5 8，就选 8 -1 -3 8 -5 -5 -5，这样可以过90%.. 出错的情况忘了记录。
3. 看样子只能 o(n²)。
结束后看 Discuss 里的讨论，大神就是大神... 首尾相连的情况其实就是 sum(A) - sum(min A[subarray])。
ok... 掌握这条信息后直接就可以做出来了，找最小的情况与最大也没什么区别。
284ms
测试地址：
https://leetcode.com/contest/weekly-contest-105/problems/maximum-sum-circular-subarray/
"""
class Solution(object):
    def maxSubarraySumCircular(self, A):
        """
        :type A: List[int]
        :rtype: int
        """
```

```
        dp = [A[0]]
        maxes = A[0]
        for i in range(1, len(A)):
            if A[i] + dp[i-1] > 0:
                maxes = max(maxes, A[i]+dp[i-1], A[i])
                dp.append(max(A[i]+dp[i-1], A[i]))
            else:
                dp.append(A[i])
                maxes = max(maxes, A[i])
        if maxes < 0:
            return maxes
        dp = [A[0]]
        mines = A[0]
        for i in range(1, len(A)):
            if A[i] + dp[-1] < 0:
                mines = min(mines, A[i]+dp[-1], A[i])
                dp.append(min(A[i]+dp[-1], A[i]))
            else:
                dp.append(A[i])
                mines = min(mines, A[i])
        maxes = max(maxes, sum(A) - mines)
        return maxes
```

Array/max_increase_to_keep_city_skyline.py

```
"""
Example:
Input: grid = [[3,0,8,4],[2,4,5,7],[9,2,6,3],[0,3,1,0]]
Output: 35
Explanation:
The grid is:
[ [3, 0, 8, 4],
  [2, 4, 5, 7],
  [9, 2, 6, 3],
  [0, 3, 1, 0] ]
The skyline viewed from top or bottom is: [9, 4, 8, 7]
The skyline viewed from left or right is: [8, 7, 9, 3]
The grid after increasing the height of buildings without affecting skylines is:
gridNew = [ [8, 4, 8, 7],
            [7, 4, 7, 7],
            [9, 4, 8, 7],
            [3, 3, 3, 3] ]
测试用例：
https://leetcode.com/problems/max-increase-to-keep-city-skyline/description/
整体思路：
获取出x和y轴的最大值，然后逐个遍历。
时间复杂度 O(mn)。
"""
class Solution(object):
    def maxIncreaseKeepingSkyline(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        length = len(grid[0])
        # Get line max.
        line_dict = {str(index):max(data) for index, data in enumerate(grid)}
        # Get column max.
        column_dict = {str(index):max((grid[index2][index] for index2 in
range(len(grid)))) for index in range(length)}
        total_increases = 0
        for index, line in enumerate(grid):
            for index2, cell in enumerate(line):
                total_increases += min([line_dict[str(index)],
column_dict[str(index2)]]) - cell
        return total_increases
```

Array/MedianOfTwoSorted.py

```
"""
There are two sorted arrays nums1 and nums2 of size m and n respectively.
Find the median of the two sorted arrays. The overall run time complexity should
be O(log (m+n)).
You may assume nums1 and nums2 cannot be both empty.
Example 1:
nums1 = [1, 3]
nums2 = [2]
The median is 2.0
Example 2:
nums1 = [1, 2]
nums2 = [3, 4]
The median is (2 + 3)/2 = 2.5
求两个列表合并后的中位数。
时间复杂度要求O(log (m+n))
1. 直接sorted后取中位，时间复杂度为O((m+n)log(m+n))。
2. 不用全部排序，本质上是一个求第k(s)小的数。 时间复杂度(O( (m+n)/2 ))还是没到要求的 O(log
(m+n))，此版beat 43%~56%。
# 打印真的是很慢，只打印一条很小的数据2.方法就会只能beat 7%..
3. 最后是 O(Log(m+n)) 的思路：
求第k小的数的时候，有一种方法是每次去除 k // 2 个数，剩余 < 1个时就找到了最终要找的数。
a = [1, 2, 4, 6]
b = [3, 5, 7, 9]
此时的 k 为 8 // 2 = 4.
k//2 = 2
则对比 a与b的第2个数也是下标为2-1哪个较小，较小的一组去除。然后循环即可。
因为是排过序的列表，不论哪一个列表中，去除 k//2 个数都不会去除目标。
因为只有k的一半，又是排过序的，假设a[k//2]<b[k//2]：
1. 去除的值只有k的一半，
2. 选择的是较小的那一组，那么组合起来：
b[:k//2-1] + a[:k//2] + b[k//2-1]
前两者无论如何都是比 b[k//2] 要小的，所以要么第k小的数是 b[k//2]，要么不是，去除a[:k//2]一点
问题都没有。
需要注意的问题：
1. 若k//2大于某一个列表时，此时调整为所大于的列表的长度，不可能出现两个列表同时都小于的情况，
   除非写错了否则不可能 len(a)+len(b) // 4 > len(a)还 > len(b)
   a / 4 + b / 4 = a时 b / 4 = 3a / 4 -> b = 3a，3b/4 = 9a/4，所以 b/4 + a/4则不可能
也大于b。
2. 奇数在取k时向上取整。
3. k//2向下取整。
这样每次去除的是k的一半，所以时间复杂度为O(log(k)) -> O(log((m+n)//2)) 符合要求。
beat 75%，前面的25%都是用的sorted，C扩展的sorted效率是最高的，也就是第一版，基本上是100%，
但是这是作弊行为= =。
测试地址：
https://leetcode.com/problems/median-of-two-sorted-arrays/description/
"""
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: float
        """
        length = len(nums1) + len(nums2)
```

```
        if length <= 2:
            return sum(nums1+nums2) / length
        raw_length = length
        length = length // 2
        if raw_length % 2 != 0:
            length += 1
        while length > 1:
            reduce_value = length // 2
            if nums1:
                if reduce_value > len(nums1):
                    reduce_value = len(nums1)
            if nums2:
                if reduce_value > len(nums2):
                    reduce_value = len(nums2)
            nums1_k_value = nums1[reduce_value-1] if nums1 else float('inf')
            nums2_k_value = nums2[reduce_value-1] if nums2 else float('inf')
            if nums1_k_value < nums2_k_value:
                nums1 = nums1[reduce_value:]
            else:
                nums2 = nums2[reduce_value:]
            length -= reduce_value
        result = sorted(nums1[:2] + nums2[:2])
        if raw_length % 2 != 0:
            return result[0]
        return sum(result[:2]) / 2
    # def findMedianSortedArrays(self, nums1, nums2):
    #     """
    #     :type nums1: List[int]
    #     :type nums2: List[int]
    #     :rtype: float
    #     """
    #     length = len(nums1) + len(nums2)
    #     if length <= 2:
    #         return sum(nums1+nums2) / length
    #     raw_length = length
    #     length = length // 2
    #     # Get kth minium number(s).
    #     if raw_length % 2 == 0:
    #         get = [length, length-1]
    #     else:
    #         get = [length]
    #     index_nums1 = 0
    #     index_nums2 = 0
    #     total_nums = 0
    #     result = []
    #     while get:
    #         if index_nums1 == len(nums1):
    #             x = nums2[index_nums2]
    #             index_nums2 += 1
    #         elif index_nums2 == len(nums2):
    #             x = nums1[index_nums1]
    #             index_nums1 += 1
    #         else:
    #             if nums1[index_nums1] < nums2[index_nums2]:
    #                 x = nums1[index_nums1]
    #                 index_nums1 += 1
    #             else:
    #                 x = nums2[index_nums2]
```

```
    #                 index_nums2 += 1
    #         if get[-1] == total_nums:
    #             result.append(x)
    #             get.pop()
    #         total_nums += 1
    #     return sum(result) / len(result)
    # sorted.
    # def findMedianSortedArrays(self, nums1, nums2):
    #     """
    #     :type nums1: List[int]
    #     :type nums2: List[int]
    #     :rtype: float
    #     """
    #     list3 = nums1 + nums2
    #     list3.sort()
    #     return self.median(list3)
    # def median(self, nums):
    #     length = len(nums)
    #     if length % 2 == 0:
    #         return (nums[(length // 2)-1] + nums[length // 2]) / 2
    #     return nums[length // 2]
```

Array/MergeArray.py

```
"""
好像是今日头条2018 秋招算法题？
有m名编辑审核文章，每个编辑独立工作，会把觉得有错误的地方通过下标标注出来：
[1, 10] 表示1-10个字符应该有问题。
现在要把多名编辑的结果合并起来：
[1, 10] [32, 45],
[5, 16] [78, 94]
那么合并后是 [1, 16] [32, 45] [78, 94]。
bug free.
Leetcode 中也有两个一样的：
https://leetcode.com/problems/merge-intervals/description/
测试通过：
beat 64%~95%...
这个更加类似：
https://leetcode.com/problems/insert-interval/description/
这个代码看 InsertInterval.py
"""
class Solution(object):
    def merge(self, _sentences):
        """
        :type intervals: List[Interval]
        :rtype: List[Interval]
        """
        _sentences = sorted(_sentences, key=lambda x: x.start)
        if not _sentences:
            return []
        result = []
        head = _sentences[0].start
        tail = _sentences[0].end
        length = len(_sentences)
        for x in range(1, length):
            i = _sentences[x]
            if tail >= i.start:
                tail = max(tail, i.end)
            else:
                result.append([head, tail])
                head = i.start
                tail = i.end
        result.append([head, tail])
        return result
test = (
    [(1, 10), (32, 45)],
    [(78, 94), (5, 16)],
    [(80, 100), (200, 220), (16, 32)]
    )
def mergeArray(sentences):
    """
        这个测试数据的结构是我自己写的，所以第一步是打散数组。
        1. 根据第一个字符出现的位置进行排序。
        2. 迭代，记录i的头，记录i的末尾，末尾与下一个i的头做比较，若前者记录的大或相等则末尾替换为两者中较大的一个。
        3. 不大的情况添加到结果中，并将头尾替换为此时的数据。
        4. 最后一轮迭代在添加一次。
    """
```

```
    # 打散，相当于原题给的数据中的读入。
    _sentences = sorted([y for x in test for y in x], key=lambda x: x[0])
    if not _sentences:
        return []
    result = []
    head = _sentences[0][0]
    tail = _sentences[0][1]
    length = len(_sentences)
    for x in range(1, length):
        i = _sentences[x]
        if tail >= i[0]:
            tail = max(tail, i[1])
        else:
            result.append([head, tail])
            head = i[0]
            tail = i[1]
    result.append([head, tail])
    print(result)
mergeArray(test)
```

Array/MergerTwoSortedList.py

```
"""
合并两个排序过的数组。
Input: 1->2->4, 1->3->4
Output: 1->1->2->3->4->4
基本就是你走一步我走一步，一人一个指向。
O(n)
测试用例：
https://leetcode.com/problems/merge-two-sorted-lists/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def mergeTwoLists(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        head = cur = ListNode(0)
        while l1 and l2:
            if l1.val < l2.val:
                cur.next = l1
                l1 = l1.next
            else:
                cur.next = l2
                l2 = l2.next
            cur = cur.next
        cur.next = l1 or l2
        return head.next
```

Array/MergeSortedArray.py

```
"""
Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one
sorted array.
Note:
The number of elements initialized in nums1 and nums2 are m and n respectively.
You may assume that nums1 has enough space (size that is greater or equal to m +
n) to hold additional elements from nums2.
Example:
Input:
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6],       n = 3
Output: [1,2,2,3,5,6]
合并两个排序过的整数数组。
要求在nums1中做修改。
测试用例：
https://leetcode.com/problems/merge-sorted-array/description/
第一版：
思路与merge two sorted list 一致...
passed 但是效率较低。
第二版：
直接以倒序的方式，从尾到头判断，这样不需要额外的空间，列表，效率也自然要高。
passed，但还不是最高的。
第三版：
通过的解法中有一种非常聪明的：
由于本身是在nums1中操作，基于 merge two sorted list 的思想中的：
如果有一方结束了那么就把另一方直接合并过去。
但我们是在nums1中做修改，所以可以修改下判断规则：
不再以整个m+n作为while的依据而是m和n都不为0。
若m为0,那么合并剩余的n到nums1中，
若n为0,则无需做任何动作，因为本来就是nums1.。
"""
class Solution(object):
    # 第一版：
    # def merge(self, nums1, m, nums2, n):
    #     """
    #     :type nums1: List[int]
    #     :type m: int
    #     :type nums2: List[int]
    #     :type n: int
    #     :rtype: void Do not return anything, modify nums1 in-place instead.
    #     """
    #     nums3 = nums1[:m]
    #     nums1_index = 0
    #     nums2_index = 0
    #     nums3_index = 0
    #     mn = m+n
    #     length_nums3 = len(nums3)
    #     length_nums2 = len(nums2)
    #     while nums1_index < mn:
    #         if nums3_index == length_nums3:
    #             for i in nums2[nums2_index:]:
    #                 nums1[nums1_index] = i
    #                 nums1_index += 1
    #             break
```

```
    #         if nums2_index == length_nums2:
    #             for i in nums3[nums3_index:]:
    #                 nums1[nums1_index] = i
    #                 nums1_index += 1
    #             break
    #         if nums3[nums3_index] < nums2[nums2_index]:
    #             nums1[nums1_index] = nums3[nums3_index]
    #             nums3_index += 1
    #         else:
    #             nums1[nums1_index] = nums2[nums2_index]
    #             nums2_index += 1
    #         nums1_index += 1
    # 第二版:
    # def merge(self, nums1, m, nums2, n):
    #     """
    #     :type nums1: List[int]
    #     :type m: int
    #     :type nums2: List[int]
    #     :type n: int
    #     :rtype: void Do not return anything, modify nums1 in-place instead.
    #     """
    #     mn = m + n - 1
    #     while mn >= 0:
    #         if m == 0:
    #             for i in nums2[:n][::-1]:
    #                 nums1[mn] = i
    #                 mn -= 1
    #             break
    #         if n == 0:
    #             for i in nums1[:m][::-1]:
    #                 nums1[mn] = i
    #                 mn -= 1
    #             break
    #         if nums1[m-1] > nums2[n-1]:
    #             nums1[mn] = nums1[m-1]
    #             m -= 1
    #         else:
    #             nums1[mn] = nums2[n-1]
    #             n -= 1
    #         mn -= 1
    # 第三版
    def merge(self, nums1, m, nums2, n):
        """
        :type nums1: List[int]
        :type m: int
        :type nums2: List[int]
        :type n: int
        :rtype: void Do not return anything, modify nums1 in-place instead.
        """
        mn = m + n - 1
        while m > 0 and n > 0:
            if nums1[m-1] > nums2[n-1]:
                nums1[mn] = nums1[m-1]
                m -= 1
            else:
                nums1[mn] = nums2[n-1]
                n -= 1
            mn -= 1
```

```
        if n > 0:
            nums1[:n] = nums2[:n]
```

Array/MinimumFallingPathSum.py

```
"""
Given a square array of integers A, we want the minimum sum of a falling path
through A.
A falling path starts at any element in the first row, and chooses one element
from each row.  The next row's choice must be in a column that is different from
the previous row's column by at most one.
Example 1:
Input: [[1,2,3],[4,5,6],[7,8,9]]
Output: 12
Explanation:
The possible falling paths are:
[1,4,7], [1,4,8], [1,5,7], [1,5,8], [1,5,9]
[2,4,7], [2,4,8], [2,5,7], [2,5,8], [2,5,9], [2,6,8], [2,6,9]
[3,5,7], [3,5,8], [3,5,9], [3,6,8], [3,6,9]
The falling path with the smallest sum is [1,4,7], so the answer is 12.
Note:
1. 1 <= A.length == A[0].length <= 100
2. -100 <= A[i][j] <= 100
从上到下，每下层元素可以选择位于它之上的 左中右 三个元素。
直接 Dp 思路：
从第二层开始，每个元素都选择位于它之上的三个元素中最小的一个元素。
最后输出最后一层中最小的元素即可。
测试地址：
https://leetcode.com/contest/weekly-contest-108/problems/minimum-falling-path-
sum/
"""
class Solution(object):
    def minFallingPathSum(self, A):
        """
        :type A: List[List[int]]
        :rtype: int
        """
        for y in range(1, len(A)):
            for x in range(len(A[0])):
                a = A[y-1][x-1] if x-1 >= 0 else float('inf')
                b = A[y-1][x+1] if x+1 < len(A[0]) else float('inf')
                A[y][x] += min(A[y-1][x], a, b)
        return min(A[-1])
```

Array/MinimumIndexSumOfTwoLists.py

```
"""
Suppose Andy and Doris want to choose a restaurant for dinner, and they both have
a list of favorite restaurants represented by strings.
You need to help them find out their common interest with the least list index
sum. If there is a choice tie between answers, output all of them with no order
requirement. You could assume there always exists an answer.
Example 1:
Input:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["Piatti", "The Grill at Torrey Pines", "Hungry Hunter Steakhouse", "Shogun"]
Output: ["Shogun"]
Explanation: The only restaurant they both like is "Shogun".
Example 2:
Input:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["KFC", "Shogun", "Burger King"]
Output: ["Shogun"]
Explanation: The restaurant they both like and have the least index sum is
"Shogun" with index sum 1 (0+1).
Note:
The length of both lists will be in the range of [1, 1000].
The length of strings in both lists will be in the range of [1, 30].
The index is starting from 0 to the list length minus 1.
No duplicates in both lists.
给两个列表，求加起来的索引是最小的重合部分，若有多个相同的则输出多个相同的，无顺序要求。
思路：
直接判断的话是 O(mn)，利用哈希表（字典）来减少查询时间。
可优化部分：
第二次不用哈希表，用两个变量，一个代表当前的最小值，一个代表所存数据，不断替换，追加。
测试用例：
https://leetcode.com/problems/minimum-index-sum-of-two-lists/description/
"""
class Solution(object):
    def findRestaurant(self, list1, list2):
        """
        :type list1: List[str]
        :type list2: List[str]
        :rtype: List[str]
        """
        list1_dict = {value:index for index, value in enumerate(list1)}
        commonInterest = {}
        for index, value in enumerate(list2):
            if list1_dict.get(value) is not None:
                try:
                    commonInterest[index + list1_dict.get(value)].append(value)
                except KeyError:
                    commonInterest[index + list1_dict.get(value)] = [value]
        return commonInterest[min(commonInterest)]
```

Array/MinimumPathSum.py

```
"""
Given a m x n grid filled with non-negative numbers, find a path from top left to
bottom right which minimizes the sum of all numbers along its path.
Note: You can only move either down or right at any point in time.
Example:
Input:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
Output: 7
Explanation: Because the path 1→3→1→1→1 minimizes the sum
给一个二维数组，里面全是非负整数，找到一条左上到右下花费最小的路线。
思路：
当前点只要加 up 和 left 中较小的一个即可。
效率 O(n)
为了判断边界直接顺着思路写了。优化的话可以先把最上面的一排按其左的数相加，最左边的一列按其上面的
数相加。
然后从 [1,1] 开始，这样不需要判断边界，写法上可以少点判断效率能提高10+ms。
测试地址：
https://leetcode.com/problems/minimum-path-sum/description/
"""
class Solution(object):
    def get_up_left(self, x, y):
        if y-1 < 0:
            up = False
        else:
            up = (x, y-1)
        if x-1 < 0:
            left = False
        else:
            left = (x-1,y)
        # up and left
        return (up, left)
    def minPathSum(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                xy = self.get_up_left(j, i)
                up = grid[xy[0][1]][xy[0][0]] if xy[0] else float('inf')
                left = grid[xy[1][1]][xy[1][0]] if xy[1] else float('inf')
                if up == float('inf') and left == float('inf'):
                    continue
                grid[i][j] = grid[i][j] + min(up, left)
        return grid[-1][-1]
```

Array/MissingNumber.py

```
"""
Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.
Example 1:
Input: [3,0,1]
Output: 2
Example 2:
Input: [9,6,4,2,3,5,7,0,1]
Output: 8
Note:
Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?
给出从 0 - n的数，找出其中缺少的那个。
思路：
一开始的思路只有 set ... set(n+1)，set(nums)，然后取差集。
这种方法固然可以通过测试，不过题目要求使用常数空间也就是 O(1)。
后来经过 Discuss 区里的点拨，发现几种有趣的方法：
1. 使用异或。 两个相同的数会相互抵消掉。也就是说，从0-n异或一遍。然后在用这个数把nums里的给异或一遍。好了剩下的就是那个缺少的了。
2. 使用和。思路与异或有异曲同工之妙，0-n加一遍。然后在减去，剩下的也是。
运用数学方法，很妙。
测试地址：
https://leetcode.com/problems/missing-number/description/
这里用了加。异或一样的。
beat 88% 28ms.
"""
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        # x = list(set(range(max(nums)+1)) - set(nums))
        # if x:
        #     return x.pop()
        # else:
        #     return max(nums)+1
        all_nums = sum(range(len(nums)+1))
        for i in nums:
            all_nums -= i
        return all_nums
```

Array/NumberOfIslands.py

```
"""
与今日头条的秋招第一题差不多的题：
Given a 2d grid map of '1's (land) and '0's (water), count the number of islands.
An island is surrounded by water and is formed by connecting adjacent lands
horizontally or vertically. You may assume all four edges of the grid are all
surrounded by water.
Example 1:
Input:
11110
11010
11000
00000
Output: 1
Example 2:
Input:
11000
11000
00100
00011
Output: 3
分析看 FootbalFans.py
passed.
测试地址：
https://leetcode.com/problems/number-of-islands/description/
"""
class Solution(object):
    def makeXY(self, x, y):
        return ((x, y-1),
                (x, y+1),
                (x-1, y),
                (x+1, y))
    def numIslands(self, court):
        """
        :type grid: List[List[str]]
        :rtype: int
        """
        fans_groups = []
        x = 0
        y = 0
        if not court:
            return 0
        x_length = len(court[0])
        y_length = len(court)
        def helper(x, y):
            Xy = self.makeXY(x, y)
            for i in Xy:
                try:
                    if i[1] < 0 or i[0] < 0:
                        continue
                    if court[i[1]][i[0]] == '1':
                        court[i[1]][i[0]] = '0'
                        t = helper(i[0], i[1])
                except IndexError:
                    continue
```

```
            else:
                return 1
        for y in range(y_length):
            for x in range(x_length):
                if court[y][x] == '1':
                    court[y][x] = '0'
                    fans_groups.append(helper(x, y))
        if not fans_groups:
            return 0
        return len(fans_groups)
```

Array/OddEvenLinkedList.py

```
"""
Given a singly linked list, group all odd nodes together followed by the even
nodes. Please note here we are talking about the node number and not the value in
the nodes.
You should try to do it in place. The program should run in O(1) space complexity
and O(nodes) time complexity.
Example 1:
Input: 1->2->3->4->5->NULL
Output: 1->3->5->2->4->NULL
Example 2:
Input: 2->1->3->5->6->4->7->NULL
Output: 2->3->6->7->1->5->4->NULL
Note:
The relative order inside both the even and odd groups should remain as it was in
the input.
The first node is considered odd, the second node even and so on ...
给一个单链表，将所有的单数节点和双数节点聚合在一块，双数节点跟在单数节点后面。
空间复杂度需要为 O(1) 时间则为 O(n)
思路：
1. 定义两个开头的节点，odd 和 even，用于区分单双。
odd.next 和 even.next 都同时为 .next.next。
不断替换，直到最后，最后不要忘了把 even 连在 odd 后面。
beat:
100%
测试地址：
https://leetcode.com/problems/odd-even-linked-list/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def oddEvenList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head:
            return head
        odd = head
        even = head.next
        even_head = even
        while odd.next and even.next:
            odd.next = odd.next.next
            even.next = even.next.next
            odd = odd.next
            even = even.next
        odd.next = even_head
        return head
```

Array/PartitionArrayIntoDisjointIntervals.py

```
"""
Given an array A, partition it into two (contiguous) subarrays left and right so
that:
Every element in left is less than or equal to every element in right.
left and right are non-empty.
left has the smallest possible size.
Return the length of left after such a partitioning.  It is guaranteed that such
a partitioning exists.
Example 1:
Input: [5,0,3,8,6]
Output: 3
Explanation: left = [5,0,3], right = [8,6]
Example 2:
Input: [1,1,1,0,6,12]
Output: 4
Explanation: left = [1,1,1,0], right = [6,12]
Note:
2 <= A.length <= 30000
0 <= A[i] <= 10^6
It is guaranteed there is at least one way to partition A as described.
给定一个数组，分成两个部分，left 和 right。
left 有以下三个特征：
1. left 中的每一个元素都小于或等于 right 中的每一个元素。
2. left 和 right 是非空。
3. left 尽量小。
那么只要 left 中的最大值比 right 中的最小值要小即可。
[5,0,3,8,6]
 5  0 3 8 6
 5 0   3 8 6
 ......
那么只要从左向右过一遍，取最大。
在从右向左过一遍取最小。
         [5,0,3,8,6]
max  ->   5 5 5 8 8
min  <-   0 0 3 6 6
对比时是  max 中的 [i] 与 min中的 i+1对比若 max[i] < min[i+1] 即找到第一个 left 与
right 的分界点了。
第一个分界点即为left最少的一个点。
测试地址：
https://leetcode.com/problems/partition-array-into-disjoint-
intervals/description/
beat 66.67%.
"""
class Solution(object):
    def partitionDisjoint(self, A):
        """
        :type A: List[int]
        :rtype: int
        """
        maxes = [A[0]]
        mines = [A[-1]]
        for i in range(1, len(A)):
            if A[i] > maxes[i-1]:
                maxes.append(A[i])
```

```
            else:
                maxes.append(maxes[i-1])
        A = A[::-1]
        for i in range(1, len(A)):
            if A[i] < mines[i-1]:
                mines.append(A[i])
            else:
                mines.append(mines[i-1])
        mines = mines[::-1]
        for i in range(len(mines)-1):
            if maxes[i] <= mines[i+1]:
                return i+1
```

Array/PartitionList.py

```
"""
Given a linked list and a value x, partition it such that all nodes less than x
come before nodes greater than or equal to x.
You should preserve the original relative order of the nodes in each of the two
partitions.
Example:
Input: head = 1->4->3->2->5->2, x = 3
Output: 1->2->2->4->3->5
无脑怼。
beat:
99%.
测试地址:
https://leetcode.com/problems/partition-list/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def partition(self, head, x):
        """
        :type head: ListNode
        :type x: int
        :rtype: ListNode
        """
        all_nodes = []
        while head:
            all_nodes.append(head.val)
            head = head.next
        less_nodes = []
        greater_nodes = []
        for i in all_nodes:
            if i < x:
                less_nodes.append(i)
            else:
                greater_nodes.append(i)
        if less_nodes:
            less_head = ListNode(less_nodes[0])
        head = less_head if less_nodes else None
        for i in less_nodes[1:]:
            less_head.next = ListNode(i)
            less_head = less_head.next
        if greater_nodes:
            greater_head = ListNode(greater_nodes[0])
        _head = greater_head if greater_nodes else None
        for i in greater_nodes[1:]:
            greater_head.next = ListNode(i)
            greater_head = greater_head.next
        if head:
            less_head.next = _head
            return head
        return _head
```

Array/Pascal's TriangleI_II.py

```
"""
Given a non-negative integer numRows, generate the first numRows of Pascal's
triangle.
In Pascal's triangle, each number is the sum of the two numbers directly above
it.
Example:
Input: 5
Output:
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
帕斯卡三角。
图看链接：
https://leetcode.com/problems/pascals-triangle/description/
顺着思路来即可。
I
测试地址：
https://leetcode.com/problems/pascals-triangle/description/
II
测试地址：
https://leetcode.com/problems/pascals-triangle-ii/description/
II 中的要求是返回最后一个，且空间为 O(K)。下面这个空间不是 O(k) 不过只要修改一下即可。
"""
# I
class Solution(object):
    def generate(self, numRows):
        """
        :type numRows: int
        :rtype: List[List[int]]
        """
        result = []
        for i in range(1, numRows+1):
            x = [0 for j in range(i)]
            x[0] = 1
            x[-1] = 1
            for j in range(1, i-1):
                x[j] = result[-1][j] + result[-1][j-1]
            result.append(x)
        return result
# II
 class Solution(object):
    def generate(self, numRows):
        """
        :type numRows: int
        :rtype: List[List[int]]
        """
        result = []
        for i in range(1, numRows+2):
            x = [0 for j in range(i)]
            x[0] = 1
```

```
            x[-1] = 1
            for j in range(1, i-1):
                x[j] = result[-1][j] + result[-1][j-1]
            result.append(x)
        return result[-1]
```

Array/PlusOne.py

```
"""
Given a non-empty array of digits representing a non-negative integer, plus one
to the integer.
The digits are stored such that the most significant digit is at the head of the
list, and each element in the array contain a single digit.
You may assume the integer does not contain any leading zero, except the number 0
itself.
Example 1:
Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Example 2:
Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
整数列表 -> 字符串 -> 整数 -> +1 ->字符串 -> 整数列表。
有点无脑，效率也可。
beat 63% 24ms.
测试地址：
https://leetcode.com/problems/plus-one/description/
"""
class Solution(object):
    def plusOne(self, digits):
        """
        :type digits: List[int]
        :rtype: List[int]
        """
        return list(map(int, list(str(int(''.join(map(str, digits)))+1))))
```

Array/ProductOfArrayExceptSelf.py

```
"""
Given an array nums of n integers where n > 1,  return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].
Example:
Input:  [1,2,3,4]
Output: [24,12,8,6]
Note: Please solve it without division and in O(n).
Follow up:
Could you solve it with constant space complexity? (The output array does not count as extra space for the purpose of space complexity analysis.)
给一个数组，返回数组中除了这一位的数其他数的乘积。
不能用除的，且时间复杂度需要为 O（n）。
进阶条件 空间复杂度为常数级别。
一刷进阶条件没达成。
基本思路是，一左一右两个存储位置的新列表。
从左到右过一遍，从右到左过一遍。算出每个数左边和右边的乘积。
最后输出每个位置的左*右。
效率是 O(n)，没用除。但空间是 O(2n)。
beat 57% ~ 80% (在 2 中需要把 range 变为 xrange).
前面那几个的思路也是同样的。多测几次应该也能beat 100%...
哎哎？突然想到，output 的数组不算在额外空间里的话，
可以直接把output数组作为 left。然后right的时候直接替换就好了啊。
没错，这样就是 O(1) 了，进阶达成。
beat 98%。
测试链接：
https://leetcode.com/problems/product-of-array-except-self/description/
"""
class Solution(object):
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        output = [1]
        for i in range(len(nums)-1):
            output.append(output[-1] * nums[i])
        right = 1
        for i in range(len(nums)-1, -1, -1):
            output[i] = right * output[i]
            right *= nums[i]
        return output
        # left = [1]
        # right = [1]
        # for i in range(len(nums)-1):
        #     left.append(left[-1] * nums[i])
        # for i in range(len(nums)-1, 0, -1):
        #     right.append(right[-1] * nums[i])
        # length = len(left)
        # return [left[i] * right[length-1-i] for i in range(length)]
```

Array/RemoveDuplicatesFromSortedArray.py

```
"""
Given a sorted array nums, remove the duplicates in-place such that each element
appear only once and return the new length.
Do not allocate extra space for another array, you must do this by modifying the
input array in-place with O(1) extra memory.
Example 1:
Given nums = [1,1,2],
Your function should return length = 2, with the first two elements of nums being
1 and 2 respectively.
It doesn't matter what you leave beyond the returned length.
Example 2:
Given nums = [0,0,1,1,1,2,2,3,3,4],
Your function should return length = 5, with the first five elements of nums
being modified to 0, 1, 2, 3, and 4 respectively.
It doesn't matter what values are set beyond the returned length.
Clarification:
Confused why the returned value is an integer but your answer is an array?
Note that the input array is passed in by reference, which means modification to
the input array will be known to the caller as well.
Internally you can think of this:
// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);
// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
删除给定数组的重复的数字，在原数组上操作。
这个从后向前，判断是否重复，若重复将下标加入要删除的列表中。
最后迭代这个要删除的列表，然后将所有下标删除。因为是从后向前所以不会打乱顺序。
beat 56%
测试地址：
https://leetcode.com/problems/remove-duplicates-from-sorted-array/description/
"""
class Solution(object):
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return
        length = len(nums)-1
        one = nums[length]
        remove_index = []
        for i in range(length-1, -1, -1):
            if one == nums[i]:
                remove_index.append(i)
                continue
            one = nums[i]
        for i in remove_index:
            nums.pop(i)
```

Array/RemoveDuplicatesFromSortedArrayII.py

```
"""
Given a sorted array nums, remove the duplicates in-place such that duplicates
appeared at most twice and return the new length.
Do not allocate extra space for another array, you must do this by modifying the
input array in-place with O(1) extra memory.
Example 1:
Given nums = [1,1,1,2,2,3],
Your function should return length = 5, with the first five elements of nums
being 1, 1, 2, 2 and 3 respectively.
It doesn't matter what you leave beyond the returned length.
Example 2:
Given nums = [0,0,1,1,1,1,2,3,3],
Your function should return length = 7, with the first seven elements of nums
being modified to 0, 0, 1, 1, 2, 3 and 3 respectively.
It doesn't matter what values are set beyond the returned length.
Clarification:
Confused why the returned value is an integer but your answer is an array?
Note that the input array is passed in by reference, which means modification to
the input array will be known to the caller as well.
Internally you can think of this:
// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);
// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
至多出现两次，思路上没有变化，顺手一写。
beat 91%
测试地址：
https://leetcode.com/problems/remove-duplicates-from-sorted-array-
ii/description/
"""
class Solution(object):
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return
        length = len(nums) - 1
        one = nums[length]
        two = 1
        remove_index = []
        for i in range(length-1, -1, -1):
            if one == nums[i]:
                if two == 2:
                    remove_index.append(i)
                else:
                    two += 1
            else:
                one = nums[i]
                two = 1
```

```
        for i in remove_index:
            nums.pop(i)
```

Array/RemoveDuplicatesFromSortedList.py

```
"""
Given a sorted linked list, delete all duplicates such that each element appear
only once.
Example 1:
Input: 1->1->2
Output: 1->2
Example 2:
Input: 1->1->2->3->3
Output: 1->2->3
删除链表中重复的节点。
一直迭代，若当前 val == next.val，则把 next 与 next.next 相连。
beat
100%
测试地址：
https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head:
            return head
        x = head
        while head.next:
            if head.val == head.next.val:
                head.next = head.next.next
            else:
                head = head.next
        return x
```

Array/RemoveDuplicatesFromSortedListII.py

```
"""
Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.
Example 1:
Input: 1->2->3->3->4->4->5
Output: 1->2->5
Example 2:
Input: 1->1->1->2->3
Output: 2->3
这次要把全部的重复都删除。
我的思路是利用标记，过一遍，先把重复的删到剩一个，然后把剩下的一个标记为重复。
然后做一个新的链表。
beat 72%
测试地址：
https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head:
            return head
        x = head
        while head.next:
            if head.val == head.next.val:
                head.next = head.next.next
                head.duplicate = True
            else:
                head = head.next
        while x:
            if hasattr(x, 'duplicate'):
                x = x.next
            else:
                break
        if not x:
            return None
        new = ListNode(x.val)
        x = x.next
        _new = new
        while x:
            if hasattr(x, 'duplicate'):
                x = x.next
            else:
                new.next = ListNode(x.val)
                new = new.next
                x = x.next
        return _new
```

Array/RemoveLinkedListElements.py

```
"""
Remove all elements from a linked list of integers that have value val.
Example:
Input:  1->2->6->3->4->5->6, val = 6
Output: 1->2->3->4->5
删除所有的val。
注意下开头即为 val 的情况。
beat
90%
测试地址：
https://leetcode.com/problems/remove-linked-list-elements/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def removeElements(self, head, val):
        """
        :type head: ListNode
        :type val: int
        :rtype: ListNode
        """
        while head:
            if head.val == val:
                head = head.next
            else:
                break
        _head = head
        if not _head:
            return None
        while head and head.next:
            if head.next.val == val:
                head.next = head.next.next
            else:
                head = head.next
        return _head
```

Array/RemoveNthNodeFromEndOfList.py

```
"""
Given a linked list, remove the n-th node from the end of list and return its
head.
Example:
Given linked list: 1->2->3->4->5, and n = 2.
After removing the second node from the end, the linked list becomes 1->2->3->5.
Note:
Given n will always be valid.
Follow up:
Could you do this in one pass?
给一个链表和一个n，删除从后数第n个节点。
n总是有效的。
进阶条件是一次性完成。
一开始的想法是总得先知道长度，然后才能倒数第n个吧。
所以一开始用的列表，Python 中的列表非常适合做这个操作。
O(n) 遍历，然后剩下的索引操作就是 O(1) 了。
失误的是，考虑到了如果删除的是head，但是写的话没写成 head.next，写成了 list_node[1]，
这样在链表中只有一个节点的时候就出错了...
本题算是失败了。2 pass.
效率上是 28ms。
看了25ms的写法，感觉非常聪明。
以前总是想，这样的必须得先过一遍知道长度才能做其他的事吧。
这个的写法是，用一条像是绳子一样的。
|----------|
slow     fast
让fast走n步。
然后fast和slow一起走，等fast.next是None，也就是到头了。那么slow就是要删除的点的前一个了。
直接把slow.next与slow.next.next结合就达标了。
如果走了n步后fast直接是None了。那么说明删除的节点是head，那么返回 head.next就好了。
不过这个提交了两次也是 28ms..
但是这个思路是真的棒。
关键词:
绳子。
测试地址:
https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def removeNthFromEnd(self, head, n):
        """
        :type head: ListNode
        :type n: int
        :rtype: ListNode
        """
        my_head = my_trail = head
        for i in range(n):
            my_head = my_head.next
        if not my_head:
            return head.next
        while my_head.next:
```

```
            my_head = my_head.next
            my_trail = my_trail.next
        my_trail.next = my_trail.next.next
        return head
#         list_node = []
#         my_head = head
#         while my_head:
#             list_node.append(my_head)
#             my_head = my_head.next
#         if n == len(list_node):
#             try:
#                 return list_node[1]
#             except:
#                 return None
#         if n == 1:
#             list_node[-2].next = None
#             return list_node[0]
#         list_node[-(n+1)].next = list_node[-(n-1)]
#         return list_node[0]
```

Array/RotateList.py

```
"""
Given a linked list, rotate the list to the right by k places, where k is non-
negative.
Example 1:
Input: 1->2->3->4->5->NULL, k = 2
Output: 4->5->1->2->3->NULL
Explanation:
rotate 1 steps to the right: 5->1->2->3->4->NULL
rotate 2 steps to the right: 4->5->1->2->3->NULL
Example 2:
Input: 0->1->2->NULL, k = 4
Output: 2->0->1->NULL
Explanation:
rotate 1 steps to the right: 2->0->1->NULL
rotate 2 steps to the right: 1->2->0->NULL
rotate 3 steps to the right: 0->1->2->NULL
rotate 4 steps to the right: 2->0->1->NULL
旋转链表。 k 非负。
k 超过链表的最大长度也可。
思路:
过一遍链表长度，k % length 取个模，防止 k 超级大。
之后 slow fast 两个，fast 先走 k 个，然后 slow 与 fast 同时走，走到最后 slow.next 即为从
后到前 k 个的起点。
剩下的就是把原来的尾置换到前。
下面这个可以优化下，不过就测试来说已经可以了。
beat 100%
24ms ~ 36ms
测试地址:
https://leetcode.com/problems/rotate-list/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def rotateRight(self, head, k):
        """
        :type head: ListNode
        :type k: int
        :rtype: ListNode
        """
        if not k or not head:
            return head
        def getLength(node):
            length = 0
            while node:
                node = node.next
                length += 1
            return length
        length = getLength(head)
        k = k % length
        slow = head
        fast = head
```

```
        while k > 0:
            fast = fast.next
            k -= 1
        while fast.next:
            slow = slow.next
            fast = fast.next
        rotate_head = slow.next
        if not rotate_head:
            return head
        slow.next = None
        _rotate_head = rotate_head
        while _rotate_head.next:
            _rotate_head = _rotate_head.next
        _rotate_head.next = head
        return rotate_head
```

Array/Search2DMatrix.py

```
"""
Write an efficient algorithm that searches for a value in an m x n matrix. This
matrix has the following properties:
Integers in each row are sorted from left to right.
The first integer of each row is greater than the last integer of the previous
row.
Example 1:
Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 3
Output: true
Example 2:
Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 13
Output: false
给一个2d数组，给一个数字，找是否存在于其中。
思路：
两个二分法：
一个竖着的二分法，
一个横着的二分法。
竖着的二分法返回下标，横着的返回是否存在。
竖着的：
处理仅有一个的情况：
<= 都返回len()-1也就是下标
> 返回 -1 表示不存在。
其他情况下寻找 left <= x <= right.
由于 right一定大于left。所以left <= 可以放在一起判断，返回的是基础下标+当前二分的len()-1。
right == 的情况则不-1。
横着的没啥好说的，普通二分法即可。
最差是O(logm + logn)两次二分法的耗时。
beat 100%.
测试用例：
https://leetcode.com/problems/search-a-2d-matrix/description/
"""
class Solution(object):
    def binarySearch(self, rawList, target, index=0):
        if target >= rawList[-1]:
            return len(rawList) - 1
        if target < rawList[0]:
            return -1
        split = len(rawList) // 2
        leftList = rawList[:split]
        rightList = rawList[split:]
        if leftList[-1] <= target and rightList[0] > target:
            return len(leftList) + index - 1
```

```
        if rightList[0] == target:
            return len(leftList) + index
        if leftList[-1] > target:
            return self.binarySearch(leftList, target, index=index)
        if rightList[0] < target:
            return self.binarySearch(rightList, target,
index=index+len(leftList))
    def binarySearch2(self, rawList, target):
        split = len(rawList) // 2
        left = rawList[:split]
        right = rawList[split:]
        if not left and not right:
            return False
        if left and left[-1] == target:
            return True
        if right and right[0] == target:
            return True
        if len(left) > 1 and left[-1] > target:
            return self.binarySearch2(left, target)
        if len(right) > 1 and right[0] < target:
            return self.binarySearch2(right, target)
        return False
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        if not any(matrix):
            return False
        column = [i[0] for i in matrix]
        column_result = self.binarySearch(column, target)
        if column_result == -1:
            return False
        return self.binarySearch2(matrix[column_result], target)
```

Array/Search2DMatrixII.py

```
"""
Write an efficient algorithm that searches for a value in an m x n matrix. This
matrix has the following properties:
Integers in each row are sorted in ascending from left to right.
Integers in each column are sorted in ascending from top to bottom.
Example:
Consider the following matrix:
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
Given target = 5, return true.
Given target = 20, return false.
思路：
已知条件：
每一行以及每一列都是升序排列的。
左上角是整个里面最小的。
右下角是整个里面最大的。
右上角是整行中最大的，但是是整列中最小的。
左下角是整列里最大的，整行里最小的。
1. 直接遍历，O(klogn)，每一行都用二分法，有些无脑，但有效。
passed，不过效率低，只beat了 7% 左右。
2. 基于上面的分析，应该是可以减少一些不必要的搜索的。
左下角是整列里最大的，如果target小于它则不必搜索这一整行，用二分法搜索这一列。
然后脱去这层
[
  [1,     4,  7, 11, 15],
  [2,     5,  8, 12, 19],
  [3,     6,  9, 16, 22],
  [10,   13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
此时同样的思路搜索13，若target大于则不必搜索这一列，直接搜这一行即可。
[
  [1,     4,     7, 11, 15],
  [2,     5,     8, 12, 19],
  [3,     6,     9, 16, 22],
  [10,   13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
直到搜索到右上角。
这样的效率也是 O(nlogn).
不出所料，比上个稍微好了那么一点点....
3. 基于上面的分析：
可以不用搜索，直接点对点。
搜了18，这个点，如果target比它大，那么就在这一行继续走，走到21，如果比它大那就继续在这一行走，
到了23，
如果比它小就搜这一列，这样搜到row或column的尽头，有就是有，没有就是没有。
效率最差是 O(m+n)。
Good, beat 82%.
测试用例：
```

```
https://leetcode.com/problems/search-a-2d-matrix-ii/description/
"""
class Solution(object):
    def binarySearch2(self, rawList, target):
        split = len(rawList) // 2
        left = rawList[:split]
        right = rawList[split:]
        if not left and not right:
            return False
        if left and left[-1] == target:
            return True
        if right and right[0] == target:
            return True
        if len(left) > 1 and left[-1] > target:
            return self.binarySearch2(left, target)
        if len(right) > 1 and right[0] < target:
            return self.binarySearch2(right, target)
        return False
    def searchMatrix(self, matrix, target):
        if not any(matrix):
            return False
        # column + row -
        column, row = 0, len(matrix) - 1
        column_length = len(matrix[0])
        while row >= 0 and column < column_length:
            if matrix[row][column] == target:
                return True
            if matrix[row][column] > target:
                row -= 1
            else:
                column += 1
        return False
    # def searchMatrix(self, matrix, target):
    #     """
    #     :type matrix: List[List[int]]
    #     :type target: int
    #     :rtype: bool
    #     """
    #     if not any(matrix):
    #         return False
    #     # column + row -
    #     column, row = 0, len(matrix) - 1
    #     column_length = len(matrix[0])
    #     while row >= 0 and column < column_length:
    #         """
    #             left bottom to right top.
    #         """
    #         if matrix[row][column] == target:
    #             return True
    #         # search column.
    #         if matrix[row][column] > target:
    #             if self.binarySearch2([i[column] for i in matrix[:row]],
target):
    #                 return True
    #         # search row.
    #         else:
    #             if self.binarySearch2(matrix[row][column+1:], target):
    #                 return True
```

```
    #         row -= 1
    #         column += 1
    #     return False
    # def searchMatrix(self, matrix, target):
    #     """
    #     :type matrix: List[List[int]]
    #     :type target: int
    #     :rtype: bool
    #     """
    #     if not any(matrix):
    #         return False
    #     for i in matrix:
    #         if i[0] == target:
    #             return True
    #         if i[0] < target:
    #             if self.binarySearch2(i, target):
    #                 return True
    #     return False
```

Array/SearchInRotatedSortedArray.py

```
"""
Suppose an array sorted in ascending order is rotated at some pivot unknown to
you beforehand.
(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).
You are given a target value to search. If found in the array return its index,
otherwise return -1.
You may assume no duplicate exists in the array.
Your algorithm's runtime complexity must be in the order of O(log n).
Example 1:
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
Example 2:
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
一个排过序的数组被旋转了一下，但不知道在哪个位置旋转的。
给定这样一个数组，并且给定一个目标，找到此数组中是否存在目标。
时间复杂度必须是 O（log n）。
排序过的找某数第一个想法就是 二分法。
这个二分法的关键点在于应该分成两个部分：
1. 递增的部分。
2. 第二个递增的部分。
---
1. 要找的旋转点可以利用 已排序 这个关键词：
由于是已经排过序的所以若是有旋转的 nums[0] 一定是大于所有旋转的元素的。
所以首先的二分是根据以 nums[0] 为 target 找到收个比它小的位置。
代码在 find_rotate 函数。
因为target是 nums[0] lo 就不管 0 了，直接从1开始。
  lo        hi
4 5 6 7 0 1 2
若 mid 大于 nums[0] 因为是排过序的，而且旋转的情况的话上面也分析过了，不可能有大于 nums[0]
的存在，所以mid左边的不可能存在旋转点。
nums[0] < nums[mid]
lo = mid + 1
否则
hi = mid
这样一轮过后，要么找到旋转点，lo确定后 hi 一直缩小。
              要么无旋转点，lo一直增大。
2. 对于两个递增的部分，分别进行一次普通二分即可。
beat 100%
20ms
测试地址：
https://leetcode.com/problems/search-in-rotated-sorted-array/description/
"""
class Solution(object):
    def find_rotate(self, nums):
        target = nums[0]
        lo = 1
        hi = len(nums)
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] > target:
                lo = mid + 1
            else:
                hi = mid
```

```
        return lo
    def bi_search(self, nums, target, lo, hi):
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] == target:
                return mid
            if nums[mid] > target:
                hi = mid
            else:
                lo = mid + 1
        return -1
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if not nums:
            return -1
        rotate_index = self.find_rotate(nums)
        lo = 0
        hi = rotate_index
        # print(hi)
        one = self.bi_search(nums, target, lo, hi)
        if one != -1:
            return one
        two = self.bi_search(nums, target, hi, len(nums))
        if two != -1:
            return two
        return -1
```

Array/SearchInRotatedSortedArrayII.py

```
"""
Suppose an array sorted in ascending order is rotated at some pivot unknown to
you beforehand.
(i.e., [0,0,1,2,2,5,6] might become [2,5,6,0,0,1,2]).
You are given a target value to search. If found in the array return true,
otherwise return false.
Example 1:
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
Example 2:
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
Follow up:
This is a follow up problem to Search in Rotated Sorted Array, where nums may
contain duplicates.
Would this affect the run-time complexity? How and why?
search in rotated sorted array 变形版本：
与之前的区别是有重复，重复的值会造成什么样的影响呢？
1, 1, 1, 1, 1, 1, 2, 1, 1
在这个例子中
旋转的点是 1，如果再用之前的方法，nums[0] 是没办法保证比旋转的点之后的元素都大的，
循环之后的mid是1，不大于nums[0] 不大于 mid 的情况中会把hi变为mid这样就会造成之后的数据丢失。
其他的情况可以与 I 一样。
我想到的解决方法：
1. 可以把与 nums[0] 相同的一直吞并。 这里用的方法是一直迭代。
总时间复杂度的话最差会有 O(n + log n)。
测试地址：
https://leetcode.com/problems/search-in-rotated-sorted-array-ii/description/
beat 89% 24ms
平均24 - 28 ms。
"""
class Solution(object):
    def find_rotate(self, nums):
        target = nums[0]
        lo = 1
        for i in range(1, len(nums)):
            if nums[i] == target:
                lo += 1
            else:
                break
        hi = len(nums)
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] > target:
                lo = mid + 1
            else:
                hi = mid
        return lo
    def bi_search(self, nums, target, lo, hi):
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] == target:
                return mid
            if nums[mid] > target:
```

```
                hi = mid
            else:
                lo = mid + 1
        return -1
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if not nums:
            return False
        rotate_index = self.find_rotate(nums)
        lo = 0
        hi = rotate_index
        one = self.bi_search(nums, target, lo, hi)
        if one != -1:
            return True
        two = self.bi_search(nums, target, hi, len(nums))
        if two != -1:
            return True
        return False
```

Array/SetMatrixZeroes.py

```
"""
Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do
it in-place.
Example 1:
Input:
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
Output:
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
Example 2:
Input:
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
Output:
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
Follow up:
1. A straight forward solution using O(mn) space is probably a bad idea.
2. A simple improvement uses O(m + n) space, but still not the best solution.
3. Could you devise a constant space solution?
若某点为 0 则此行与此列均设为0。
关键字：
1. 坐标去重。
2. 坐标标记。
3. 利用原来的空间进行标记。
1. 先说去重，去重是我想到的第一种思路：
利用set的O(1) 特性去去重，将坐标转换为字符串，然后在转换回去。
这个的效率正如 进阶的 1 所说，O(mn) 空间，bad idea.
能 beat 20% 左右。
2. 坐标标记
   x     x
x [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
只需要标记 x 和 y一整行即可，也正如进阶 2 所说，空间 O(m+n)，good but not best.
用这个已经可以beat 94%
3. 主要在于四角，稍后在更。
测试地址：
https://leetcode.com/problems/set-matrix-zeroes/description/
"""
class Solution(object):
```

```python
    def setZeroes(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        # x
        raw_length = len(matrix[0])
        # y
        column_length = len(matrix)
        # O(mn) 第一版.
        # {"12", "23", "34"}
#         zero_xy = set()
#         def makeXY(x, y):
#             xy = set()
#             x = str(x)
#             y = str(y)
#             for i in range(raw_length):
#                 xy.add(','.join([str(i), y]))
#             for i in range(column_length):
#                 xy.add(','.join([x, str(i)]))
#             return xy
#         for y in range(column_length):
#             for x in range(raw_length):
#                 if matrix[y][x] == 0:
#                     zero_xy.update(makeXY(x, y))
#         for i in zero_xy:
#             t = i.split(',')
#             matrix[int(t[1])][int(t[0])] = 0
#         第二版 O(M+N)
#         raw = []
#         column = []
#         for y in range(column_length):
#             for x in range(raw_length):
#                 if matrix[y][x] == 0:
#                     raw.append(x)
#                     column.append(y)
#         for x in raw:
#             for y in range(column_length):
#                 matrix[y][x] = 0
#         for y in column:
#             for x in range(raw_length):
#                 matrix[y][x] = 0
```

Array/ShuffleAnArray.py

```
"""
Shuffle a set of numbers without duplicates.
Example:
// Init an array with set 1, 2, and 3.
int[] nums = {1,2,3};
Solution solution = new Solution(nums);
// Shuffle the array [1,2,3] and return its result. Any permutation of [1,2,3]
must equally likely to be returned.
solution.shuffle();
// Resets the array back to its original configuration [1,2,3].
solution.reset();
// Returns the random shuffling of array [1,2,3].
solution.shuffle();
将一个数组洗牌。
直接用 random.sample 即可。
sample 中的算法直接用了 set() 以及无限从数组下标中选择然后判断是否在set中...有点无脑。
在后面的例子中看到一种非常有趣的：
sorted 可以指定key，把这个key指定为 random.random()即可。
测试地址：
https://leetcode.com/problems/shuffle-an-array/description/
平均 beat 30% 左右。
用 sample 完全取决于运气，因为底层的源码就是取决于运气的...
"""
import random
class Solution(object):
    def __init__(self, nums):
        """
        :type nums: List[int]
        """
        self.nums = nums
    def reset(self):
        """
        Resets the array to its original configuration and return it.
        :rtype: List[int]
        """
        return self.nums
    def shuffle(self):
        """
        Returns a random shuffling of the array.
        :rtype: List[int]
        """
        return random.sample(self.nums, len(self.nums))
# Your Solution object will be instantiated and called as such:
# obj = Solution(nums)
# param_1 = obj.reset()
# param_2 = obj.shuffle()
```

Array/SingleNumber.py

```
"""
Given a non-empty array of integers, every element appears twice except for one.
Find that single one.
Note:
Your algorithm should have a linear runtime complexity. Could you implement it
without using extra memory?
Example 1:
Input: [2,2,1]
Output: 1
Example 2:
Input: [4,1,2,1,2]
Output: 4
给定一个非空数组，除了一个元素外，其余均出现两次。找出它。
需要在线性时间内，且不用额外空间。
用到了 missing number 的思路，利用异或的性质，相同的异或会抵消掉。
直接在原数组上操作，用了 i 变量，一个变量都不用要怎么写？
Discuss里也没找到相关的。
测试地址：
https://leetcode.com/problems/single-number/description/
"""
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        for i in range(1, len(nums)):
            nums[i] = nums[i] ^ nums[i-1]
        return nums[-1]
```

Array/SlidingWindowMaximum.py

```
"""
Given an array nums, there is a sliding window of size k which is moving from the
very left of the array to the very right. You can only see the k numbers in the
window. Each time the sliding window moves right by one position. Return the max
sliding window.
Example:
Input: nums = [1,3,-1,-3,5,3,6,7], and k = 3
Output: [3,3,5,5,6,7]
Explanation:
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
Note:
You may assume k is always valid, 1 ≤ k ≤ input array's size for non-empty
array.
Follow up:
Could you solve it in linear time?
这个我的思路是：
1. 先把前 k 个数取出来，然后排序一组，不排序一组。
2. 排序的一组作为查找使用。 不排序的一组作为删除增加会用。
3. 这里也可以使用堆代替排序，红黑树应该最好不过了。
4. 这里使用排序过的列表是为了能够使用二分法，从而达到 log n 级别的查找和后续添加。
   但同时因为即使在 log n级别查找到要添加删除的位置，进行列表的添加和删除仍然是一个 O(n) 级别
的事情...
   所以使用堆或者红黑树是最好的，添加和删除都是 log n 级别的。
5. sorted list 主要是进行获取最大与删除冗余，这里使用二分法来删除冗余。
6. unsorted list 用于知道要删除和添加的都是哪一个。
beat 31% 176ms.
测试地址：
https://leetcode.com/problems/sliding-window-maximum/description/
"""
from collections import deque
import bisect
class Solution(object):
    def find_bi(self, nums, target):
        lo = 0
        hi = len(nums)
        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] == target:
                return mid
            if nums[mid] < target:
                lo = mid + 1
            else:
                hi = mid
    def maxSlidingWindow(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
```

```
        :rtype: List[int]
        """
        if not nums:
            return []
        x = nums[:k]
        y = sorted(x)
        x = deque(x)
        maxes = max(x)
        result = [maxes]
        for i in nums[k:]:
            pop = x.popleft()
            x.append(i)
            index = self.find_bi(y, pop)
            y.pop(index)
            bisect.insort_left(y, i)
            result.append(y[-1])
        return result
```

Array/SortArrayByParity.py

```
"""
Given an array A of non-negative integers, return an array consisting of all the
even elements of A, followed by all the odd elements of A.
You may return any answer array that satisfies this condition.
Example 1:
Input: [3,1,2,4]
Output: [2,4,3,1]
The outputs [4,2,3,1], [2,4,1,3], and [4,2,1,3] would also be accepted.
Note:
1 <= A.length <= 5000
0 <= A[i] <= 5000
开胃菜，偶数放一堆，奇数放一堆。
O(n).
测试链接：
https://leetcode.com/contest/weekly-contest-102/problems/sort-array-by-parity/
contest.
"""
class Solution(object):
    def sortArrayByParity(self, A):
        """
        :type A: List[int]
        :rtype: List[int]
        """
        even = []
        odd = []
        for i in A:
            if i % 2 == 0:
                even.append(i)
            else:
                odd.append(i)
        return even + odd
```

Array/SortArrayByParityII.py

```
"""
Given an array A of non-negative integers, half of the integers in A are odd, and half of the integers are even.
Sort the array so that whenever A[i] is odd, i is odd; and whenever A[i] is even, i is even.
You may return any answer array that satisfies this condition.
Example 1:
Input: [4,2,5,7]
Output: [4,5,2,7]
Explanation: [4,7,2,5], [2,5,4,7], [2,7,4,5] would also have been accepted.
Note:
2 <= A.length <= 20000
A.length % 2 == 0
0 <= A[i] <= 1000
根据单双排列，一个双一个单。
"""
class Solution(object):
    def sortArrayByParityII(self, A):
        """
        :type A: List[int]
        :rtype: List[int]
        """
        odd = []
        even = []
        for i in A:
            if i%2 == 0:
                even.append(i)
            else:
                odd.append(i)
        result = []
        for j, k in zip(even, odd):
            result.append(j)
            result.append(k)
        return result
```

Array/SortColors.py

```
"""
Given an array with n objects colored red, white or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.
Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.
Note: You are not suppose to use the library's sort function for this problem.
Example:
Input: [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
Follow up:
A rather straight forward solution is a two-pass algorithm using counting sort.
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.
Could you come up with a one-pass algorithm using only constant space?
给一个放置了颜色的数组，里面包含三种颜色分别为 红 白 蓝，用 0 1 2 表示，它们在数组中是无序的，现在要把它们进行排序。
要求是：
原数组排序。
进阶条件是：
1. one-pass.
2. 使用常数空间。O（1）空间。
直接.sort排序是作弊行为。
进阶条件里给出一个直接的方法：
过一遍，分别记录出 0 1 2 的个数，然后按个数将它们分别替换。
这样做虽然是 O(1) 空间，不过不是 one-pass。
自己的思路：
由于只有三种要排序的，以1为中心点，那么出现0放到最左边即可，出现2放到最右边即可。
那么设置一个指针。从 0 开始，若为0则将它从原数组弹出，然后放到0的位置，若是2则放到末尾。若是1则不变。
1. 出现0和1的情况都将index向前推进1，2的话则不推进。
这样做是one-pass,O(1)，符合进阶条件。缺点是 insert和pop都是 O(n) 时间复杂度的算法。使用deque会快一些，现在也可以。
beat:
64%
测试地址：
https://leetcode.com/problems/sort-colors/description/
"""
class Solution(object):
    def sortColors(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place instead.
        """
        index = 0
        times = 0
        length = len(nums)
        for i in range(length):
            if nums[index] == 0:
                x = nums.pop(index)
                nums.insert(0, x)
                index += 1
            elif nums[index] == 2:
                x = nums.pop(index)
```

```
                nums.append(x)
            else:
                index += 1
```

Array/SpiralMatrix.py

```
"""
Given a matrix of m x n elements (m rows, n columns), return all elements of the
matrix in spiral order.
Example 1:
Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
Output: [1,2,3,6,9,8,7,4,5]
Example 2:
Input:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
思路:
0, 0开始走,
right -> down,
down -> left,
left -> up,
up -> right.
每走过一点，将值添加到结果中，走过的点记为 'x'。当四周都是 'x' 或边界时结束。
测试地址:
https://leetcode.com/problems/spiral-matrix/description/
beat 99.10%.
"""
def checkStop(matrix, x, y):
    t = [(x+1, y),
         (x-1, y),
         (x, y+1),
         (x, y-1),
         (x, y)]
    for i in t:
        try:
            if i[1] < 0 or i[0] < 0:
                continue
            if matrix[i[1]][i[0]] != 'x':
                return False
        except IndexError:
            continue
    else:
        return True
class Solution(object):
    def right(self, matrix, x, y, result, stop):
        if checkStop(matrix, x, y):
            return result
        while 1:
            try:
                # matrix
                if matrix[y][x] == 'x':
```

```
                raise IndexError
            result.append(matrix[y][x])
            matrix[y][x] = 'x'
            x += 1
        except IndexError:
            x -= 1
            return self.down(matrix, x, y+1, result, stop)
    def down(self, matrix, x ,y, result, stop):
        if checkStop(matrix, x, y):
            return result
        while 1:
            try:
                # matrix
                if matrix[y][x] == 'x':
                    raise IndexError
                result.append(matrix[y][x])
                matrix[y][x] = 'x'
                y += 1
            except IndexError:
                y -= 1
                return self.left(matrix, x-1, y, result, stop)
    def left(self, matrix, x, y, result, stop):
        if checkStop(matrix, x, y):
            return result
        while 1:
            try:
                # matrix
                if matrix[y][x] == 'x' or x < 0:
                    raise IndexError
                result.append(matrix[y][x])
                matrix[y][x] = 'x'
                x -= 1
            except IndexError:
                x += 1
                return self.up(matrix, x, y-1, result, stop)
    def up(self, matrix, x, y, result, stop):
        if checkStop(matrix, x, y):
            return result
        while 1:
            try:
                # matrix
                if matrix[y][x] == 'x' or y < 0:
                    raise IndexError
                result.append(matrix[y][x])
                matrix[y][x] = 'x'
                y -= 1
            except IndexError:
                y += 1
                return self.right(matrix, x+1, y, result, stop)
    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        x, y = 0, 0
        result = []
        # right -> down
        # down -> left
```

```
        # left -> up
        # up -> right
        stop = {}
        return self.right(matrix, 0, 0, result, stop)
```

Array/SpiralMatrixII.py

```
"""
Given a positive integer n, generate a square matrix filled with elements from 1
to n2 in spiral order.
Example:
Input: 3
Output:
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
想清楚在写。
beat 94%
测试地址：
https://leetcode.com/problems/spiral-matrix-ii/description/
"""
class Solution(object):
    def generateMatrix(self, n):
        """
        :type n: int
        :rtype: List[List[int]]
        """
        maps = [[0 for i in range(n)] for j in range(n)]
        current_value = [i for i in range(1, n*n+1)]
        current_value.reverse()
        def makeXY(x, y):
            # up
            # down
            # right
            # left
            return [(x, y-1),
                    (x, y+1),
                    (x+1, y),
                    (x-1, y)]
        def right(x, y):
            while 1:
                if not current_value:
                    return maps
                xy = makeXY(x, y)
                if (y > -1 and x > -1) and (y < n and x < n):
                    if maps[y][x] == 0:
                        maps[y][x] = current_value.pop()
                        y, x = xy[2][1], xy[2][0]
                    else:
                        # down
                        return down(x-1, y+1)
                else:
                    # down
                    return down(x-1, y+1)
        def down(x, y):
            while 1:
                if not current_value:
                    return maps
                xy = makeXY(x, y)
```

```
            if (y > -1 and x > -1) and (y < n and x < n):
                if maps[y][x] == 0:
                    maps[y][x] = current_value.pop()
                    y, x = xy[1][1], xy[1][0]
                else:
                    # left
                    return left(x-1, y-1)
            else:
                # left
                return left(x-1, y-1)
        def left(x, y):
            while 1:
                if not current_value:
                    return maps
                xy = makeXY(x, y)
                if y > -1 and x > -1 and y < n and x < n:
                    if maps[y][x] == 0:
                        maps[y][x] = current_value.pop()
                        y, x = xy[3][1], xy[3][0]
                    else:
                        # up
                        return up(x+1, y-1)
                else:
                    # up
                    return up(x+1, y-1)
        def up(x, y):
            while 1:
                if not current_value:
                    return maps
                xy = makeXY(x, y)
                if y > -1 and x > -1 and y < n and x < n:
                    if maps[y][x] == 0:
                        maps[y][x] = current_value.pop()
                        y, x = xy[0][1], xy[0][0]
                    else:
                        # right
                        return right(x+1, y+1)
                else:
                    # right
                    return right(x+1, y+1)
        return right(0, 0)
```

Array/SubarraySumEqualsK.py

```
"""
Given an array of integers and an integer k, you need to find the total number of continuous subarrays whose sum equals to k.
Example 1:
Input:nums = [1,1,1], k = 2
Output: 2
Note:
The length of the array is in range [1, 20,000].
The range of numbers in the array is [-1000, 1000] and the range of the integer k is [-1e7, 1e7].
思路：
1. 直接用了暴力的搜索法，TLE。
2. Discuss 里用的哈希 + 保留和的方式。
具体的是：
   pre 一直累加。
   如果 pre - k 存在于保存的字典中，那么结果里加上这个次数即可。
这种方法可行。但暂时没搞明白具体的原理。
测试地址：
https://leetcode.com/problems/subarray-sum-equals-k/description/
"""
class Solution(object):
    def subarraySum(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        dicts = {0:1}
        result = 0
        pre_sum = 0
        for i in nums:
            pre_sum += i
            result += dicts.get(pre_sum-k, 0)
            dicts[pre_sum] = dicts.get(pre_sum, 0) + 1
        return result
```

Array/SumOfSubarrayMinimums.py

```
"""
Given an array of integers A, find the sum of min(B), where B ranges over every
(contiguous) subarray of A.
Since the answer may be large, return the answer modulo 10^9 + 7.
Example 1:
Input: [3,1,2,4]
Output: 17
Explanation: Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2],
[1,2,4], [3,1,2,4].
Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.  Sum is 17.
Note:
1 <= A.length <= 30000
1 <= A[i] <= 30000
给一个数组，返回所有子数组中最小的数之和。
思路：
每个点都有左右两条路：
 3，1 ，2 ，4
 left 3 == 0，没有比它大的。
 right 3 == 0，到是有比它大的，但和它不挨着。
 _self = 1,
 left * right = 0,
 total = 0 + 0 + 1 + 0
 left 1 == 1，有一个3。
 right 1 == 2，有两个，2,4。
 _self = 1
 left * right = 1 * 2 = 2
 total = 1 + 2 + 1 + 2 = 6。
 # 左边和右边最后进行的是相乘。
 对于每个数的左边来说，只要这个数是最小的就可以了。
 对于每个数的右边来说，它必须是唯一最小的才行。
 反过来也行，反正就是只能出现一个。
 3,1,2,1
 对于第一个 1，来说 向右走 1==1，出现了重复。
 这时不能判断为也是一组，因为如果与它组成 1,2,1 那当进行最后一个 1 左边的判断时，还会出现一个
1,2,1。这样就重复了。
---
第一版的代码只能跑通100中的95个例子..
遇到最大值，3W个，直接 TLE 了。
第一版中每个点都需要对所有的其他点重新计算，导致最差会在 O(n²) 。
优化点：
通过子问题来减少判断的次数。
因为要与之连，所以只要有一个不可以，它前面的就不需要在去管了，肯定是不可以的。
3，1 ，2 ，4
left(加上自身):
 [1, 2, 1, 1]
3 这个点的 1 由其左边比它大的而来，由于没有，所以只有自己 1.
1 这个点 由其左边比它大的而来，1 + 1 = 2
2 这个点 由其左边比它大的而来，1
4 同样。
right(加上自身) 可以倒过来进行判断:
倒过来之后判断的还是左边，这样比较好写。
4,2,1,3
 [1, 2, 3, 1] reverse [1,3,2,1]
因为加上了自身，所以 left * right 即可：
```

```
[1,2,1,1]
[1,3,2,1]
[1,6,2,1]
 3 1 2 4
 3 + 6 + 4 + 4 = 17.
写的时候还需要一个额外变量：
 [4, 2, 1, 3]
 这个变量存的是目前的总和：
   最小的数
 [(4,1)]
     个数
 [(2,2)]
 [(1,3)]
 [(1,3),(3,1)]
如果没有的话，又要与原来的一模一样了。
ok，这次他通过了， 268ms。
测试地址：
https://leetcode.com/contest/weekly-contest-102/problems/sum-of-subarray-minimums/
"""
class Solution(object):
    def sumSubarrayMins(self, A):
        """
        :type A: List[int]
        :rtype: int
        """
        mod = 10**9 + 7
        # count
        left = []
        right = []
        # (num, count)
        _left = []
        _right = []
        for i in A:
            base = 1
            while _left:
                if i < _left[-1][0]:
                    base += _left[-1][1]
                    _left.pop()
                else:
                    break
            _left.append((i, base))
            left.append(base)
        for i in reversed(A):
            base = 1
            while _right:
                if i <= _right[-1][0]:
                    base += _right[-1][1]
                    _right.pop()
                else:
                    break
            _right.append((i, base))
            right.append(base)
        right.reverse()
        result = 0
        for i in range(len(A)):
            result += A[i] * left[i] * right[i]
        return result % mod
```

```
        # result = 0
        # length = len(A)
        # for i in range(length):
        #     _self = 1
        #     left = 0
        #     right = 0
        #     for j in range(i-1, -1, -1):
        #         if A[i] < A[j]:
        #             left += 1
        #             continue
        #         break
        #     for j in range(i+1, length):
        #         if A[i] <= A[j]:
        #             right += 1
        #             continue
        #         break
        #     sums = left * right
        #     # print([sums, _self, left, right])
        #     result += (A[i] * sum([sums, _self, left, right]))
        # # print(result)
        # return result % mod
```

Array/SurfaceAreaOf3DShapes.py

```
"""
Contest 1:
On a N * N grid, we place some 1 * 1 * 1 cubes.
Each value v = grid[i][j] represents a tower of v cubes placed on top of grid
cell (i, j).
Return the total surface area of the resulting shapes.
Example 1:
Input: [[2]]
Output: 10
Example 2:
Input: [[1,2],[3,4]]
Output: 34
Example 3:
Input: [[1,0],[0,2]]
Output: 16
Example 4:
Input: [[1,1,1],[1,0,1],[1,1,1]]
Output: 32
Example 5:
Input: [[2,2,2],[2,1,2],[2,2,2]]
Output: 46
Note:
1 <= N <= 50
0 <= grid[i][j] <= 50
测试地址：
https://leetcode.com/contest/weekly-contest-99/problems/surface-area-of-3d-
shapes/
每个叠起来的正方体面数有
6 * x - 2 * (x-1)个。
确定好上下左右每个被覆盖的面减去即可。
"""
class Solution(object):
    def surfaceArea(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        result = 0
        lengthY = len(grid)
        lengthX = len(grid[0])
        for i in range(lengthY):
            for j in range(lengthX):
                if grid[i][j] != 0:
                    temp_result = 6 * grid[i][j] - 2 * (grid[i][j] - 1)
                    left = min(grid[i][j-1], grid[i][j]) if j-1 >= 0 else 0
                    right = min(grid[i][j+1], grid[i][j]) if j+1 <= lengthX-1
else 0
                    up = min(grid[i-1][j], grid[i][j]) if i-1 >= 0 else 0
                    down = min(grid[i+1][j], grid[i][j]) if i+1 <= lengthY-1
else 0
                    result += (temp_result - sum([left, right, up, down]))
        return result
```

Array/SurroundedRegions.py

```
"""
Given a 2D board containing 'X' and 'O' (the letter O), capture all regions
surrounded by 'X'.
A region is captured by flipping all 'O's into 'X's in that surrounded region.
Example:
X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:
X X X X
X X X X
X X X X
X O X X
Explanation:
Surrounded regions shouldn’t be on the border, which means that any 'O' on the
border of the board are not flipped to 'X'.
Any 'O' that is not on the border and it is not connected to an 'O' on the border
will be flipped to 'X'.
Two cells are connected if they are adjacent cells connected horizontally or
vertically.
给一个二维数组里面是 'X' 和 'O'，所有并未连着任何边界的 'O' 都变成 'X' 即可。
X X X X
X X O O
X O X X
X X X X
变形后为
X X X X
X X O O
X X X X
X X X X
思路：
1. 一开始用递归，迭代每一个点，把里面所有的和带边界的 O 及其相邻的找出来，把不相邻的 和 相邻的
分别放到两个坐标列表里，然后进行 'O' 'X' 的分配..
  这个思路和写法都没问题... 无奈的是测试数据过大会导致超过最大递归深度，而且比较慢。
  放弃。
2. 这个思路沿用上一个的思路，只不过这次所做的是只迭代四周的即可。 因为不与四周的 'O' 相邻的是不
需要在一开始就变得。
所以只迭代找出四周的 'O' 并用递归把它的邻居们都变成另一个字符，或者存坐标也可以。
之后迭代原地图，然后把是标识的变回 'O'，原来是 'O' 的变成 'X' 即可。
效率 O(row*column)
测试链接：
https://leetcode.com/problems/surrounded-regions/description/
beat 40%~60%.
时间上可以通过一些细节做优化，能提升数毫秒，不过就此题来说没有必要，前面的基本都是这个思路。
"""
class Solution(object):
    def makeAround(self, x, y):
        # if x - 1 >= 0 and y - 1 >= 0 and x + 1 < x_maxes and y + 1 < y_maxes:
            # return True
            # right left down up
        return [(y, x-1),
                (y, x+1),
                (y-1, x),
```

```
                (y+1, x)]
        # return False
    def solve(self, board):
        """
        :type board: List[List[str]]
        :rtype: void Do not return anything, modify board in-place instead.
        """
        #     return board
        if not board:
            x_length = 0
        else:
            x_length = len(board[0])
        y_length = len(board)
        def translate_o_to_e(x, y):
            # pass
            # coordinate = self.makeAround(x, y)
            # if coordinate:
            for i in self.makeAround(x, y):
                if i[0] >= 0 and i[1] >= 0 and i[0] < y_length and i[1] <
x_length:
                    if board[i[0]][i[1]] == "O":
                        board[i[0]][i[1]] = "E"
                        translate_o_to_e(i[1], i[0])
        # up down
        e_coordinate = []
        for i in range(x_length):
            if board[0][i] == 'O':
                board[0][i] = 'E'
                translate_o_to_e(i, 0)
            if board[y_length-1][i] == "O":
                board[y_length-1][i] = 'E'
                translate_o_to_e(i, y_length-1)
        # left right
        for i in range(y_length):
            if board[i][0] == 'O':
                board[i][0] = 'E'
                translate_o_to_e(0, i)
            if board[i][x_length-1] == 'O':
                board[i][x_length-1] = 'E'
                translate_o_to_e(x_length-1, i)
        for y in range(y_length):
            for x in range(x_length):
                if board[y][x] == 'E':
                    board[y][x] = 'O'
                elif board[y][x] == 'O':
                    board[y][x] = 'X'
#         e_coordinate = []
#         for y in range(y_length):
#             for x in range(x_length):
#                 if board[y][x] == 'O':
#                     coordinate = self.makeAround(x, y, x_length, y_length)
#                     if not coordinate:
#                         board[y][x] = 'E'
#                         e_coordinate.append((y, x))
#                     else:
#                         for i in coordinate:
#                             if board[i[0]][i[1]] == 'E':
#                                 board[y][x] = 'E'
```

```
#                                 e_coordinate.append((y, x))
#                                 break
#                         else:
#                             board[y][x] = 'X'
#         for i in e_coordinate:
#             board[i[0]][i[1]] = 'O'
#         def found_o(y, x):
#             result = []
#             coordinate = self.makeAround(x, y, x_length, y_length)
#             if not coordinate:
#                 return False, result
#             for i in coordinate:
#                 if board[i[0]][i[1]] == 'O':
#                     board[i[0]][i[1]] = 'E'
#                     x = found_o(i[0], i[1])
#                     result.append(i)
#                     result.extend(x[1])
#                     if not x[0]:
#                         return False, result
#             return True, result
#         o_coordinate = []
#         x_coordinate = []
#         for y in range(y_length):
#             for x in range(x_length):
#                 if board[y][x] == 'O':
#                     x = found_o(y, x)
#                     if x[0]:
#                         x_coordinate.extend(x[1])
#                     else:
#                         o_coordinate.extend(x[1])
#         # print(o_coordinate)
#         for i in o_coordinate:
#             board[i[0]][i[1]] = 'O'
#         # print(x_coordinate)
#         for i in x_coordinate:
#             # print(i)
#             board[i[0]][i[1]] = 'X'
              # coordinate = self.makeAround(x, y, x_length, y_length)
              # if coordinate:
                  # board[y][x] = 'X'
```

Array/ThirdMaximumNumbers.py

```
"""
Given a non-empty array of integers, return the third maximum number in this
array. If it does not exist, return the maximum number. The time complexity must
be in O(n).
Example 1:
Input: [3, 2, 1]
Output: 1
Explanation: The third maximum is 1.
Example 2:
Input: [1, 2]
Output: 2
Explanation: The third maximum does not exist, so the maximum (2) is returned
instead.
Example 3:
Input: [2, 2, 3, 1]
Output: 1
Explanation: Note that the third maximum here means the third maximum distinct
number.
Both numbers with value 2 are both considered as second maximum.
返回第三大的数，如果不存在则返回最大的，重复的算一个。
https://leetcode.com/problems/third-maximum-number/description/
找到第 k 大个数一般的思路有：
1. 排序后放到数组中，排序算法使用归并和快排在理想情况下都是O(nlogn)，归并比较稳定一些。之后的索
引是O(1)。
   这种的适合并不需要插入的情况，因为每次插入的时间复杂度为 O(n)。
2. 建立二叉搜索树，进阶的话红黑树或AVL树。
   这种情况下搜索和插入在理想情况下都是O(logn)。
3. 就此题来说的O(n)思路：
   建立三个变量，first,second,third,首先确保不是None，然后挨个放数据，最后输出结果。
这里直接用排序了。
"""
class Solution(object):
    def thirdMax(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        nums = set(nums)
        if len(nums) < 3:
            return max(nums)
        return sorted(nums, reverse=True)[2]
```

Array/ThreeSum.py

```
"""
Given an array nums of n integers, are there elements a, b, c in nums such that a
+ b + c = 0? Find all unique triplets in the array which gives the sum of zero.
Note:
The solution set must not contain duplicate triplets.
Example:
Given array nums = [-1, 0, 1, 2, -1, -4],
A solution set is:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
3sum。
a + b + c = 0.
1. 第一次尝试首先是减去一个数，然后剩下的用twoSum的二分法做判断进行查找，结果非常慢。
TLE.
2. 学习了一波 O(n) 的算法:
start->+  -<-end
当等于之后，还有可能存在同样会相等的数：
我们预先去重了，所以是唯一的，start和end同时朝着各自的方向进1就可以。
然后就是去重问题：
在非0的情况下，如果下标多于等于2个，则判断-2*A 是否在里面。
在0的情况下，多于等于3个才可以。
beat 94%
测试链接：
https://leetcode.com/problems/3sum/description/
"""
class Solution(object):
    def threeSum(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        sortedNums = nums
        result = []
        index_dict = {}
        for i, d in enumerate(sortedNums):
            try:
                index_dict[d].append(i)
            except KeyError:
                index_dict[d] = [i]
        sortedNums = sorted(set(sortedNums))
        length = len(sortedNums) - 1
        result = []
        for i, data in enumerate(sortedNums):
            if data > 0:
                break
            target = 0 - data
            start = i + 1
            end = length
            while start < end:
                if sortedNums[start] + sortedNums[end] == target:
                    # print(target)
```

```
                    result.append(sorted((sortedNums[start], sortedNums[end],
data)))
                    # break
                    end -= 1
                    start += 1
                    continue
                if sortedNums[start] + sortedNums[end] > target:
                    end -= 1
                else:
                    start += 1
        for i in index_dict:
            if i == 0:
                if len(index_dict.get(i)) >= 3:
                    result.append([0, 0, 0])
                    continue
            if len(index_dict.get(i)) >= 2 and i != 0:
                if index_dict.get(-2*i):
                    result.append(sorted((i,i,-2*i)))
        return result
            # for j in range(i+1, length):
                # pass
                # target = j - data
                # (data, sortedNums[j], target-sortedNums[j])
                # x = self.binarySearch(sortedNums, target, j+1)
                # if x:
                #     result.append(sorted((data, sortedNums[j], target-
sortedNums[j])))
        # length = len(sortedNums)
        #         data2 = sortedNums[j]
        #         if index_dict.get(target-data2):
        #             if data == target-data2:
        #                 index_dict.get(data)[-1] >= j:
        #             result.append(sorted((data, data2, target-data2)))
    # def twoSum(self, ranges, nums, target):
        # print(target)
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        # indexmap = dict()
        # num1index = 0
        # for num0 in ranges:
        # num2index = None
        # num2index
        # for num0 in ranges:
        #     num1 = nums[num0]
        #     num2index = indexmap.get(target-num1)
        #     if (
        #         (num2index is not None) and
        #         (num2index != num1index)
        #        ):
        #         break
        #     else:
        #         indexmap[num1] = num1index
        #     num1index = num1index + 1
        # result = [num2index, num1index]
        # # print(result)
```

```
        # if all(result):
        #     return result
        # return False
            # for j, data2 in enumerate(sortedNums):
                #     a = sorted((data, data2, target-data2))
                #     if len(set(a)) != 3:
                #      if len(index_dict.get(target-data2)) >= 2:
                #         if a not in result:
                #             result.append(a)
                #     else:
                #         if a not in result:
                    # print((data, data2, target-data2))
                    # for x in (data, data2, target-data2):
                    #     temp = index_dict.get(x)
                    #     if not temp:
                    #         break
                    #     temp.pop()
                    #     if not temp:
                    #         index_dict.pop(x)
                    # else:
                    # if len(set((data, data2, target-data2))) == 2:
                    #         result.append(sorted((data, data2, target-data2)))
                    # else:
        #         sortedNums[i+1:]
                # if self.binarySearch(sortedNums[i+1:][j+1:], target-data2):
                    # if (data, data2, target-data2) not in result:
        return result
```

```
a =
[82597,-9243,62390,83030,-97960,-26521,-61011,83390,-38677,12333,75987,46091,837
94,19355,-71037,-6242,-28801,324,1202,-90885,-2989,-95597,-34333,35528,5680,8909
3,-90606,50360,-29393,-27012,53313,65213,99818,-82405,-41661,-3333,-51952,72135,
-1523,26377,74685,96992,92263,15929,5467,-99555,-43348,-41689,-60383,-3990,32165
,65265,-72973,-58372,12741,-48568,-46596,72419,-1859,34153,62937,81310,-61823,-9
6770,-54944,8845,-91184,24208,-29078,31495,65258,14198,85395,70506,-40908,56740,
-12228,-40072,32429,93001,68445,-73927,25731,-91859,-24150,10093,-60271,-81683,-
18126,51055,48189,-6468,25057,81194,-58628,74042,66158,-14452,-49851,-43667,1109
2,39189,-17025,-79173,13606,83172,92647,-59741,19343,-26644,-57607,82908,-20655,
1637,80060,98994,39331,-31274,-61523,91225,-72953,13211,-75116,-98421,-41571,-69
074,99587,39345,42151,-2460,98236,15690,-52507,-95803,-48935,-46492,-45606,-7925
4,-99851,52533,73486,39948,-7240,71815,-585,-96252,90990,-93815,93340,-71848,587
33,-14859,-83082,-75794,-82082,-24871,-15206,91207,-56469,-93618,67131,-8682,757
19,87429,-98757,-7535,-24890,-94160,85003,33928,75538,97456,-66424,-60074,-8527,
-28697,-22308,2246,-70134,-82319,-10184,87081,-34949,-28645,-47352,-83966,-60418
,-15293,-53067,-25921,55172,75064,95859,48049,34311,-86931,-38586,33686,-36714,9
6922,76713,-22165,-80585,-34503,-44516,39217,-28457,47227,-94036,43457,24626,-87
359,26898,-70819,30528,-32397,-69486,84912,-1187,-98986,-32958,4280,-79129,-6560
4,9344,58964,50584,71128,-55480,24986,15086,-62360,-42977,-49482,-77256,-36895,-
74818,20,3063,-49426,28152,-97329,6086,86035,-88743,35241,44249,19927,-10660,894
04,24179,-26621,-6511,57745,-28750,96340,-97160,-97822,-49979,52307,79462,94273,
-24808,77104,9255,-83057,77655,21361,55956,-9096,48599,-40490,-55107,2689,29608,
20497,66834,-34678,23553,-81400,-66630,-96321,-34499,-12957,-20564,25610,-4322,-
58462,20801,53700,71527,24669,-54534,57879,-3221,33636,3900,97832,-27688,-98715,
5992,24520,-55401,-57613,-69926,57377,-77610,20123,52174,860,60429,-91994,-62403
,-6218,-90610,-37263,-15052,62069,-96465,44254,89892,-3406,19121,-41842,-87783,-
64125,-56120,73904,-22797,-58118,-4866,5356,75318,46119,21276,-19246,-9241,-9742
5,57333,-15802,93149,25689,-5532,95716,39209,-87672,-29470,-16324,-15331,27632,-
39454,56530,-16000,29853,46475,78242,-46602,83192,-73440,-15816,50964,-36601,897
58,38375,-40007,-36675,-94030,67576,46811,-64919,45595,76530,40398,35845,41791,6
7697,-30439,-82944,63115,33447,-36046,-50122,-34789,43003,-78947,-38763,-89210,3
2756,-20389,-31358,-90526,-81607,88741,86643,98422,47389,-75189,13091,95993,-155
01,94260,-25584,-1483,-67261,-70753,25160,89614,-90620,-48542,83889,-12388,-9642
,-37043,-67663,28794,-8801,13621,12241,55379,84290,21692,-95906,-85617,-17341,-6
3767,80183,-4942,-51478,30997,-13658,8838,17452,-82869,-39897,68449,31964,98158,
-49489,62283,-62209,-92792,-59342,55146,-38533,20496,62667,62593,36095,-12470,54
53,-50451,74716,-17902,3302,-16760,-71642,-34819,96459,-72860,21638,47342,-69897
,-40180,44466,76496,84659,13848,-91600,-90887,-63742,-2156,-84981,-99280,94326,-
33854,92029,-50811,98711,-36459,-75555,79110,-88164,-97397,-84217,97457,64387,30
513,-53190,-83215,252,2344,-27177,-92945,-89010,82662,-11670,86069,53417,42702,9
7082,3695,-14530,-46334,17910,77999,28009,-12374,15498,-46941,97088,-35030,95040
,92095,-59469,-24761,46491,67357,-66658,37446,-65130,-50416,99197,30925,27308,54
122,-44719,12582,-99525,-38446,-69050,-22352,94757,-56062,33684,-40199,-46399,96
842,-50881,-22380,-65021,40582,53623,-76034,77018,-97074,-84838,-22953,-74205,79
715,-33920,-35794,-91369,73421,-82492,63680,-14915,-33295,37145,76852,-69442,601
25,-74166,74308,-1900,-30195,-16267,-60781,-27760,5852,38917,25742,-3765,49097,-
63541,98612,-92865,-30248,9612,-8798,53262,95781,-42278,-36529,7252,-27394,-5021
,59178,80934,-48480,-75131,-54439,-19145,-48140,98457,-6601,-51616,-89730,78028,
32083,-48904,16822,-81153,-8832,48720,-80728,-45133,-86647,-4259,-40453,2590,286
13,50523,-4105,-27790,-74579,-17223,63721,33489,-47921,97628,-97691,-14782,-6564
4,18008,-93651,-71266,80990,-76732,-47104,35368,28632,59818,-86269,-89753,34557,
-92230,-5933,-3487,-73557,-13174,-43981,-43630,-55171,30254,-83710,-99583,-13500
,71787,5017,-25117,-78586,86941,-3251,-23867,-36315,75973,86272,-45575,77462,-98
836,-10859,70168,-32971,-38739,-12761,93410,14014,-30706,-77356,-85965,-62316,63
918,-59914,-64088,1591,-10957,38004,15129,-83602,-51791,34381,-89382,-26056,8942
,5465,71458,-73805,-87445,-19921,-80784,69150,-34168,28301,-68955,18041,6059,823
42,9947,39795,44047,-57313,48569,81936,-2863,-80932,32976,-86454,-84207,33033,32
```

867,9104,-16580,-25727,80157,-70169,53741,86522,84651,68480,84018,61932,7332,-61
322,-69663,76370,41206,12326,-34689,17016,82975,-23386,39417,72793,44774,-96259,
3213,79952,29265,-61492,-49337,14162,65886,3342,-41622,-62659,-90402,-24751,8851
1,54739,-21383,-40161,-96610,-24944,-602,-76842,-21856,69964,43994,-15121,-85530
,12718,13170,-13547,69222,62417,-75305,-81446,-38786,-52075,-23110,97681,-82800,
-53178,11474,35857,94197,-58148,-23689,32506,92154,-64536,-73930,-77138,97446,-8
3459,70963,22452,68472,-3728,-25059,-49405,95129,-6167,12808,99918,30113,-12641,
-26665,86362,-33505,50661,26714,33701,89012,-91540,40517,-12716,-57185,-87230,29
914,-59560,13200,-72723,58272,23913,-45586,-96593,-26265,-2141,31087,81399,92511
,-34049,20577,2803,26003,8940,42117,40887,-82715,38269,40969,-50022,72088,21291,
-67280,-16523,90535,18669,94342,-39568,-88080,-99486,-20716,23108,-28037,63342,3
6863,-29420,-44016,75135,73415,16059,-4899,86893,43136,-7041,33483,-67612,25327,
40830,6184,61805,4247,81119,-22854,-26104,-63466,63093,-63685,60369,51023,51644,
-16350,74438,-83514,99083,10079,-58451,-79621,48471,67131,-86940,99093,11855,-22
272,-67683,-44371,9541,18123,37766,-70922,80385,-57513,-76021,-47890,36154,72935
,84387,-92681,-88303,-7810,59902,-90,-64704,-28396,-66403,8860,13343,33882,85680
,7228,28160,-14003,54369,-58893,92606,-63492,-10101,64714,58486,29948,-44679,-22
763,10151,-56695,4031,-18242,-36232,86168,-14263,9883,47124,47271,92761,-24958,-
73263,-79661,-69147,-18874,29546,-92588,-85771,26451,-86650,-43306,-59094,-47492
,-34821,-91763,-47670,33537,22843,67417,-759,92159,63075,94065,-26988,55276,6590
3,30414,-67129,-99508,-83092,-91493,-50426,14349,-83216,-76090,32742,-5306,-9331
0,-60750,-60620,-45484,-21108,-58341,-28048,-52803,69735,78906,81649,32565,-8680
4,-83202,-65688,-1760,89707,93322,-72750,84134,71900,-37720,19450,-78018,22001,-
23604,26276,-21498,65892,-72117,-89834,-23867,55817,-77963,42518,93123,-83916,63
260,-2243,-97108,85442,-36775,17984,-58810,99664,-19082,93075,-69329,87061,79713
,16296,70996,13483,-74582,49900,-27669,-40562,1209,-20572,34660,83193,75579,7344
,64925,88361,60969,3114,44611,-27445,53049,-16085,-92851,-53306,13859,-33532,866
22,-75666,-18159,-98256,51875,-42251,-27977,-18080,23772,38160,41779,9147,94175,
99905,-85755,62535,-88412,-52038,-68171,93255,-44684,-11242,-104,31796,62346,-54
931,-55790,-70032,46221,56541,-91947,90592,93503,4071,20646,4856,-63598,15396,-5
0708,32138,-85164,38528,-89959,53852,57915,-42421,-88916,-75072,67030,-29066,495
42,-71591,61708,-53985,-43051,28483,46991,-83216,80991,-46254,-48716,39356,-8270
,-47763,-34410,874,-1186,-7049,28846,11276,21960,-13304,-11433,-4913,55754,79616
,70423,-27523,64803,49277,14906,-97401,-92390,91075,70736,21971,-3303,55333,-939
96,76538,54603,-75899,98801,46887,35041,48302,-52318,55439,24574,14079,-24889,83
440,14961,34312,-89260,-22293,-81271,-2586,-71059,-10640,-93095,-5453,-70041,665
43,74012,-11662,-52477,-37597,-70919,92971,-17452,-67306,-80418,7225,-89296,2429
6,86547,37154,-10696,74436,-63959,58860,33590,-88925,-97814,-83664,85484,-8385,-
50879,57729,-74728,-87852,-15524,-91120,22062,28134,80917,32026,49707,-54252,-44
319,-35139,13777,44660,85274,25043,58781,-89035,-76274,6364,-63625,72855,43242,-
35033,12820,-27460,77372,-47578,-61162,-70758,-1343,-4159,64935,56024,-2151,4377
0,19758,-30186,-86040,24666,-62332,-67542,73180,-25821,-27826,-45504,-36858,-120
41,20017,-24066,-56625,-52097,-47239,-90694,8959,7712,-14258,-5860,55349,61808,-
4423,-93703,64681,-98641,-25222,46999,-83831,-54714,19997,-68477,66073,51801,-66
491,52061,-52866,79907,-39736,-68331,68937,91464,98892,910,93501,31295,-85873,27
036,-57340,50412,21,-2445,29471,71317,82093,-94823,-54458,-97410,39560,-7628,664
52,39701,54029,37906,46773,58296,60370,-61090,85501,-86874,71443,-72702,-72047,1
4848,34102,77975,-66294,-36576,31349,52493,-70833,-80287,94435,39745,-98291,8452
4,-18942,10236,93448,50846,94023,-6939,47999,14740,30165,81048,84935,-19177,-135
94,32289,62628,-90612,-542,-66627,64255,71199,-83841,-82943,-73885,8623,-67214,-
9474,-35249,62254,-14087,-90969,21515,-83303,94377,-91619,19956,-98810,96727,-91
939,29119,-85473,-82153,-69008,44850,74299,-76459,-86464,8315,-49912,-28665,5905
2,-69708,76024,-92738,50098,18683,-91438,18096,-19335,35659,91826,15779,-73070,6
7873,-12458,-71440,-46721,54856,97212,-81875,35805,36952,68498,81627,-34231,8171
2,27100,-9741,-82612,18766,-36392,2759,41728,69743,26825,48355,-17790,17165,5655
8,3295,-24375,55669,-16109,24079,73414,48990,-11931,-78214,90745,19878,35673,-15
317,-89086,94675,-92513,88410,-93248,-19475,-74041,-19165,32329,-26266,-46828,-1
8747,45328,8990,-78219,-25874,-74801,-44956,-54577,-29756,-99822,-35731,-18348,-

68915,-83518,-53451,95471,-2954,-13706,-8763,-21642,-37210,16814,-60070,-42743,27697,-36333,-42362,11576,85742,-82536,68767,-56103,-63012,71396,-78464,-68101,-15917,-11113,-3596,77626,-60191,-30585,-73584,6214,-84303,18403,23618,-15619,-89755,-59515,-59103,-74308,-63725,-29364,-52376,-96130,70894,-12609,50845,-2314,42264,-70825,64481,55752,4460,-68603,-88701,4713,-50441,-51333,-77907,97412,-66616,-49430,60489,-85262,-97621,-18980,44727,-69321,-57730,66287,-92566,-64427,-14270,11515,-92612,-87645,61557,24197,-81923,-39831,-10301,-23640,-76219,-68025,92761,-76493,68554,-77734,-95620,-11753,-51700,98234,-68544,-61838,29467,46603,-18221,-35441,74537,40327,-58293,75755,-57301,-7532,-94163,18179,-14388,-22258,-46417,-48285,18242,-77551,82620,250,-20060,-79568,-77259,82052,-98897,-75464,48773,-79040,-11293,45941,-67876,-69204,-46477,-46107,792,60546,-34573,-12879,-94562,20356,-48004,-62429,96242,40594,2099,99494,25724,-39394,-2388,-18563,-56510,-83570,-29214,3015,74454,74197,76678,-46597,60630,-76093,37578,-82045,-24077,62082,-87787,-74936,58687,12200,-98952,70155,-77370,21710,-84625,-60556,-84128,925,65474,-15741,-94619,88377,89334,44749,22002,-45750,-93081,-14600,-83447,46691,85040,-66447,-80085,56308,44310,24979,-29694,57991,4675,-71273,-44508,13615,-54710,23552,-78253,-34637,50497,68706,81543,-88408,-21405,6001,-33834,-21570,-46692,-25344,20310,71258,-97680,11721,59977,59247,-48949,98955,-50276,-80844,-27935,-76102,55858,-33492,40680,66691,-33188,8284,64893,-7528,6019,-85523,8434,-64366,-56663,26862,30008,-7611,-12179,-70076,21426,-11261,-36864,-61937,-59677,929,-21052,3848,-20888,-16065,98995,-32293,-86121,-54564,77831,68602,74977,31658,40699,29755,98424,80358,-69337,26339,13213,-46016,-18331,64713,-46883,-58451,-70024,-92393,-4088,70628,-51185,71164,-75791,-1636,-29102,-16929,-87650,-84589,-24229,-42137,-15653,94825,13042,88499,-47100,-90358,-7180,29754,-65727,-42659,-85560,-9037,-52459,20997,-47425,17318,21122,20472,-23037,65216,-63625,-7877,-91907,24100,-72516,22903,-85247,-8938,73878,54953,87480,-31466,-99524,35369,-78376,89984,-15982,94045,-7269,23319,-80456,-37653,-76756,2909,81936,54958,-12393,60560,-84664,-82413,66941,-26573,-97532,64460,18593,-85789,-38820,-92575,-43663,-89435,83272,-50585,13616,-71541,-53156,727,-27644,16538,34049,57745,34348,35009,16634,-18791,23271,-63844,95817,21781,16590,59669,15966,-6864,48050,-36143,97427,-59390,96931,78939,-1958,50777,43338,-51149,39235,-27054,-43492,67457,-83616,37179,10390,85818,2391,73635,87579,-49127,-81264,-79023,-81590,53554,-74972,-83940,-13726,-39095,29174,78072,76104,47778,25797,-29515,-6493,-92793,22481,-36197,-65560,42342,15750,97556,99634,-56048,-35688,13501,63969,-74291,50911,39225,93702,-3490,-59461,-30105,-46761,-80113,92906,-68487,50742,36152,-90240,-83631,24597,-50566,-15477,18470,77038,40223,-80364,-98676,70957,-63647,99537,13041,31679,86631,37633,-16866,13686,-71565,21652,-46053,-80578,-61382,68487,-6417,4656,20811,67013,-30868,-11219,46,74944,14627,56965,42275,-52480,52162,-84883,-52579,-90331,92792,42184,-73422,-58440,65308,-25069,5475,-57996,59557,-17561,2826,-56939,14996,-94855,-53707,99159,43645,-67719,-1331,21412,41704,31612,32622,1919,-69333,-69828,22422,-78842,57896,-17363,27979,-76897,35008,46482,-75289,65799,20057,7170,41326,-76069,90840,-81253,-50749,3649,-42315,45238,-33924,62101,96906,58884,-7617,-28689,-66578,62458,50876,-57553,6739,41014,-64040,-34916,37940,13048,-97478,-11318,-89440,-31933,-40357,-59737,-76718,-14104,-31774,28001,4103,41702,-25120,-31654,63085,-3642,84870,-83896,-76422,-61520,12900,88678,85547,33132,-88627,52820,63915,-27472,78867,-51439,33005,-23447,-3271,-39308,39726,-74260,-31874,-36893,93656,910,-98362,60450,-88048,99308,13947,83996,-90415,-35117,70858,-55332,-31721,97528,82982,-86218,6822,25227,36946,97077,-4257,-41526,56795,89870,75860,-70802,21779,14184,-16511,-89156,-31422,71470,69600,-78498,74079,-19410,40311,28501,26397,-67574,-32518,68510,38615,19355,-6088,-97159,-29255,-92523,3023,-42536,-88681,64255,41206,44119,52208,39522,-52108,91276,-70514,83436,63289,-79741,9623,99559,12642,85950,83735,-21156,-67208,98088,-7341,-27763,-30048,-44099,-14866,-45504,-91704,19369,13700,10481,-49344,-85686,33994,19672,36028,60842,66564,-24919,33950,-93616,-47430,-35391,-28279,56806,74690,39284,-96683,-7642,-75232,37657,-14531,-86870,-9274,-26173,98640,88652,64257,46457,37814,-19370,9337,-22556,-41525,39105,-28719,51611,-93252,98044,-90996,21710,-47605,-64259,-32727,53611,-31918,-3555,33316,-66472,21274,-37731,-2919,15016,48779,-88868,1897,41728,46344,-89667,37848,68092,-44011,85354,-43776,38739,-31423,-66330,65167,-22016,59405,34328,-60042,87660,-67698,-59174,-

1408,-46809,-43485,-88807,-60489,13974,22319,55836,-62995,-37375,-4185,32687,-36
551,-75237,58280,26942,-73756,71756,78775,-40573,14367,-71622,-77338,24112,23414
,-7679,-51721,87492,85066,-21612,57045,10673,-96836,52461,-62218,-9310,65862,-22
748,89906,-96987,-98698,26956,-43428,46141,47456,28095,55952,67323,-36455,-60202
,-43302,-82932,42020,77036,10142,60406,70331,63836,58850,-66752,52109,21395,-102
38,-98647,-41962,27778,69060,98535,-28680,-52263,-56679,66103,-42426,27203,80021
,10153,58678,36398,63112,34911,20515,62082,-15659,-40785,27054,43767,-20289,6583
8,-6954,-60228,-72226,52236,-35464,25209,-15462,-79617,-41668,-84083,62404,-6906
2,18913,46545,20757,13805,24717,-18461,-47009,-25779,68834,64824,34473,39576,315
70,14861,-15114,-41233,95509,68232,67846,84902,-83060,17642,-18422,73688,77671,-
26930,64484,-99637,73875,6428,21034,-73471,19664,-68031,15922,-27028,48137,54955
,-82793,-41144,-10218,-24921,-28299,-2288,68518,-54452,15686,-41814,66165,-72207
,-61986,80020,50544,-99500,16244,78998,40989,14525,-56061,-24692,-94790,21111,37
296,-90794,72100,70550,-31757,17708,-74290,61910,78039,-78629,-25033,73172,-9195
3,10052,64502,99585,-1741,90324,-73723,68942,28149,30218,24422,16659,10710,-6259
4,94249,96588,46192,34251,73500,-65995,-81168,41412,-98724,-63710,-54696,-52407,
19746,45869,27821,-94866,-76705,-13417,-61995,-71560,43450,67384,-8838,-80293,-2
8937,23330,-89694,-40586,46918,80429,-5475,78013,25309,-34162,37236,-77577,86744
,26281,-29033,-91813,35347,13033,-13631,-24459,3325,-71078,-75359,81311,19700,47
678,-74680,-84113,45192,35502,37675,19553,76522,-51098,-18211,89717,4508,-82946,
27749,85995,89912,-53678,-64727,-14778,32075,-63412,-40524,86440,-2707,-36821,63
850,-30883,67294,-99468,-23708,34932,34386,98899,29239,-23385,5897,54882,98660,4
9098,70275,17718,88533,52161,63340,50061,-89457,19491,-99156,24873,-17008,64610,
-55543,50495,17056,-10400,-56678,-29073,-42960,-76418,98562,-88104,-96255,10159,
-90724,54011,12052,45871,-90933,-69420,67039,37202,78051,-52197,-40278,-58425,65
414,-23394,-1415,6912,-53447,7352,17307,-78147,63727,98905,55412,-57658,-32884,-
44878,22755,39730,3638,35111,39777,74193,38736,-11829,-61188,-92757,55946,-71232
,-63032,-83947,39147,-96684,-99233,25131,-32197,24406,-55428,-61941,25874,-69453
,64483,-19644,-68441,12783,87338,-48676,66451,-447,-61590,50932,-11270,29035,656
98,-63544,10029,80499,-9461,86368,91365,-81810,-71914,-52056,-13782,44240,-30093
,-2437,24007,67581,-17365,-69164,-8420,-69289,-29370,48010,90439,13141,69243,506
68,39328,61731,78266,-81313,17921,-38196,55261,9948,-24970,75712,-72106,28696,74
61,31621,61047,51476,56512,11839,-96916,-82739,28924,-99927,58449,37280,69357,11
219,-32119,-62050,-48745,-83486,-52376,42668,82659,68882,38773,46269,-96005,9763
0,25009,-2951,-67811,99801,81587,-79793,-18547,-83086,69512,33127,-92145,-88497,
47703,59527,1909,88785,-88882,69188,-46131,-5589,-15086,36255,-53238,-33009,8266
4,53901,35939,-42946,-25571,33298,69291,53199,74746,-40127,-39050,91033,51717,-9
8048,87240,36172,65453,-94425,-63694,-30027,59004,88660,3649,-20267,-52565,-6732
1,34037,4320,91515,-56753,60115,27134,68617,-61395,-26503,-98929,-8849,-63318,10
709,-16151,61905,-95785,5262,23670,-25277,90206,-19391,45735,37208,-31992,-92450
,18516,-90452,-58870,-58602,93383,14333,17994,82411,-54126,-32576,35440,-60526,-
78764,-25069,-9022,-394,92186,-38057,55328,-61569,67780,77169,19546,-92664,-9494
8,44484,-13439,83529,27518,-48333,72998,38342,-90553,-98578,-76906,81515,-16464,
78439,92529,35225,-39968,-10130,-7845,-32245,-74955,-74996,67731,-13897,-82493,3
3407,93619,59560,-24404,-57553,19486,-45341,34098,-24978,-33612,79058,71847,7671
3,-95422,6421,-96075,-59130,-28976,-16922,-62203,69970,68331,21874,40551,89650,5
1908,58181,66480,-68177,34323,-3046,-49656,-59758,43564,-10960,-30796,15473,-202
16,46085,-85355,41515,-30669,-87498,57711,56067,63199,-83805,62042,91213,-14606,
4394,-562,74913,10406,96810,-61595,32564,31640,-9732,42058,98052,-7908,-72330,15
58,-80301,34878,32900,3939,-8824,88316,20937,21566,-3218,-66080,-31620,86859,542
89,90476,-42889,-15016,-18838,75456,30159,-67101,42328,-92703,85850,-5475,23470,
-80806,68206,17764,88235,46421,-41578,74005,-81142,80545,20868,-1560,64017,83784
,68863,-97516,-13016,-72223,79630,-55692,82255,88467,28007,-34686,-69049,-41677,
88535,-8217,68060,-51280,28971,49088,49235,26905,-81117,-44888,40623,74337,-2466
2,97476,79542,-72082,-35093,98175,-61761,-68169,59697,-62542,-72965,59883,-64026
,-37656,-92392,-12113,-73495,98258,68379,-21545,64607,-70957,-92254,-97460,-6343
6,-8853,-19357,-51965,-76582,12687,-49712,45413,-60043,33496,31539,-57347,41837,
67280,-68813,52088,-13155,-86430,-15239,-45030,96041,18749,-23992,46048,35243,-7

```
9450,85425,-58524,88781,-39454,53073,-48864,-82289,39086,82540,-11555,25014,-543
1,-39585,-89526,2705,31953,-81611,36985,-56022,68684,-27101,11422,64655,-26965,-
63081,-13840,-91003,-78147,-8966,41488,1988,99021,-61575,-47060,65260,-23844,-21
781,-91865,-19607,44808,2890,63692,-88663,-58272,15970,-65195,-45416,-48444,-782
26,-65332,-24568,42833,-1806,-71595,80002,-52250,30952,48452,-90106,31015,-22073
,62339,63318,78391,28699,77900,-4026,-76870,-45943,33665,9174,-84360,-22684,-168
32,-67949,-38077,-38987,-32847,51443,-53580,-13505,9344,-92337,26585,70458,-5276
4,-67471,-68411,-1119,-2072,-93476,67981,40887,-89304,-12235,41488,1454,5355,-34
855,-72080,24514,-58305,3340,34331,8731,77451,-64983,-57876,82874,62481,-32754,-
39902,22451,-79095,-23904,78409,-7418,77916]
# a = sorted([-1, 0, 1, 2, -1, -4, -1])
# a = [3,0,-2,-1,1,2]
[-2, -1, 0, 1, 2, 3]
# a = [0,0,0,0]
# a = [-1,0,1]
s = Solution()
# print(len(a))
print(s.threeSum(a))
```

Array/TopKFrequentElements.py

```
"""
Given a non-empty array of integers, return the k most frequent elements.
Example 1:
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
Example 2:
Input: nums = [1], k = 1
Output: [1]
Note:
You may assume k is always valid, 1 ≤ k ≤ number of unique elements.
Your algorithm's time complexity must be better than O(n log n), where n is the
array's size.
思路:
首先是计算出每个数字出现的频率，之后排序频率，输出最高的k位。
字典（哈希表），sorted（排序）。
beat 88%
测试用例:
https://leetcode.com/problems/top-k-frequent-elements/description/
"""
class Solution(object):
    def topKFrequent(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: List[int]
        """
        nums_dict = {}
        for i in nums:
            try:
                nums_dict[i] += 1
            except KeyError:
                nums_dict[i] = 1
        return sorted(nums_dict, key=lambda x: nums_dict[x], reverse=True)[:k]
```

Array/Triangle.py

```
"""
Given a triangle, find the minimum path sum from top to bottom. Each step you may
move to adjacent numbers on the row below.
For example, given the following triangle
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).
Note:
Bonus point if you are able to do this using only O(n) extra space, where n is
the total number of rows in the triangle.
从底到顶，min(self+right)。
O(1) 空间。
beat
99%
测试地址：
https://leetcode.com/problems/triangle/description/
"""
class Solution(object):
    def minimumTotal(self, triangle):
        """
        :type triangle: List[List[int]]
        :rtype: int
        """
        for i in range(len(triangle)-2, -1, -1):
            length = len(triangle[i])
            for j in range(length):
                _right = triangle[i+1][j+1]
                _self = triangle[i+1][j]
                mins = min(_right, _self)
                triangle[i][j] += mins
        return min(triangle[0])
```

Array/TwoSumIIAlreadySorted.py

```
"""
Given an array of integers that is already sorted in ascending order, find two
numbers such that they add up to a specific target number.
The function twoSum should return indices of the two numbers such that they add
up to the target, where index1 must be less than index2.
Note:
Your returned answers (both index1 and index2) are not zero-based.
You may assume that each input would have exactly one solution and you may not
use the same element twice.
Example:
Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.
Two Sum 已经排序后的输出，直接可以O(n):
解释传送门:
TwoSum.
ThreeSum.
"""
class Solution(object):
    def twoSum(self, sortedNums, target):
        """
        :type numbers: List[int]
        :type target: int
        :rtype: List[int]
        """
        start = 0
        end = len(sortedNums) - 1
        while start <= end:
            # print(nums[start] + nums[end])
            if sortedNums[start] + sortedNums[end] == target:
                return start+1, end+1
            if sortedNums[start] + sortedNums[end] > target:
                end -= 1
            else:
                start += 1
```

Array/two_sum.py

```
"""
Given nums = [2, 7, 11, 15], target = 9,
Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
数组中两个数相加，得到目标。
排序后的数组查找最快的就是二分法了。
基本思路：
排序，
二分查找。
用目标逐个相减然后查找是否存在（其实可以顺便返回位置，这样就不用index了。）。
所以时间复杂度 O(nlogn)
改进：
有O(n) 方式。
2018/8/17：
O(n) 方式总结：
在有序状态下，从后向前是逐渐缩小，从前向后逐渐增大。
[-4, -1, 0, 1, 2]
 0             4
当 start + end < target 时，end已经不能在增大了，只能增大start。
当 start + end > target 时，start不能在缩小了，只能缩小end。
一直到找到 target 或 start 与 end 相遇结束。
测试数据：
https://leetcode.com/problems/two-sum/description/
"""
try:
    range = xrange
except NameError:
    range = range
class Solution(object):
    def twoSum(self, nums, target):
        result = self._twoSum(nums, target)
        a = nums.index(result[0])
        # b = nums.index(result[-1], a+1)
        for i in range(len(nums)-1, -1, -1):
            if nums[i] == result[1]:
                return [a, i]
                # b = i
                # break
        # return [a, b]
    def _twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        sortedNums = sorted(nums)
        start = 0
        end = len(sortedNums) - 1
        while start <= end:
            # print(nums[start] + nums[end])
            if sortedNums[start] + sortedNums[end] == target:
                return sortedNums[start], sortedNums[end]
            if sortedNums[start] + sortedNums[end] > target:
                end -= 1
```

```
            else:
                start += 1
    # def binarySearch(self, rawList, target):
    #     split = len(rawList) // 2
    #     left = rawList[:split]
    #     right = rawList[split:]
    #     if not left and not right:
    #         return None
    #     if left and left[-1] == target:
    #         return True
    #     if right and right[0] == target:
    #         return True
    #     if len(left) > 1 and left[-1] > target:
    #         return self.binarySearch(left, target)
    #     if len(right) > 1 and right[0] < target:
    #         return self.binarySearch(right, target)
    # def twoSum(self, nums, target):
    #     """
    #     :type nums: List[int]
    #     :type target: int
    #     :rtype: List[int]
    #     """
    #     sortedNums = sorted(nums)
    #     for i, data in enumerate(sortedNums):
    #         if self.binarySearch(sortedNums[:i]+sortedNums[i+1:], target-
data):
    #             result = sorted([nums.index(data), nums.index(target-data)])
    #             if result[0] == result[1]:
    #                 return [result[0], nums[result[0]+1:].index(target-
data)+result[0]+1]
    #             return result
            # elif target < 0 and data > target:
                # if self.binarySearch(nums[:i]+nums[i+1:], target-data):
                    # return sorted([i, nums[i+1:].index(target-data)+i+1])
```

Array/WordSubsets.py

```
"""
We are given two arrays A and B of words.  Each word is a string of lowercase
letters.
Now, say that word b is a subset of word a if every letter in b occurs in a,
including multiplicity.  For example, "wrr" is a subset of "warrior", but is not
a subset of "world".
Now say a word a from A is universal if for every b in B, b is a subset of a.
Return a list of all universal words in A.  You can return the words in any
order.
Example 1:
Input: A = ["amazon","apple","facebook","google","leetcode"], B = ["e","o"]
Output: ["facebook","google","leetcode"]
Example 2:
Input: A = ["amazon","apple","facebook","google","leetcode"], B = ["l","e"]
Output: ["apple","google","leetcode"]
Example 3:
Input: A = ["amazon","apple","facebook","google","leetcode"], B = ["e","oo"]
Output: ["facebook","google"]
Example 4:
Input: A = ["amazon","apple","facebook","google","leetcode"], B = ["lo","eo"]
Output: ["google","leetcode"]
Example 5:
Input: A = ["amazon","apple","facebook","google","leetcode"], B =
["ec","oc","ceo"]
Output: ["facebook","leetcode"]
Note:
1 <= A.length, B.length <= 10000
1 <= A[i].length, B[i].length <= 10
A[i] and B[i] consist only of lowercase letters.
All words in A[i] are unique: there isn't i != j with A[i] == A[j].
如果 B 中每一个元素出现的字符均在 A 中某一个元素中出现，则视为子单词，给一个A 和 B求子单词数
量。
思路：
哈希 A，
哈希 B。
B 哈希的时候要注意，有重复。
比如["ec","oc","ceo"]
只要有 c e o 各一次即可。不需要每个都判断一次。
测试地址：
https://leetcode.com/problems/word-subsets/description/
这个写法比较慢，前面的思路基本都是哈希。
"""
from collections import Counter
class Solution(object):
    def wordSubsets(self, A, B):
        """
        :type A: List[str]
        :type B: List[str]
        :rtype: List[str]
        """
        dict_A = {i:Counter(i) for i in A}
        dict_B = {}
        for i in B:
            c = Counter(i)
```

```
            for j in c:
                if c.get(j) > dict_B.get(j, 0):
                    dict_B[j] = c.get(j)
        result = []
        for i in dict_A:
            for j in dict_B:
                if dict_B[j] > dict_A[i].get(j):
                    break
            else:
                result.append(i)
        return result
```

Backtracking/GenerateParentheses.py

```
"""
Given n pairs of parentheses, write a function to generate all combinations of
well-formed parentheses.
For example, given n = 3, a solution set is:
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
生成有效的括号对。
关键字:
    递归
beat
100%
测试地址:
https://leetcode.com/problems/generate-parentheses/description/
"""
class Solution(object):
    def generateParenthesis(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        result = []
        def _generateParenthesis(x, y, parenthesis):
            if not x and not y:
                result.append(parenthesis)
                return
            if y > x:
                _generateParenthesis(x, y-1, parenthesis=parenthesis+')')
            if x:
                _generateParenthesis(x-1, y, parenthesis=parenthesis+'(')
        _generateParenthesis(n-1, n, '(')
        return result
```

Backtracking/subsets.py

```
"""
Given a set of distinct integers, nums, return all possible subsets (the power
set).
Note: The solution set must not contain duplicate subsets.
Example:
Input: nums = [1,2,3]
Output:
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
直接上递归，每条线都有两个决策：
1. 加上。
2. 不加。
beat 96%
测试地址：
https://leetcode.com/problems/subsets/description/
"""
class Solution(object):
    def subsets(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        result = []
        length = len(nums)
        def makeSubsets(index, current_subsets):
            if index == length:
                return
            result.append(current_subsets+[nums[index]])
            makeSubsets(index+1, current_subsets+[nums[index]])
            makeSubsets(index+1, current_subsets)
        makeSubsets(0, [])
        return result+[[]]
```

BFS/BinaryTreeLevelOrderTraversal.py

```
"""
Given a binary tree, return the level order traversal of its nodes' values. (ie,
from left to right, level by level).
For example:
Given binary tree [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
return its level order traversal as:
[
  [3],
  [9,20],
  [15,7]
]
迭代出二叉树层级别的节点值。
直接用 BFS 即可.
下面是一个用双端列表的迭代方式实现。
用递归更好写一些。
测试地址:
https://leetcode.com/problems/binary-tree-level-order-traversal/description/
beat 92% 28ms.
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
from collections import deque
class Solution(object):
    def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        result = []
        temp = deque([root])
        next_temp = deque()
        _result = []
        while 1:
            if temp:
                node = temp.popleft()
                _result.append(node.val)
                if node.left:
                    next_temp.append(node.left)
                if node.right:
                    next_temp.append(node.right)
            else:
                result.append(_result)
                _result = []
```

```
            temp = next_temp
            next_temp = deque()
        if not temp and not next_temp:
            if _result:
                result.append(_result)
            return result
```

BFS/BinaryTreeLevelOrderTraversalII.py

```
"""
Given a binary tree, return the bottom-up level order traversal of its nodes'
values. (ie, from left to right, level by level from leaf to root).
For example:
Given binary tree [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
return its bottom-up level order traversal as:
[
  [15,7],
  [9,20],
  [3]
]
无难度..做了1的话直接把返回结果倒置即可。
beat 94%
28ms
测试地址:
https://leetcode.com/problems/binary-tree-level-order-traversal-ii/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
from collections import deque
class Solution(object):
    def levelOrderBottom(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        result = []
        temp = deque([root])
        next_temp = deque()
        _result = []
        while 1:
            if temp:
                node = temp.popleft()
                _result.append(node.val)
                if node.left:
                    next_temp.append(node.left)
                if node.right:
                    next_temp.append(node.right)
            else:
                result.append(_result)
                _result = []
                temp = next_temp
                next_temp = deque()
```

```
            if not temp and not next_temp:
                if _result:
                    result.append(_result)
                return result[::-1]
```

BFS/BinaryTreeZigzagLevelOrderTraversal.py

```
"""
Given a binary tree, return the zigzag level order traversal of its nodes'
values. (ie, from left to right, then right to left for the next level and
alternate between).
For example:
Given binary tree [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
return its zigzag level order traversal as:
[
  [3],
  [20,9],
  [15,7]
]
与 Binary tree order traversal 非常相似，不同的是这个是一层 左->右，一层 右->左。
同样的思路，加一个标记，若是RIGHT的层就倒过来。
beat 99.94%.
24ms
测试地址：
https://leetcode.com/problems/binary-tree-zigzag-level-order-
traversal/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
from collections import deque
class Solution(object):
    def zigzagLevelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        result = []
        temp = deque([root])
        next_temp = deque()
        _result = []
        LEFT = True
        RIGHT = False
        currentDirection = LEFT
        while 1:
            if temp:
                node = temp.popleft()
                _result.append(node.val)
                if node.left:
                    next_temp.append(node.left)
                if node.right:
```

```
                next_temp.append(node.right)
            else:
                if currentDirection == LEFT:
                    result.append(_result)
                    currentDirection = RIGHT
                else:
                    _result.reverse()
                    result.append(_result)
                    currentDirection = LEFT
                _result = []
                temp = next_temp
                next_temp = deque()
            if not temp and not next_temp:
                if _result:
                    if currentDirection == LEFT:
                        result.append(_result)
                        currentDirection = RIGHT
                    else:
                        _result.reverse()
                        result.append(_result)
                return result
```

BFS/FindBottomLeftTreeValue.py

```
"""
Given a binary tree, find the leftmost value in the last row of the tree.
Example 1:
Input:
    2
   / \
  1   3
Output:
1
Example 2:
Input:
        1
       / \
      2   3
     /   / \
    4   5   6
       /
      7
Output:
7
返回最底层最左边的一个节点的值。
思路：
使用广度优先算法：
1. 也可以用深度优先算法，不过广度优先的话所需代码更少，更好理解。
2. 广度优先：
        1              遍历1
       / \
      2   3            遍历 2 3
     /   / \
    4   5   6          遍历 4 5 6
       /
      7                遍历 7
   深度优先：
        1              遍历1
       / \
      2   3            遍历2 -> 4
     /   / \
    4   5   6          遍历 3 -> 5 -> 7, 3 -> 6
       /
      7
3. 广度优先运用在此处是以「层」为概念的，遍历完一层，再遍历一层。
   由于是返回最左边，所以从右向左。
   用两个列表，一个存放本层所有节点，一个存放本层所有节点的下层所有节点。
   遍历完本层后合并存放下层的列表，如果没有节点则返回结果。
   遍历本层时遵从 right to left. 有值就保存为结果。
测试用例：
https://leetcode.com/problems/find-bottom-left-tree-value/description/
40ms beat 74%.
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
```

```
#         self.right = None
class Solution(object):
    def findBottomLeftValue(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        waiting_for_access = [root]
        new_waiting_for_access = []
        result = root.val
        while 1:
            while waiting_for_access:
                node = waiting_for_access.pop()
                if node.right:
                    new_waiting_for_access.append(node.right)
                    result = node.right.val
                if node.left:
                    new_waiting_for_access.append(node.left)
                    result = node.left.val
            if not new_waiting_for_access:
                return result
            waiting_for_access.extend(new_waiting_for_access[::-1])
            new_waiting_for_access = []
```

BFS/MaximumDepthOfBinaryTree.py

```
"""
Given a binary tree, find its maximum depth.
The maximum depth is the number of nodes along the longest path from the root
node down to the farthest leaf node.
Note: A leaf is a node with no children.
Example:
Given binary tree [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
return its depth = 3.
找到二叉树的最大深度。
思路：
按层进行广度优先搜索即可。
beat 99% 32ms.
测试地址：
https://leetcode.com/problems/maximum-depth-of-binary-tree/description/
"""
class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        else:
            left_height = self.maxDepth(root.left)
            right_height = self.maxDepth(root.right)
            return max(left_height, right_height) + 1
```

BFS/WordLadder.py

```
"""
Given two words (beginWord and endWord), and a dictionary's word list, find the
length of shortest transformation sequence from beginWord to endWord, such that:
Only one letter can be changed at a time.
Each transformed word must exist in the word list. Note that beginWord is not a
transformed word.
Note:
Return 0 if there is no such transformation sequence.
All words have the same length.
All words contain only lowercase alphabetic characters.
You may assume no duplicates in the word list.
You may assume beginWord and endWord are non-empty and are not the same.
Example 1:
Input:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -
> "cog",
return its length 5.
Example 2:
Input:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore no possible
transformation.
这个BFS有点...
一开始的思路就是 BFS，写法不同。
一开始的写法是每次都从 wordList 里找单词，结果是有一个case跑不通。
后来实在没思路学习 Discuss 里的内容，Discuss 里的 BFS 是每一个都生成26个新的单词，然后判断是
否在 wordList 中。
这种方法可以全部跑通...
可优化的点在于如何高效的找到存在于 _wordList 中可变换的值。
beat 66%
测试地址：
https://leetcode.com/problems/word-ladder/description/
"""
from collections import deque
class Solution(object):
    def ladderLength(self, beginWord, endWord, wordList):
        """
        :type beginWord: str
        :type endWord: str
        :type wordList: List[str]
        :rtype: int
        """
        if len(beginWord) == 1:
            return 2
        _wordList = set(wordList)
        result = deque([[beginWord, 1]])
        while result:
```

```
            word, length = result.popleft()
            if word == endWord:
                return length
            _length = length + 1
            for i in range(len(word)):
                for c in 'qwertyuiopasdfghjklzxcvbnm':
                    new = word[:i]+c+word[i+1:]
                    if new in _wordList:
                        _wordList.remove(new)
                        result.append([new, _length])
        return 0
```

Design/InsertDeleteGetRandomO(1).py

```
"""
Design a data structure that supports all following operations in average O(1)
time.
insert(val): Inserts an item val to the set if not already present.
remove(val): Removes an item val from the set if present.
getRandom: Returns a random element from current set of elements. Each element
must have the same probability of being returned.
Example:
// Init an empty set.
RandomizedSet randomSet = new RandomizedSet();
// Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomSet.insert(1);
// Returns false as 2 does not exist in the set.
randomSet.remove(2);
// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);
// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();
// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);
// 2 was already in the set, so return false.
randomSet.insert(2);
// Since 2 is the only number in the set, getRandom always return 2.
randomSet.getRandom();
设计一种结构，支持以下三种操作：
insert(val) 插入一个 val，若存在则返回 False，否则True.
remove(val) 从中删除一个 val，若不存在则返回 False，否则True.
getRandom() 返回其中的任意一个数。
以上操作均需在 O(1) 的时间复杂度内完成。
插入 O(1) 的结构一般会用 哈希表，链表，或者在列表尾部插入。
删除 O(1) 的话链表需要查找，即时用到二分也不是 O(1)，列表的话给定确定的下标查找是 O(1) 的。
根据上面的结论，哈希表肯定是要用了，添加删除和查找都是 O(1)，不过哈希表并不能进行 getRandom，
或者说不能以 o(1) 时间完成 getRandom。
而上面提到，列表的给定下标的查找是 O(1) 的，并且插入到列表尾的话也是 O(1) 的操作。
所以这里的思路是用两个结构：
在不考虑内存的情况下，
哈希表用于进行一般的查找删除，列表用于找随机。
1. 这样在添加时，哈希表的 key:value 为 val:index。
随后在列表尾添加 val.
2. 删除时，将要删除的元素的下标与列表的最后一个元素互换，删除列表尾和哈希表内的目标元素，
并将 `最后一个元素` 在 `哈希表`中的 `下标` 替换为原本要删除的那个元素的下标。
这里要注意，写的时候若是 先从哈希表中取出目标的下标，然后在从列表尾取出最后一个元素添加的话，要判
断下是否是同一个元素，若是同一个则可以不进行替换。
beat:
90%
测试地址：
https://leetcode.com/problems/insert-delete-getrandom-o1/description/
"""
import random
class RandomizedSet(object):
    def __init__(self):
        """
        Initialize your data structure here.
        """
```

```
        self.data_dict = {}
        self.data_list = []
        self.length = 0
    def insert(self, val):
        """
        Inserts a value to the set. Returns true if the set did not already
contain the specified element.
        :type val: int
        :rtype: bool
        """
        if self.data_dict.get(val) is not None:
            return False
        self.data_dict[val] = self.length
        self.length += 1
        self.data_list.append(val)
        return True
    def remove(self, val):
        """
        Removes a value from the set. Returns true if the set contained the
specified element.
        :type val: int
        :rtype: bool
        """
        if self.data_dict.get(val) is not None:
            x = self.data_dict.pop(val)
            y = self.data_list[-1]
            if y != val:
                self.data_dict[y] = x
            self.data_list[-1], self.data_list[x] = self.data_list[x],
self.data_list[-1]
            self.data_list.pop()
            self.length -= 1
            return True
        return False
    def getRandom(self):
        """
        Get a random element from the set.
        :rtype: int
        """
        return self.data_list[random.randint(0, self.length-1)]
# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.getRandom()
```

Design/LRUCache.py

```
"""
Design and implement a data structure for Least Recently Used (LRU) cache. It
should support the following operations: get and put.
get(key) - Get the value (will always be positive) of the key if the key exists
in the cache, otherwise return -1.
put(key, value) - Set or insert the value if the key is not already present. When
the cache reached its capacity, it should invalidate the least recently used item
before inserting a new item.
Follow up:
Could you do both operations in O(1) time complexity?
Example:
LRUCache cache = new LRUCache( 2 /* capacity */ );
cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // returns 1
cache.put(3, 3);    // evicts key 2
cache.get(2);       // returns -1 (not found)
cache.put(4, 4);    // evicts key 1
cache.get(1);       // returns -1 (not found)
cache.get(3);       // returns 3
cache.get(4);       // returns 4
LRU cache:
最近最少使用缓存。
维基百科：https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU
在capacity满后，有新数据加入时使用次数最低且最先被加入的一个会被删除。
我的思路是：
1. 给每一个数据带权重。
1.1  capacity 未满时：
1.1.1 若是不重复的数据则直接添加，{key: {value, weight}}
1.1.2 若已重复则更新权重。
2. 根据权重在满数据时删除。
2.1 数据已满之后：
2.1.1 若是已重复的则更新权重。
2.1.2 若是未重复的则删除权重最低的一个，并将新数据添加 {key: {value, weight}}。
权重问题：
每次有更新，新增，权重就会增加1，无上限。
---
第一版实现，并通过测试：
https://leetcode.com/problems/lru-cache/description/
效率还可以。beat 25% 左右。
"""
class LRUCache(object):
    def __init__(self, capacity):
        """
        :type capacity: int
        """
        self.lru_cache = {}
        self.lru_cache_number = {}
        self.current_cache_number = 0
        self.capacity = capacity
    def get(self, key):
        """
        :type key: int
        :rtype: int
```

```
        """
        return self._get(key)
    def _get(self, key):
        # if inexistent then return -1
        # or add weight and return value
        value = self.lru_cache.get(key)
        if not value:
            return -1
        else:
            self._add_exist_key_weight(key)
            return value.get('value')
    def _add_exist_key_weight(self, key):
        # remove raw weight and add new weight
        data = self.lru_cache.get(key)
        raw_weight = data.get('weight')
        data['weight'] = self.current_cache_number
        self.lru_cache_number.pop(raw_weight)
        self.lru_cache_number[self.current_cache_number] = key
        self.current_cache_number += 1
    def put(self, key, value):
        """
        :type key: int
        :type value: int
        :rtype: void
        """
        # check capacity
        # True will run clear
        # Then put data.
        is_reached_capacity = self.check_cache_capacity()
        if not is_reached_capacity:
            if self.lru_cache.get(key):
                self._replace(key, value)
            else:
                self._put(key, value)
        else:
            if self.lru_cache.get(key):
                self._replace(key, value)
            else:
                self._put_and_remove(key, value)
    def _put(self, key, value):
        """
            {
                key: {'value': value,
                      'weight': self.current_cache_number}
            }
            {
                weight: key
            }
        """
        self.lru_cache[key] = {'value': value, 'weight':
self.current_cache_number}
        self.lru_cache_number[self.current_cache_number] = key
        self.current_cache_number += 1
    def _put_and_remove(self, key, value):
        # remove the least key and put the new key.
        min_weight = min(self.lru_cache_number)
        self.lru_cache.pop(self.lru_cache_number[min_weight])
        self.lru_cache_number.pop(min_weight)
```

```
            self._put(key, value)
        def _replace(self, key, value):
            """
                replace the existed key to new value and weight.
            """
            raw_data = self.lru_cache.get(key)
            raw_weight = raw_data.get('weight')
            raw_data['value'] = value
            raw_data['weight'] = self.current_cache_number
            self.lru_cache_number.pop(raw_weight)
            self.lru_cache_number[self.current_cache_number] = key
            self.current_cache_number += 1
        def check_cache_capacity(self):
            """
                True is reached capacity.
                False is not.
            """
            if len(self.lru_cache) == self.capacity:
                return True
            return False
    # Your LRUCache object will be instantiated and called as such:
    # obj = LRUCache(capacity)
    # param_1 = obj.get(key)
    # obj.put(key,value)
```

Design/RLEIterator.py

```
"""
Write an iterator that iterates through a run-length encoded sequence.
The iterator is initialized by RLEIterator(int[] A), where A is a run-length
encoding of some sequence.  More specifically, for all even i, A[i] tells us the
number of times that the non-negative integer value A[i+1] is repeated in the
sequence.
The iterator supports one function: next(int n), which exhausts the next n
elements (n >= 1) and returns the last element exhausted in this way.  If there
is no element left to exhaust, next returns -1 instead.
For example, we start with A = [3,8,0,9,2,5], which is a run-length encoding of
the sequence [8,8,8,5,5].  This is because the sequence can be read as "three
eights, zero nines, two fives".
Example 1:
Input: ["RLEIterator","next","next","next","next"], [[[3,8,0,9,2,5]],[2],[1],[1],
[2]]
Output: [null,8,8,5,-1]
Explanation:
RLEIterator is initialized with RLEIterator([3,8,0,9,2,5]).
This maps to the sequence [8,8,8,5,5].
RLEIterator.next is then called 4 times:
.next(2) exhausts 2 terms of the sequence, returning 8.  The remaining sequence
is now [8, 5, 5].
.next(1) exhausts 1 term of the sequence, returning 8.  The remaining sequence is
now [5, 5].
.next(1) exhausts 1 term of the sequence, returning 5.  The remaining sequence is
now [5].
.next(2) exhausts 2 terms, returning -1.  This is because the first term
exhausted was 5,
but the second term did not exist.  Since the last term exhausted does not exist,
we return -1.
Note:
0 <= A.length <= 1000
A.length is an even integer.
0 <= A[i] <= 10^9
There are at most 1000 calls to RLEIterator.next(int n) per test case.
Each call to RLEIterator.next(int n) will have 1 <= n <= 10^9.
实现一个RLE。
什么是RLE呢：
111222333
会被压缩为
3个1 3个2 3个3这样的。
题目是给出的压缩后的，next会返回第x个数，不够的话返回-1。
直接迭代然后判断数量即可。
测试地址：
https://leetcode.com/contest/weekly-contest-101/problems/rle-iterator/
contest，运行了76ms。
"""
class RLEIterator(object):
    def __init__(self, A):
        """
        :type A: List[int]
        """
        self.a = A
    def next(self, n):
```

```
        """
        :type n: int
        :rtype: int
        """
        for i in range(0, len(self.a), 2):
            if self.a[i] > 0:
                if self.a[i] - n >= 0:
                    self.a[i] -= n
                    return self.a[i+1]
                else:
                    t = self.a[i]
                    self.a[i] -= n
                    n -= t
        return -1
# Your RLEIterator object will be instantiated and called as such:
# obj = RLEIterator(A)
# param_1 = obj.next(n)
```

DFS/PathSum.py

```
"""
Given a binary tree and a sum, determine if the tree has a root-to-leaf path such
that adding up all the values along the path equals the given sum.
Note: A leaf is a node with no children.
Example:
Given the below binary tree and sum = 22,
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \      \
7    2      1
return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.
给一颗二叉树和一个值，找到从根到叶的所有路径的和中是否有一个与给定的值相当。
因为只要有一个就可以了，所以直接用深度优先，最差是 O(n)。
测试用例：
https://leetcode.com/problems/path-sum/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root:
            return False
        def helper(prev, root, sum):
            if prev + root.val == sum:
                if not root.left and not root.right:
                    return True
            if root.left:
                if helper(prev + root.val, root.left, sum):
                    return True
            if root.right:
                if helper(prev + root.val, root.right, sum):
                    return True
            return False
        return helper(0, root, sum)
```

DFS/WordSearch.py

```
"""
Given a 2D board and a word, find if the word exists in the grid.
The word can be constructed from letters of sequentially adjacent cell, where
"adjacent" cells are those horizontally or vertically neighboring. The same
letter cell may not be used more than once.
Example:
board =
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
Given word = "ABCCED", return true.
Given word = "SEE", return true.
Given word = "ABCB", return false.
给定一个2D数组，找到是否该字符串存在于其中。
首先进行了一次失败的尝试....
没读第二行，直接用二叉树做搜索，理想情况下就是O(nlogn)。
发现没通过，要 「相邻」 的字符才可以。
那这可以用DFS深度优先。找到合适的点就一直深入下去，如果找到了就返回True。
如果没找到就找其他相邻的还有没有，没有就False有就继续寻找下去，直到word耗尽。
测试用例：
https://leetcode.com/problems/word-search/description/
"""
'''python
class TreeNode(object):
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
        self.count = 1
class binarySearchTree(object):
    """
        二叉搜索树，
        它的性质是左边节点都小于根节点，右边的都大于根节点。
        而且一般来说它是不存在重复元素的。
    """
    def __init__(self, root):
        if isinstance(root, TreeNode):
            self.root = root
        else:
            self.root = TreeNode(root)
    def add(self, value):
        # 从顶点开始遍历，找寻其合适的位置。
        root = self.root
        if self.searchAndAddOne(value):
            return
        while 1:
            if root.val < value:
                if root.right is None:
                    # if self.search(value):
                    #     break
                    root.right = TreeNode(value)
                    break
```

```
                else:
                    root = root.right
                    continue
            if root.val > value:
                if root.left is None:
                    # if self.search(value):
                    #     break
                    root.left = TreeNode(value)
                    break
                else:
                    root = root.left
                    continue
            if root.val == value:
                root.count += 1
                break
    def searchAndAddOne(self, value):
        # 查找一个值是否存在于这颗树中,如果存在则计数+1
        return self._search(self.root, value, add=True)
    def searchAndReduceOne(self, value):
        return self._search(self.root, value, reduce=True)
    def _search(self, root, value, add=False, reduce=False):
        if root.val == value:
            if add:
                root.count += 1
            if reduce:
                print(root.count, root.val)
                if root.count > 0:
                    root.count -= 1
                else:
                    return False
            return True
        if root.right:
            if root.val < value:
                return self._search(root.right, value, add=add, reduce=reduce)
        if root.left:
            if root.val > value:
                return self._search(root.left, value, add=add, reduce=reduce)
        return False
class Solution(object):
    def exist(self, board, word):
        """
        :type board: List[List[str]]
        :type word: str
        :rtype: bool
        """
        myTree = binarySearchTree(TreeNode('-'))
        for i in board:
            for j in i:
                myTree.add(j)
        for i in word:
            if not myTree.searchAndReduceOne(i):
                return False
        return True
a = Solution()
print(a.exist([['A']*2], "AAA"))
'''
"""
```

这一个的思路是从第一个开始，寻找其上下左右，如果有合适的则进入那个继续寻找，直到找到合适的或耗尽。
此版本TTL.
多次失败的尝试后，终于找到快速的方法。
思路仍然是DFS。之前写的时候，使用了deepcopy，每次都会生成一个全新的地图，这样做的坏处是每次生成一个新的地图都会耗费大量时间和空间。
经过测试，一个大点的地图（最下面的那个例子）生成100次会花费大约400ms。
改进后的思路是不在生成新的地图，只在原地图上捣鼓。
思路是这样：
如果此点与word中第一个相同，则将其覆盖。之后进行上下左右移动，有一个命中就会一直判断，如果全不命中就会收回。再从上一个点开始判断。
覆盖的原因是如果不替换那么可能形成环。但一个字符只允许使用一次。

FRANCE
0 1 2 3 4 5 6
"F" "Y" "C" "E" "N" "R" "D"
list
0 1 2 3 4 5 6
"K" "L" "N" "F" "I" "N" "U"
list
0 1 2 3 4 5 6
"A" "A" "A" "R" "A" "H" "R"
list
0 1 2 3 4 5 6
"N" "D" "K" "L" "P" "N" "E"
list
0 1 2 3 4 5 6
"A" "L" "A" "N" "S" "A" "P"
list
0 1 2 3 4 5 6
"O" "O" "G" "O" "T" "P" "N"
list
0 1 2 3 4 5 6
"H" "P" "O" "L" "A" "N" "O"

第一个 F 0, 0处命中，然后继续上下左右找有没有R命中的。结果没有。那么被替换的F就会还原。
第二个 F 2, 3处命中，然后继续上x，下命中，则替换掉然后继续判断。直到A处判断了FRA但断开了，
这时会还原A，但R的左边还有A同样进行判断但之后也断开了。到R上下左右都没有则还原回F，上下左右又没有则还原到初始。
"""

```
import copy
import pdb
class Solution(object):
    def exist(self, board, word):
        """
        :type board: List[List[str]]
        :type word: str
        :rtype: bool
        """
        for i, d in enumerate(board):
            for i2, d2 in enumerate(d):
                if d2 == word[0]:
                    if self.search(board, i2, i, word):
                        return True
        return False
```

```
    def search(self, board, x, y, word):
        """
        """
        if not word:
            return True
        # if x == len(board[0][0]) and y == len(board):
        #     return False
        if board[y][x] != word[0]:
            return False
        if board[y][x] == word[0]:
            word = word[1:]
        temp_data = board[y][x]
        board[y][x] = '#'
        # if not word:
            # return True
        # up
        if y-1 >= 0:
            if board[y-1][x] != '#':
                if self.search(board, x, y-1, word): # u
                    return True
        # down
        if y+1 < len(board):
            if board[y+1][x] != '#':
                if self.search(board, x, y+1, word): #d
                    return True
        # left
        if x-1 >= 0:
            if board[y][x-1] != '#':
                if self.search(board, x-1, y, word): #l
                    return True
        # right
        if x+1 < len(board[0]):
            if board[y][x+1] != '#':
                if self.search(board, x+1, y, word): # r
                    return True
        board[y][x] = temp_data
        return False
        # return
        # if not word:
        #     return True
        # # up
        # if y-1 >= 0:
        #     if board[y-1][x] == word[0]:
        #         temp_board = board
        #         # temp_board = copy.deepcopy(board)
        #         temp_board[y-1][x] = '#'
        #         if self.search(temp_board, x, y-1, word[1:]):
        #             return True
        # # down
        # if y+1 < len(board):
        #     if board[y+1][x] == word[0]:
        #         temp_board = board
        #         #
        #         # temp_board = copy.deepcopy(board)
        #         temp_board[y+1][x] = '#'
        #         if self.search(temp_board, x, y+1, word[1:]):
        #             return True
        # # left
```

```
        # if x-1 >= 0:
        #     if board[y][x-1] == word[0]:
        #         temp_board = board
        #         # temp_board = copy.deepcopy(board)
        #         temp_board[y][x-1] = '#'
        #         if self.search(temp_board, x-1, y, word[1:]):
        #             return True
        # # right
        # if x+1 < len(board[0]):
        #     if board[y][x+1] == word[0]:
        #         temp_board = board
        #         # temp_board = copy.deepcopy(board)
        #         temp_board[y][x+1] = '#'
        #         if self.search(temp_board, x+1, y, word[1:]):
        #             return True
        # return False
"""
改进一下，还是用DFS，不过这次不会在预先去寻找入口。
结果更慢。
"""
# class Solution(object):
#     def exist(self, board, word):
#         """
#         :type board: List[List[str]]
#         :type word: str
#         :rtype: bool
#         """
#         return self.search(board, 0, 0, word)
#     def search(self, board, x, y, word):
#         """
#         """
#         # return
#         if not word:
#             return True
#         # if x == len(board[0][0]) and y == len(board):
#         #     return False
#         if board[y][x] == word[0]:
#             word = word[1:]
#         board[y][x] = '#'
#         # if not word:
#             # return True
#         # up
#         if y-1 >= 0:
#             if board[y-1][x] != '#':
#                 temp_board = copy.deepcopy(board)
#                 if self.search(temp_board, x, y-1, word):
#                     return True
#         # down
#         if y+1 < len(board):
#             if board[y+1][x] != '#':
#                 temp_board = copy.deepcopy(board)
#                 if self.search(temp_board, x, y+1, word):
#                     return True
#         # left
#         if x-1 >= 0:
#             if board[y][x-1] != '#':
#                 temp_board = copy.deepcopy(board)
#                 if self.search(temp_board, x-1, y, word):
```

```
#                     return True
#         # right
#         if x+1 < len(board[0]):
#             if board[y][x+1] != '#':
#                 temp_board = copy.deepcopy(board)
#                 if self.search(temp_board, x+1, y, word):
#                     return True
#         return False
"""
2018/10/25
刷 Word Search II 时重新回顾了一下。
用这次刷II的方法重新测试，
beat
84%
测试地址：
https://leetcode.com/problems/word-search/description/
练习是有效果的。 ^_^
"""
class Solution(object):
    def exist(self, board, word):
        """
        :type board: List[List[str]]
        :type word: str
        :rtype: bool
        """
        def find(board, x, y, word):
            if not word:
                return True
            if board[y][x] == word[0]:
                raw = board[y][x]
                board[y][x] = 0
                if len(word) == 1:
                    board[y][x] = raw
                    return True
            else:
                return False
            # up
            if y-1 >= 0 and board[y-1][x] == word[1]:
                if find(board, x, y-1, word[1:]):
                    board[y][x] = raw
                    return True
            # down
            if y+1 < len(board) and board[y+1][x] == word[1]:
                if find(board, x, y+1, word[1:]):
                    board[y][x] = raw
                    return True
            # left
            if x-1 >= 0 and board[y][x-1] == word[1]:
                if find(board, x-1, y, word[1:]):
                    board[y][x] = raw
                    return True
            # right
            if x+1 < len(board[0]) and board[y][x+1] == word[1]:
                if find(board, x+1, y, word[1:]):
                    board[y][x] = raw
                    return True
            board[y][x] = raw
            return False
```

```
        for i in range(len(board)):
            for j in range(len(board[0])):
                if board[i][j] == word[0]:
                    if find(board, j, i, word):
                        return True
        return False
```

DFS/WordSearchII.py

```
"""
Given a 2D board and a list of words from the dictionary, find all words in the
board.
Each word must be constructed from letters of sequentially adjacent cell, where
"adjacent" cells are those horizontally or vertically neighboring. The same
letter cell may not be used more than once in a word.
Example:
Input:
words = ["oath","pea","eat","rain"] and board =
[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]
Output: ["eat","oath"]
Note:
You may assume that all inputs are consist of lowercase letters a-z.
I 的升级版。
I 中只需要找一个单词是否存在。
II 中升级为找很多单词。
直接扩展 I 的话对重复比较多的单词会TLE，
比如 aaaa,aaaab,aaaac,aaaae ...
这样的都有共同的前缀，aaaa，基于这个条件，可以用前缀树来将它们整合在一起，省去重复搜索 aaaa 的
步骤。
前缀树可以用字典（哈希表）来模拟。
```

```
trie = {}
for w in words:
    # 从根字典开始。
    t = trie
    # leetcode
    for l in w:
        # 若当前前缀不在这个前缀树字典中，则创建为一个新的前缀树字典。
        # 'l' not in t
        # t -> {'l': {}
        # }
        if l not in t:
            t[l] = {}
        # 下一次循环的为 'l' 下的 'e'
        # 所以替换为 t['l']
        t = t[l]
    # 这里添加一个标记，表示当前前缀树可以是末尾。
    # 如 leet 和 leetcode 可以在 leet 处结束。
    t['mark'] = 'mark'
```

```
之后的操作基本一样，只是把字符串换成了字典。
这里有个有趣的点，下面的写法会有重复的数据进去，
如果结果一开始用 set，添加时可以直接去重，效率是比不过先用 list 再用 set 去重的。
beat:
46%
测试地址：
```

```
https://leetcode.com/problems/word-search-ii/description/
前面的思路都是同一种思路，也就是可以优化。
"""
class Solution(object):
    def findWords(self, board, words):
        """
        :type board: List[List[str]]
        :type words: List[str]
        :rtype: List[str]
        """
        result = []
        words = set(words)
        trie = {}
        for i in words:
            t = trie
            for x in i:
                if x not in t:
                    t[x] = {}
                t = t[x]
            t['!'] = '!'
        def find(board, x, y, word, pre):
            # print(word)
            if '!' in word:
                result.append(pre)
            for w in word:
                raw = board[y][x]
                board[y][x] = 0
                # up
                if y-1 >= 0 and board[y-1][x] == w:
                    find(board, x, y-1, word[w], pre+w)
                # down
                if y+1 < len(board) and board[y+1][x] == w:
                    find(board, x, y+1, word[w], pre+w)
                # left
                if x-1 >= 0 and board[y][x-1] == w:
                    find(board, x-1, y, word[w], pre+w)
                # right
                if x+1 < len(board[0]) and board[y][x+1] == w:
                    find(board, x+1, y, word[w], pre+w)
                board[y][x] = raw
        maps = {}
        for i in range(len(board)):
            for j in range(len(board[0])):
                try:
                    maps[board[i][j]].append((i,j))
                except:
                    maps[board[i][j]] = [(i,j)]
        for i in trie:
            if maps.get(i):
                xy = maps.get(i)
                for j in xy:
                    find(board, j[1], j[0], trie[i], i)
        return list(set(result))
```

DP/BitwiseORsOfSubarray.py

```
"""
We have an array A of non-negative integers.
For every (contiguous) subarray B = [A[i], A[i+1], ..., A[j]] (with i <= j), we
take the bitwise OR of all the elements in B, obtaining a result A[i] | A[i+1] |
... | A[j].
Return the number of possible results.  (Results that occur more than once are
only counted once in the final answer.)
Example 1:
Input: [0]
Output: 1
Explanation:
There is only one possible result: 0.
Example 2:
Input: [1,1,2]
Output: 3
Explanation:
The possible subarrays are [1], [1], [2], [1, 1], [1, 2], [1, 1, 2].
These yield the results 1, 1, 2, 1, 3, 3.
There are 3 unique values, so the answer is 3.
Example 3:
Input: [1,2,4]
Output: 6
Explanation:
The possible results are 1, 2, 3, 4, 6, and 7.
对子数组进行或运算，最后结果是有多少个唯一的解。
思路是DP：
走的弯路：
一开始写的：
[1, 1, 2, 2, 4]
A[0] = {1}
基于A[0]，判断是否在A[0]里，不在的话在添加，在的话就继承A[0]。
A[1] = {1}
A[2] = {1, 2, 3}
A[3] = {1, 2 ,3}
运行到这里都没什么错误，因为就碰巧进行了一次相邻的或运算。
A[4] = {1, 2, 3, 4, 5, 6, 7}
到了这里就有了错误，4不应该与这么多进行或运算。
这里就不知道怎么做了，如果要把上一次的结果也加到里面，怎么才能保证所进行的或运算不包含不相邻的两个点如：
[1, 2, 4]
不会进行 [1,4]的运算。
重新的梳理应该是：
[1]
A[0] = {1}
[   1] [1, 1]
A[1] = {1}
注意，这里与上一个进行或运算，但不把上一个也存到A[2]里面，
[        2] [1, 1, 2] [   1, 2]
A[2] = {2, 3}
基于上一个，但不会将上一个的结果加到本次里影响最终运算。
---
最终输出结果时，进行一次全部的set整理。
测试地址：
```

```
https://leetcode.com/contest/weekly-contest-100/problems/bitwise-ors-of-
subarrays/
Accepted.
"""
class Solution(object):
    def subarrayBitwiseORs(self, A):
        """
        :type A: List[int]
        :rtype: int
        """
        if not A:
            return 0
        dp = [{A[0]}]
        for i in range(1, len(A)):
            new = {A[i]}
            for j in dp[i-1]:
                new.add(j|A[i])
            dp.append(new)
        return len(set.union(*dp))
```

DP/ClimbingStairs.py

```
"""
You are climbing a stair case. It takes n steps to reach to the top.
Each time you can either climb 1 or 2 steps. In how many distinct ways can you
climb to the top?
Note: Given n will be a positive integer.
Example 1:
Input: 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
Example 2:
Input: 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
给定一个阶梯数，每次可以上一步，也可以上两步，输出有爬到顶有几种方式。
和 decode ways 非常相似的思路，刚解出 decode ways...这个相当于买一赠一了。
思路:
 只有一层时，只有一种方式，一步。
 只有两层是，有两种方式，一步一步，两步。
 有三层时，有三种方式，一步一步一步，一步两步，两步一步。
 ...
 3 层时的子问题来自于 2层时的子问题与1层时的子问题。
beat 97%
测试地址:
https://leetcode.com/problems/climbing-stairs/description/
"""
class Solution(object):
    def climbStairs(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n < 1:
            return 0
        if n == 1:
            return 1
        dp = []
        dp.append(1)
        dp.append(2)
        for i in range(2, n):
            dp.append(dp[i-1]+dp[i-2])
        return dp[-1]
```

DP/CoinChange.py

```
"""
You are given coins of different denominations and a total amount of money
amount. Write a function to compute the fewest number of coins that you need to
make up that amount. If that amount of money cannot be made up by any combination
of the coins, return -1.
Example 1:
Input: coins = [1, 2, 5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
Example 2:
Input: coins = [2], amount = 3
Output: -1
Note:
You may assume that you have an infinite number of each kind of coin.
思路是DP:
1. 一次失败的尝试:
子问题定位 当前钱数需要的最少硬币量。
代码写的有点问题，导致运行异常缓慢。
外层循环是 1-amount ，逐个点去寻找。
内层循环则一遍遍重复与已经解出来的点进行对比，这样做包含了很多无用的信息。
2. 经过思考后，发现问题所在，
[1, 2, 5] 11
7这个点，所需要的不是从 1-6 都进行一遍判断后取最小值。
只需要2,5,6这三个点就可以。
具体是 7-5,7-2,7-1。
---
这样就是一个经典的DP算法。这个算法可以通过，但有时也会 TLE...
---
二次优化:
外层循环大可不必 1 - amount，可以 min(coins) - amount。
这样可以提高一些效率。
---
三次尝试:
可以把外层循环与内层循环调换。这样的效率同样是 O(n*amount)。
测试地址:
https://leetcode.com/problems/coin-change/description/
beat 33%...
"""
class Solution(object):
    def coinChange(self, coins, amount):
        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """
        dp = {0: 0}
        mins = min(coins)
        inf = float('inf')
        for i in range(mins, amount+1):
            dp[i] = min((dp.get(i-j, inf)+1 for j in coins))
        # for i in coins:
        #     for j in range(i, amount+1):
        #         dp[j] = min(dp.get(j, inf), dp.get(j-i, inf)+1)
        # for i in range(mins, amount+1):
```

```
            # dp[i] = min((dp.get(i-j, inf)+1 for j in coins))
            # temp = []
            # for j in coins:
            #     temp.append(dp.get(i-j, inf)+1)
            # dp[i] = min(temp)
        if dp.get(amount) == inf:
            return -1
        return dp.get(amount, -1)
        # coins = set(coins)
        # dp = [(0, 0)]
        # for i in range(1, amount+1):
            # temp = []
            # for j in range(len(dp)-1, -1, -1):
            #     j = dp[j]
            #     if i - j[1] in coins:
            #         temp.append((j[0]+1, i))
            # if temp:
                # dp.append(min(temp, key=lambda x: x[0]))
        # print(dp)
        # if dp[-1][1] == amount:
            # return dp[-1][0]
        # return -1
a = Solution()
# print(a.coinChange([70, 71], 142))
print(a.coinChange([70,177,394,428,427,437,176,145,83,370], 7582))
```

DP/DecodeWays.py

```
"""
A message containing letters from A-Z is being encoded to numbers using the
following mapping:
'A' -> 1
'B' -> 2
...
'Z' -> 26
Given a non-empty string containing only digits, determine the total number of
ways to decode it.
Example 1:
Input: "12"
Output: 2
Explanation: It could be decoded as "AB" (1 2) or "L" (12).
Example 2:
Input: "226"
Output: 3
Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
定义 'A-Z' 分别为 '1-26'。
给定一个包含数字的字符串，给出有几种可能的解码办法。
思路：
1. 首先是一个递归的搜索思路，用递归总是很容易想到...
   每次都判断前一个字符和前两个字符是否在 1-26 之中，在的话就分别再开启一次递归。
   到是不会出错，就是效率低，TLE 了。
2. 看了下 Discuss 里的提点：
   使用 Dp。
   Dp 的思考：
       Dp 的子问题是：
           当前这个点能与组成几个。
       那么如何解出该点的子问题呢？
           我一开始的思考是这样：
               "1221"
               dp[0] = 1  1
               dp[1] = 2  1,2  12
               dp[2] = 3  1,2,2 12,2               1,22
               dp[3] = 5  1,2,2,1 12,2,2 1,22,2    1,2,21 12, 21
               每一层的dp都包含一下上一次结果,后面是一位数的部分，若组合起来 < 27 即可组合
成...
               不过这个不好弄，而且虽然效率会比递归高，但...也高不到哪里去，肯定不是 Discuss
中的解法。
               emmm，想了一会没想到好办法，去 Discuss 里看了下思路，一下就明白了...
               多的这个组合恰好和 i-2 的进行了组合，这个的子问题解法是基于前面两个子问题的。
       注意处理下 0。
beat 99%.
测试用例：
https://leetcode.com/problems/decode-ways/description/
"""
class Solution(object):
    def numDecodings(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0
```

```
        if len(s) == 1:
            if 0 < int(s) < 10:
                return 1
            return 0
        letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        letters_dict = {str(i):letters[i-1] for i in range(1, 27)}
        dp = []
        if 0 < int(s[0]) < 10:
            dp.append(1)
        else:
            dp.append(0)
        if 10 <= int(s[0]+s[1]) < 27:
            if 0 < int(s[1]) < 10:
                dp.append(2)
            else:
                dp.append(1)
        else:
            if s[1] != '0':
                dp.append(dp[0])
            else:
                dp.append(0)
        for i in range(2, len(s)):
            x = 0
            if s[i] != '0':
                x += dp[i-1]
            if s[i-1] != '0':
                if '10' <= s[i-1] + s[i] < '27':
                    x += dp[i-2]
            dp.append(x)
        return dp[-1]
#         self.result = 0
#         def makeDecode(rest_str):
#             if not rest_str:
#                 self.result += 1
#                 # result.append(1)
#                 return
#             if len(rest_str)>=2:
#                 if rest_str[:2] in letters_dict:
#                     makeDecode(rest_str[2:])
#             if rest_str[:1] in letters_dict:
#                 makeDecode(rest_str[1:])
#         if len(s)>=2:
#             if s[:2] in letters_dict:
#                 makeDecode(s[2:])
#         if s[:1] in letters_dict:
#             makeDecode(s[1:])
#         return self.result
```

DP/FindAllAnagramsInAString.py

```
"""
Given a string s and a non-empty string p, find all the start indices of p's
anagrams in s.
Strings consists of lowercase English letters only and the length of both strings
s and p will not be larger than 20,100.
The order of output does not matter.
Example 1:
Input:
s: "cbaebabacd" p: "abc"
Output:
[0, 6]
Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".
Example 2:
Input:
s: "abab" p: "ab"
Output:
[0, 1, 2]
Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".
给定一个字符串 s 和 非空字符串 p。 找到所有 p 在 s 中的变位词，将变位词开始部分的索引添加到结
果中。
字符串包含的字符均为小写英文字母。
结果的顺序无关紧要。
思路：
看到标的是个 easy .. 估计数据不会很大，直接 sorted(s[i:i+len(p)]) == sorted(p) 判断了..
结果是 TLE。
好吧，只能认真思考下了。
想到的是利用动态规划，还有点回溯的味道。
子问题为：
当前点加上之前的点是否覆盖了全部的p。
若全部覆盖了，则返回 i - len(p) + 1 的开始索引，同时，将 s[i - len(p) + 1] 处的字符添加到
dp 的判断点中。
这样做可以扫描到这种情况：
s = abababab
p = ab
循环这种。
若没有存在于最后一个dp中，那么先判断是否有重复：
1. 如果在 p 中，但不在上一个子问题的需求解里。
 2. 如果与最前面的一个字符发生了重复。
   3. 如果与之前的一个字符不同。
满足这三个条件那么可以无视这个重复的字符。
比如这样：
s = abac
p = abc
这样相当于把重复的一个 a 忽略掉了。
同时还要判断是否它的前一个是否一致，一致的话就不能单纯的忽略了。
s = abaac / aabc
p = abc
没有满足的上述三个条件，那么重新复制一份原始子问题，然后将当前字符判断一下，继续下一个循环即可。
beat 99% 84ms
```

测试地址：
https://leetcode.com/problems/find-all-anagrams-in-a-string/description/
2018/10/27 修正：
上次分析后是有不足的，可以AC只是因为这个题的测试用例很少。
问题出在上面分析的 2. 和 3. 中，
重新梳理下：
若没有存在于最后一个dp中，那么先判断是否有重复：
1. 如果在 p 中，但不在上一个子问题的需求解里。
  2. 此时要找一个合适的 ·复活点·，直接上例子吧
s "cbdbcae"
   cbd   b 当出现第二个 b 时，b不与之前分析后所找到的 c 相同，那么可以判定为要从 b 这个点开始复活了，但如果是的话那就找不到了。
   要复活的点是 从 b 之前的一个 d。 也就是复活点要在重复的那个字符后面开始，那么我们要做的就是把在它之前的字符给加回来。
p "abcde"
待学习 Discuss 滑动窗口。
"""
```
class Solution(object):
    def findAnagrams(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: List[int]
        """
        # p = sorted(p)
        if not s:
            return []
        _p = {}
        for i in p:
            try:
                _p[i] += 1
            except:
                _p[i] = 1
        result = []
        x = _p.copy()
        if s[0] in _p:
            x[s[0]] -= 1
            if not x[s[0]]:
                x.pop(s[0])
        dp = x
        if not dp:
            return [i for i in range(len(s)) if s[i] == p]
        for i in range(1, len(s)):
            if s[i] in dp:
                t = dp
                t[s[i]] -= 1
                if not t[s[i]]:
                    t.pop(s[i])
                if not t:
                    result.append(i-len(p)+1)
                    x = {}
                    _t = s[i-len(p)+1]
                    x[_t] = 1
                    dp = x
            else:
                if s[i] in _p:
                    if s[i] != s[i-len(p)+sum(dp.values())]:
                        for t in s[i-len(p)+sum(dp.values()):i]:
```

```
                            if t == s[i]:
                                break
                            try:
                                dp[t] += 1
                            except:
                                dp[t] = 1
                    continue
                x = _p.copy()
                if s[i] in x:
                    x[s[i]] -= 1
                    if not x[s[i]]:
                        x.pop(s[i])
                dp = x
        return result
# 下面这个代码有瑕疵。
class Solution(object):
    def findAnagrams(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: List[int]
        """
        # p = sorted(p)
        if not s:
            return []
        _p = {}
        for i in p:
            try:
                _p[i] += 1
            except:
                _p[i] = 1
        result = []
        x = _p.copy()
        if s[0] in _p:
            x[s[0]] -= 1
            if not x[s[0]]:
                x.pop(s[0])
        dp = x
        if not dp:
            return [i for i in range(len(s)) if s[i] == p]
        for i in range(1, len(s)):
            if s[i] in dp:
                t = dp
                t[s[i]] -= 1
                if not t[s[i]]:
                    t.pop(s[i])
                if not t:
                    result.append(i-len(p)+1)
                    x = {}
                    _t = s[i-len(p)+1]
                    x[_t] = 1
                    dp = x
            else:
                if s[i] in _p and s[i] != s[i-1] and s[i] == s[i-
len(p)+sum(dp.values())]:
                    continue
                x = _p.copy()
                if s[i] in x:
```

```
                    x[s[i]] -= 1
                    if not x[s[i]]:
                        x.pop(s[i])
                dp = x
        return result
```

DP/FlipStringToMonotoneIncreasing.py

```
"""
A string of '0's and '1's is monotone increasing if it consists of some number of
'0's (possibly 0), followed by some number of '1's (also possibly 0.)
We are given a string S of '0's and '1's, and we may flip any '0' to a '1' or a
'1' to a '0'.
Return the minimum number of flips to make S monotone increasing.
Example 1:
Input: "00110"
Output: 1
Explanation: We flip the last digit to get 00111.
Example 2:
Input: "010110"
Output: 2
Explanation: We flip to get 011111, or alternatively 000111.
Example 3:
Input: "00011000"
Output: 2
Explanation: We flip to get 00000000.
Note:
1 <= S.length <= 20000
S only consists of '0' and '1' characters.
给定一个由 '0' 和 '1' 组成的字符串，把它变成单调递增的字符串。返回最少的改变次数。
思路：
从左向右走过一遍，把找到的 1 变成 0。
从右向左过一遍，把找到的 0 变成 1。
最后过一遍，找到相加最少的一个点即可（可能不止一个）。
测试地址：
https://leetcode.com/contest/weekly-contest-107/problems/flip-string-to-monotone-
increasing/
"""
class Solution(object):
    def minFlipsMonoIncr(self, S):
        """
        :type S: str
        :rtype: int
        """
        x = [0] if S[0] == '0' else [1]
        # left to right
        # 1 -> 0
        for i in range(1, len(S)):
            if S[i] == '1':
                x.append(x[-1]+1)
            else:
                x.append(x[-1])
        # right to left
        # 0 -> 1
        S = S[::-1]
        y = [0] if S[0] == '1' else [1]
        for i in range(1, len(S)):
            if S[i] == '0':
                y.append(y[-1]+1)
            else:
                y.append(y[-1])
        y.reverse()
```

```
        return min([i+j for i,j in zip(x,y)]) - 1
```

DP/HouseRobber.py

```
"""
You are a professional robber planning to rob houses along a street. Each house
has a certain amount of money stashed, the only constraint stopping you from
robbing each of them is that adjacent houses have security system connected and
it will automatically contact the police if two adjacent houses were broken into
on the same night.
Given a list of non-negative integers representing the amount of money of each
house, determine the maximum amount of money you can rob tonight without alerting
the police.
Example 1:
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
Example 2:
Input: [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5
(money = 1).
             Total amount you can rob = 2 + 9 + 1 = 12.
你是专业强盗计划偷钱。每个房子都有钱，不能连续偷，否则会触发警报，计划能偷的最多的钱，要偷哪些房子。
用Dp的子问题思路：
不能连续的偷：
1 和 2只能偷这两个，没得选择。
3 能顺带偷1。
4 则需要在 1 和 2 中抉择。
5 则需要在 2 和 3 中抉择。
所以
子问题则是每个点能偷到的最大数。
初始化3个
例
[2,7,9,3,1]
dp = [2,7,11(9+2)]
max(3+7, 3+2)
dp = [2,7,11(9+2),10(3+7)]
max(1+11,1+7)
dp = [2,7,11(9+2),10(3+7),12(1+11)]
可以中途记录，也可以最后max，当然中途记录的话效率的最高的。
就这个测试来说max也没差。
测试地址：
https://leetcode.com/problems/house-robber/description/
beat 99% 20ms
"""
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        if len(nums) <= 2:
            return max(nums)
```

```
        robber = [nums[0], nums[1], nums[2] + nums[0]]
        for i in range(3, len(nums)):
            robber.append(max(nums[i] + robber[i-2], nums[i] + robber[i-3]))
        return max(robber)
```

DP/HouseRobberII.py

```
"""
You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.
Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.
Example 1:
Input: [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2),
             because they are adjacent houses.
Example 2:
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
与1不同的是这个首尾相连，是个环。
   1
 3  2
  1
只有3户人家时只能抢一户。
只有4户人家时能抢其中的两户。
5户以上时可以应用1时的规则，不过这次不要把最后一户算进来。
进行两次，一次以 0 为首，第二次以 -1 为首。
两次即可包含所有结果了。
效率 O(n)。
beat 100% 20ms
测试地址:
https://leetcode.com/problems/house-robber-ii/description/
"""
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        if len(nums) <= 3:
            return max(nums)
        if len(nums) <= 4:
            return max(max(nums), nums[1]+nums[-1], nums[2]+nums[0])
        robber = [nums[0], nums[1], nums[2]+nums[0]]
        for i in range(3, len(nums)-1):
            robber.append(max(nums[i] + robber[i-2], nums[i] + robber[i-3]))
        maxes = max(robber)
        nums = [nums[-1]] + nums
        nums.pop()
        robber = [nums[0], nums[1], nums[2]+nums[0]]
        for i in range(3, len(nums)-1):
```

```
            robber.append(max(nums[i] + robber[i-2], nums[i] + robber[i-3]))
        return max(robber+[maxes])
```

DP/InterleavingString.py

```
"""
Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.
Example 1:
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
Output: true
Example 2:
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"
Output: false
经过s1和s2的不断削减组合，最终是否能形成s3.
在最初使用了搜索的递归方法后进行学习DP以及改良，最终beat 85%以上。
思路：
最开始的思路就是搜索，符合条件就开启一个递归，符合条件就开启一个递归，毫不意外的TTL。
当时以为自己用的DP，怎么还会超时了呢。后来学习一下之后，发现自己只是用了一个搜索，而不是DP。
DP的主要任务时找到子问题，通过解决子问题来解决上层问题。还有就是子问题可以有多种解时，保留下来的子问题是哪个。
知道了这个之后，此问题的子问题是 「是否可以组成当前的字符串」。
例：
s1 = "aabcc"
s2 = "dbbca"
s3 = "aadbbcbcac"
s3_0 = s1[0] or s2[0]
s3_1 = for i in s3_0 if s1[1] + i or if s2[0] + i
...
这样直至最后，看是否能够找到，如果其中的一环中出现了空，那么肯定是组不成了。
经过初版的测试后，发现比递归的搜索还要慢很多很多。甚至出现了MLE（内存占用太多）。
最终定位到原因是含有非常多同样的数据。
如这个测试用例：
s1 = aabc s2 = abad s3 = aab....
在判断s3_0时会加入两个可能的结果
分别是s1 = abc s2 = abad 和 s1 = aabc s2 = bad
这样在s3_1时就会有多个同样的数据
s1 = abc s2 = bad 从s3_0[0][1] 也就是s2中取a.
s1 = abc s2 = bad 从s3_1[0][0] 也就是s1中取a.
这两个最终结果相同，对下一次结果没有影响，只取一个即可。
这仅仅是一个小例子，如果这样的数量过多，会形成非常大的冗余数据，只要出现20次这样的情况就会让最终的列表变成100多w个，而且一个很小的数据量即可达到20次。
但这100多w个所导致的下个结果是同一种，所以我们根本不需要这么多，只保留一个即可。
再思考一下，最初只定位了子问题，没有考虑保留哪些子问题的解，而是全部留下来。现在将缺少的「保留」条件加上。
所以在组成数据时判断是否已经重复即可解决此问题。
这样每次的迭代量变成1+个，基本不会超过4个。
但也有例外，如果 s1 == s2, s3 == s1+s2直接用不加判断的话会耗费大量时间，
如下：
print(Solution().isInterleave('c'*1000, 'c'*1000, 'c'*2000))
大概要半分钟。
因为此时我们制定的子问题规则毫无用处。每一个都可以避开。
或者
s1 == s2, s3 == s1与s2的前一部分相同的部分相加 + 后一部分相同的部分相加。
如 'ccb' 'ccb' 'ccccbb'
Leetcode 中的测试数据并不存在这样的情况。
如果遇到此种情况，可以用DFS来结合，确定一个子问题，一条路走到底，可以完美解决此问题。
不是这种情况下，也可以用DFS结合使用，不过效率差不多，没有此种情况也不需要再重写改写了。
测试用例：
https://leetcode.com/submissions/detail/165737688/
```

```
"""
'''
此版本 TTL MLE。
class Solution(object):
    def isInterleave(self, s1, s2, s3):
        """
        :type s1: str
        :type s2: str
        :type s3: str
        :rtype: bool
        """
        """
            recursive.
        """
        if len(s3) != len(s1) + len(s2):
            return False
        if not s3:
            return True
        dp = {}
        # s3 index, s1, s2
        for i, d in enumerate(s3):
            if i == 0:
                temp = []
                if s1 and s1[0] == d:
                    temp.append((s1[1:], s2))
                if s2 and s2[0] == d:
                    temp.append((s1, s2[1:]))
                dp[str(i)] = temp
                continue
            temp = []
            if dp[str(i-1)]:
                for j in dp[str(i-1)]:
                    s1, s2 = j[0], j[1]
                    if s1 and s1[0] == d:
                        temp.append((s1[1:], s2))
                    if s2 and s2[0] == d:
                        temp.append((s1, s2[1:]))
                    dp[str(i)] = temp
            else:
                return False
            # print(dp)
        try:
            for i in dp[str(len(s3)-1)]:
                if not any(i):
                    return True
            else:
                return False
        except KeyError:
            return False
'''
class Solution(object):
    def isInterleave(self, s1, s2, s3):
        """
        :type s1: str
        :type s2: str
        :type s3: str
        :rtype: bool
        """
```

```
        """
            recursive.
        """
        if len(s3) != len(s1) + len(s2):
            return False
        if not s3:
            return True
        dp = {}
        # s3 index, s1, s2
        for i, d in enumerate(s3):
            if i == 0:
                temp = []
                if s1 and s1[0] == d:
                    temp.append((s1[1:], s2))
                if s2 and s2[0] == d:
                    temp.append((s1, s2[1:]))
                dp[str(i)] = temp
                continue
            temp = []
            if dp[str(i-1)]:
                for j in dp[str(i-1)]:
                    s1, s2 = j[0], j[1]
                    if s1 and s1[0] == d:
                        if (s1[1:], s2) not in temp:
                            temp.append((s1[1:], s2))
                    if s2 and s2[0] == d:
                        if (s1, s2[1:]) not in temp:
                            temp.append((s1, s2[1:]))
                    dp[str(i)] = temp
            else:
                return False
            # print(len(dp[str(i-1)]))
            del dp[str(i-1)]
            # print(dp)
        try:
            for i in dp[str(len(s3)-1)]:
                if not any(i):
                    return True
            else:
                return False
        except KeyError:
            return False
print(Solution().isInterleave('c'*500+'d', 'c'*500+'d', 'c'*1000+'dd'))
```

DP/longesSubsequence.py

```
"""
整体思路是使用 DP。
DP 的核心是找到子问题。
该问题的子问题当前从此点向前与哪些组合可以形成最长的子串。
但与初级的抛硬币不同（要判断的只有3个），判断的点为从此点向前的所有点（子问题）。
时间复杂度 O(n**2)。
测试用例：
https://leetcode.com/problems/longest-increasing-subsequence/description/
另：有一个 O(nlogn) 的算法，暂未研究。
"""
"""
牛刀小试，DP 抛硬币，1 3 5 最少凑11。
def coins(num):
    coins_result = {'0': 0}
    for i in range(1, num+1):
        coins_5, coins_3, coins_1 = float('inf'), float('inf'), float('inf')
        if i-5 >= 0:
            coins_5 = coins_result[str(i-5)] + 1
        if i-3 >= 0:
            coins_3 = coins_result[str(i-3)] + 1
        if i-1 >= 0:
            coins_1 = coins_result[str(i-1)] + 1
        coins_result[str(i)] = min([coins_5, coins_3, coins_1])
    print(coins_result)
    print(coins_result[str(num)])
# coins(100)
"""
class Solution(object):
    def lengthOfLIS(self, strings):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not strings:
            return 0
        long_ss = []
        for i in strings:
            if not long_ss:
                long_ss.append(([i], 1))
                continue
            maxs = max(long_ss, key=lambda x: x[0][-1] < i and x[1]+1)
            if maxs[0][-1] >= i:
                long_ss.append(([i], 1))
            else:
                long_ss.append((maxs[0]+[i], maxs[1]+1))
        return max(long_ss, key=lambda x: x[1])[1]
```

DP/LongestContinuousIncreasingSubsequence.py

```
"""
Given an unsorted array of integers, find the length of longest continuous
increasing subsequence (subarray).
Example 1:
Input: [1,3,5,4,7]
Output: 3
Explanation: The longest continuous increasing subsequence is [1,3,5], its length
is 3.
Even though [1,3,5,7] is also an increasing subsequence, it's not a continuous
one where 5 and 7 are separated by 4.
Example 2:
Input: [2,2,2,2,2]
Output: 1
Explanation: The longest continuous increasing subsequence is [2], its length is
1.
Note: Length of the array will not exceed 10,000.
easy:
Dp, 子问题是前一个点所累积的数量。
测试地址:
https://leetcode.com/problems/longest-continuous-increasing-
subsequence/description/
今日的零启动算法。
"""
class Solution(object):
    def findLengthOfLCIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        dp = [1]
        for i in range(1, len(nums)):
            if nums[i] > nums[i-1]:
                dp.append(dp[i-1]+1)
            else:
                dp.append(1)
        return max(dp)
```

DP/MaxASubarrayLessThanMinBSubarray.py

```
"""
今日头条笔试第四题：
给两个长度为n的数组，a,b。
求有多少个 [l,r]，其中max(a[l:r])<min(b[l:r])。
思路：
DP
3 2 1
3 3 3
Dp1 = [(3), (2), (1)]
    ↓
      [(3), (2, 3), (1)]
    ↓
      [(3), (2, 3), (1, 2, 3)]
Dp2求的是最小。
然后求一下个数。
---
不优化还是挺慢的。
"""
import random
tests = (
# [
# [3, 2, 1],
# [3, 3, 3]
# ], # 3   2 < 3 1 < 3 (2, 1) < (3, 3)
# [
# [3, 3, 4 ,5 ,7],
# [2, 3, 5, 1, 2]
# ], # 1   4 < 5
[
[5, 6, 5, 3, 1, 8, 1, 1, 3, 5],
[7, 8, 7, 2, 1, 9, 5, 7, 8, 1]
],  # 7   5 < 7 6 < 8  5 < 7 (5, 6) < (7, 8) (5, 6, 5) < (7, 8, 7), (6, 5) < (8,
7), 8 < 9
[[random.randint(1, 1000000000) for i in range(100000)],
[random.randint(1, 1000000000) for i in range(100000)]]
)
# def maxALEminB(a, b):
#     dp1 = [[i] for i in a]
#     dp2 = [[i] for i in b]
#     for i in range(1, len(dp1)):
#         value = dp1[i][0]
#         for j in dp1[i-1]:
#             if j > value:
#                 dp1[i].append(j)
#             else:
#                 dp1[i].append(value)
#     for i in range(1, len(dp2)):
#         value = dp2[i][0]
#         for j in dp2[i-1]:
#             if j < value:
#                 dp2[i].append(j)
#             else:
#                 dp2[i].append(value)
#     result = 0
```

```
#     for i, j in zip(dp1, dp2):
#         for i2, j2 in zip(i, j):
#             if i2 < j2:
#                 result += 1
#     print(dp1)
#     print(dp2)
#     return result
# for i in tests:
#     print(maxALEminB(*i))
# 优化版
# 1e5 个随机数由原来的无结果变为 2.5s 可解。
# 在无重复的数据时时间复杂度会退化为 O(N²).
#
def maxALEminB(a, b):
    """
        思路与前一致，前一版的主要瓶颈在于有太多重复的数据，若可以将这些重复的数据统计到一起，可以极大的减少对比次数。
        [[3], [2], [1]]
        现在变为
        [[(3, 1)], [(2, 1)], [(1, 1)]]
        即缩小重复的数据的检测结果。
        例：
        [5, 6, 5, 3, 1, 8, 1, 1, 3, 5],
        [[5], [6, 6], [5, 6, 6], [3, 5, 6, 6], [1, 3, 5, 6, 6], [8, 8, 8, 8, 8, 8], ...]
        dp1的检测为，若当前的值大于之前中的值的某一个[-1:0]，从前到后是按升序排列的。
                    就从最初出现的那个地方截断，之前的值全部替换为此最大值，若没有一个大于的，则直接将之前的追加到当前的结果中。
        dp2的检测只将 大于 变为 小于。
        关于对比：
            每一组的个数可能不同，但总量是一致的。
            两组同时取出自己的一个数据，若b > a则结果中加上 b 和 a 中出现的次数较少的一个。并减去，开始下一轮，b或a中任意一个的出现次数为0时，
            取出相应的本组的下一个数据，同时为0时开始下一轮循环。
    """
    dp1 = [[[i, 1]] for i in a]
    dp2 = [[[i, 1]] for i in b]
    # [
    #  [
    #   [i, 1]
    #  ]
    # ]
    for i in range(1, len(dp1)):
        value = dp1[i][0]
        j = dp1[i-1]
        # 二次优化后提高了 0.2 s.
        if j[0][0] > value[0]:
            dp1[i].extend(j)
            continue
        for x in range(len(j)-1, -1, -1):
            x2 = j[x]
            if x2[0] <= value[0]:
                dp1[i] = [[value[0], sum([t[1] for t in j[:x+1]])+1]] + j[x+1:]
                break
    for i in range(1, len(dp2)):
        value = dp2[i][0]
        j = dp2[i-1]
        if j[0][0] <= value[0]:
```

```
            dp2[i].extend(j)
            continue
        for x in range(len(j)-1, -1, -1):
            x2 = j[x]
            if x2[0] >= value[0]:
                dp2[i] = [[value[0], sum([t[1] for t in j[:x+1]])+1]] + j[x+1:]
                break
    result = 0
    for i, j in zip(dp1, dp2):
        valueA = 0
        valueB = 0
        while 1:
            if not i and not j:
                break
            if valueA == 0:
                a = i.pop()
                valueA = a[1]
            if valueB == 0:
                b = j.pop()
                valueB = b[1]
            minx = min(valueA, valueB)
            if b[0] > a[0]:
                result += minx
            valueA -= minx
            valueB -= minx
    return result
# 可优化
# def maxALEminB(a, b):
#     dp1 = [[i] for i in a]
#     dp2 = [[i] for i in b]
#     result = 0
#     for i in range(1, len(dp1)):
#         value = dp1[i][0]
#         value2 = dp2[i][0]
#         if value < value2:
#             result += 1
#         for j in range(len(dp1[i-1])):
#             maxA = dp1[i-1][j] if dp1[i-1][j] > value else value
#             minB = dp2[i-1][j] if dp2[i-1][j] < value2 else value2
#             if maxA < minB:
#                 result += 1
#             dp1[i].append(maxA)
#             dp2[i].append(minB)
#     if dp1[0][0] < dp2[0][0]:
#         result += 1
#     return result
for i in tests:
    print(maxALEminB(*i))
```

DP/MaximumLengthOfPairChain.py

```
"""
You are given n pairs of numbers. In every pair, the first number is always
smaller than the second number.
Now, we define a pair (c, d) can follow another pair (a, b) if and only if b < c.
Chain of pairs can be formed in this fashion.
Given a set of pairs, find the length longest chain which can be formed. You
needn't use up all the given pairs. You can select pairs in any order.
Example 1:
Input: [[1,2], [2,3], [3,4]]
Output: 2
Explanation: The longest chain is [1,2] -> [3,4]
Note:
The number of given pairs will be in the range [1, 1000].
给定一组有n个数对的数组。每个数对中的第一个数总小于第二个数。
若某数组(c, d) 中的 c > (a, b) 中的 b那么这就可以是一个数对链。现在求最长的数对链。
可以是任意顺序的链接。
第一版思路：
Dp：
由于是任意顺序，要先经过排序，知道哪一个是最小的。
首先按照每个数对的 [1] 进行排序，因为 0总是比1要小的。
dp的子问题：
当前点可与前面的点组成的最大链。
最后输出最大的链。
效率是 O(n²)。
测试通过，beat 20% ~ TLE...
优化：
思考后，经过 [1] 的排序后，第一个肯定是最适合作为首节点，因为没有顺序问题，相当于建立堆一样。
那么剩下的就是在之后的里面挑一个[0]紧挨着第一个的，如此往复。
也就是说：
先进行[1]的排序，选出第一个。在进行[0]的排序，选出大于之前的那个的替换，然后截断再进行[1]的排
序。
原始：
[[1,2], [2,3], [3,4]]
进行[1]排序：
[[1,2], [2,3], [3,4]]
选出[1,2]
进行[0]的排序：
[[2,3], [3,4]]
选出 [3, 4]。无剩余项，结束。
更进一步：
进行第一次[1]的排序并取出最适合作为第一个的之后，要做的是取出最适合做第二个的。
1. 最适合做第二个的有一个条件，就是它的[0]必须大于第一个的[1]。
2. 选出第二个后，第三个的对比条件仍是第三个的[0]与第二个的[1]做比较。
3. 第二个与第三个一定也在 [1] 的顺序链中：
第三个(用3表示)可以与第二个(用2表示)结合，也可以与第一个(用1表示)结合，第二个只可以与第一个结
合。
1-2-3
2-3
1-3
1[0] < 1[1]
2[0] > 1[1]
2[1] > 2[0]
3[0] > 2[1]
所以
```

```
2[0] > 1[0]
3[0] > 2[0]
又因为 3[1]>2[1]>1[1]，所以2一定在1与3之间。
但1与3之间还有其他的数，这些数我们也用2表示，1与3中可能出现的另一种数是 2[0] < 1[1] 的，这种数可以直接抛弃，因为不符合要求。
ok.
基本算是beat 100%，发现前辈们也是用的这种方法。果然不是第一个想到的啊哈哈。
测试地址：
https://leetcode.com/problems/maximum-length-of-pair-chain/description/
"""
class Solution(object):
    def findLongestChain(self, pairs):
        """
        :type pairs: List[List[int]]
        :rtype: int
        """
        if not pairs:
            return 0
    # O (nlogn) + O(n)
#         x = sorted(pairs, key=lambda x: x[1])
#         mins = x[0]
#         maxes = 1
#         for i in x:
#             if i[0] > mins[1]:
#                 maxes += 1
#                 mins = i
#         return maxes
    # O(n²)
        pairs.sort(key=lambda x: x[1])
        dp = [1]
        currentMaxes = 0
        for i in range(1, len(pairs)):
            maxes = max([dp[j]+1 if pairs[i][0] > pairs[j][1] else 1 for j in
range(i)])
            dp.append(maxes)
            currentMaxes = max(maxes, currentMaxes)
        return currentMaxes
```

DP/NumberofLongestIncreasingSubsequence.py

```
"""
Given an unsorted array of integers, find the number of longest increasing
subsequence.
Example 1:
Input: [1,3,5,4,7]
Output: 2
Explanation: The two longest increasing subsequence are [1, 3, 4, 7] and [1, 3,
5, 7].
Example 2:
Input: [2,2,2,2,2]
Output: 5
Explanation: The length of longest continuous increasing subsequence is 1, and
there are 5 subsequences' length is 1, so output 5.
Note: Length of the given array will be not exceed 2000 and the answer is
guaranteed to be fit in 32-bit signed int.
思路：
Dp，
Dp子问题：
分成两个子问题：
第一个子问题是length，第二个是count，记录的话肯定是要记录最长最多的。
第一个子问题：
length:
每个单独的可以看做是1
[4,3,2,0,3,1,7,9,1,7,9]
[1,1,1,1,1,1,1,1,1,1,1]
初始化每个长度均为1。
若现在的num比之前的num大，长度上则在这个长度的基础上+1。
[4,3,2,0,3,1,7,9,1,7,9]
         3>2,3>0，最大都是2。
[1,1,1,1,2,    1,1,1,1,1,1]
...
第二个子问题：
count：
[4,3,2,0,3,1,7,9,1,7,9]
[1,1,1,1,1,1,1,1,1,1,1]
初始化每个count也都是1。
若现在的num比之前的num要大，若是第一次出现，则等于这个次数，否则加上这个次数。
[4,3,2,0,3,1,7,9,1,7,9]
         3>2，由于是第一次出现，数量变为2所记录的数量。
         3>0，第二次出现，数量加上0所记录的数量。
[1,1,1,1,2,    1,1,1,1,1,1]
关于如何判断是不是第一次出现：
首先全部初始化为1。若当前num大于之前num时，可以判断长度是否与之前的相等：
3>2时，此时应该为 (2, 3)，
2 所记录的长度数量为 (1, 1)
3 此时记录的长度数量为 (1, 1)
这时由于是第一次出现，3处长度应为2([2, 3])，但还未变化，所以可以由长度判断是否为第一次。
经过此轮循环后，3处所记录的长度数量应为：(2, 1)
3>0时，这时
0 所记录的长度数量为 (1, 1)
3 所记录的长度数量为 (2, 1)
这时可以直接加。
测试地址：
```

```
https://leetcode.com/problems/number-of-longest-increasing-
subsequence/description/
O(n^2) 题目最大只有2000个数据，勉强通过。
"""
class Solution(object):
    def findNumberOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        length = [1] * len(nums)
        count = [1] * len(nums)
        maxLength = 1
        for i in range(len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
                    if length[j] == length[i]:
                        count[i] = count[j]
                    if length[j]+1 == length[i]:
                        count[i] += count[j]
                    x = max(length[j]+1, length[i])
                    length[i] = x
                    maxLength = max(maxLength, x)
        return sum([count[i] for i,j in enumerate(length) if j == maxLength])
```

DP/PickCards.py

```
"""
貌似今日头条笔试？第三题。
两个人选卡牌，每张卡牌都有两个分值 x，y。选到卡牌时x加给自己，y加给团队。
给一组卡牌，求两个人选的卡牌 x 相等且y最大的情况。
思路：
根据 y 的分值先排序。
之后进行判断：
若里面的x相加是奇数直接跳过，在进行一次将此时下标排除的判断，最后都不通过将最小的一个y值的卡牌放入废弃池。
若里面的x相加是偶数：
    排序：
    1. 取末尾一个数，若大于目标（x相加/2）则返回 False。将最小的一个y值的卡牌放入废弃池。
    2. 等于直接返回True.
    3. 小于的情况减去。
    对剩余的进行Dp:
    每一个都有两种操作：
    1. 减去或不减。
    2. 大于的不减，小于的也可以不减。
    3. 直到有一个是0或都不是，返回 False。
---
判断还是这么判断，只不过不废弃了，直接用itertools生成从大到小生成数据，先倒序，然后len(x) - 2
这样直接返回只适用于没有负数的情况，若有负数需要全部计算后在进行判断。
此种方法属于暴力法...效率不高.
最差会有
n * (n-1) / (n - 1) + n * (n-1) / (n-2) ....次。
做完3道已经用了接近两个小时。。
"""
test = [
[3, 1],
[2, 2],
[1, 4],
[1, 4]
]
test2 = [
[7, 1],
[6, 2],
[5, 3],
[4, 4],
[3, 5],
[2, 6],
[1, 7],
[1, 7]
]
import itertools
def maxCardsScope(cards):
    cards = sorted(cards, key=lambda x: x[1], reverse=True)
    result_cards = {}
    currentCards = cards
    allCardsScopes = sum([x[0] for x in cards])
    if allCardsScopes % 2 == 0:
        if judgeEqual(cards, allCardsScopes / 2):
            return sum([x[1] for x in cards])
            # result_cards[sum([x[1] for x in cards])] = [cards]
    for x in range(len(cards)-1, 1, -1):
```

```
        for i in itertools.combinations(cards, x):
            temp = sum([t[0] for t in i])
            if temp % 2 != 0:
                continue
            if judgeEqual(i, temp / 2):
                return sum([t[1] for t in i])
def judgeEqual(cards, target):
    cards = sorted(cards, key=lambda x: x[0], reverse=True)
    if cards[0][0] == target:
        return True
    if cards[0][0] > target:
        return False
    if cards[0][0] < target:
        # init = target - cards[0][0]
        dp = {-1: [target]}
        for i, d in enumerate(cards):
            temp = dp[i-1].copy()
            for x in dp[i-1]:
                if x - d[0] == 0:
                    return True
                if x - d[0] > 0:
                    temp.append(x - d[0])
            dp[i] = temp
    return False
print(maxCardsScope(test2))
```

DP/ReversePairs.py

```
"""
Given an array nums, we call (i, j) an important reverse pair if i < j and
nums[i] > 2*nums[j].
You need to return the number of important reverse pairs in the given array.
Example1:
Input: [1,3,2,3,1]
Output: 2
Example2:
Input: [2,4,3,5,1]
Output: 3
Note:
1. The length of the given array will not exceed 50,000.
2. All the numbers in the input array are in the range of 32-bit integer.
如果索引 i < j 并且 nums[i] > 2 * nums[j]，那么就称它为牛波翻转对。
给一个数组，返回里面有几个牛波翻转对。
思路:
一个 Dp 思路，从后向前，解出每一个点若要达成牛波翻转对所需要的值是多少。
比如 1 ，要达成需要 2 以上。
    3，则需要 6 以上。
将这些放到一个列表里，然后来新值之后判断若将它插入这个列表的话所插入的位置（下标），
这个位置即为已经判断过的能与它达成牛波翻转对的位置。
用二分法可以达成 n log n 的查询，美中不足的是列表的插入仍需要 O(n)。
beat 17%
测试地址:
https://leetcode.com/problems/reverse-pairs/description/
"""
import bisect
class Solution(object):
    def reversePairs(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        _base = []
        result = 0
        for i in range(len(nums)-1,-1,-1):
            index = bisect.bisect_left(_base, nums[i])
            result += index
            bisect.insort_right(_base, nums[i]*2)
        return result
```

DP/SubarrayProductLessThanK.py

```
"""
Your are given an array of positive integers nums.
Count and print the number of (contiguous) subarrays where the product of all the
elements in the subarray is less than k.
Example 1:
Input: nums = [10, 5, 2, 6], k = 100
Output: 8
Explanation: The 8 subarrays that have product less than 100 are: [10], [5], [2],
[6], [10, 5], [5, 2], [2, 6], [5, 2, 6].
Note that [10, 5, 2] is not included as the product of 100 is not strictly less
than k.
Note:
0 < nums.length <= 50000.
0 < nums[i] < 1000.
0 <= k < 10^6.
思路 Dp，处理下 1 即可。 不考虑 0，nums[i] 不会为 0。
beat 19%
测试地址:
https://leetcode.com/problems/subarray-product-less-than-k/description/
可剪枝优化。
"""
class Solution(object):
    def numSubarrayProductLessThanK(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        dp = []
        result = 0
        start = 0
        for i in range(len(nums)):
            if nums[i] < k:
                result += 1
                dp = [nums[i]]
                start = i
                break
        for i in range(start+1, len(nums)):
            if nums[i] == 1 and nums[i] < k:
                dp.append(1)
                result += len(dp)
                continue
            new = []
            if nums[i] < k:
                result += 1
                new.append(nums[i])
            for j in dp:
                if j * nums[i] < k:
                    result += 1
                    new.append(j * nums[i])
            dp = new
        return result
```

DP/UniquePath.py

```
"""
A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).
The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).
How many possible unique paths are there?
Above is a 7 x 3 grid. How many possible unique paths are there?
Note: m and n will be at most 100.
Example 1:
Input: m = 3, n = 2
Output: 3
Explanation:
From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Right -> Down
2. Right -> Down -> Right
3. Down -> Right -> Right
Example 2:
Input: m = 7, n = 3
Output: 28
思路:
直接DP。子问题:
经过当前点的路径一共有多少条。
由于只能向右或者向下且不可返回，每一个点的路径可由左和右的点的路径数相加得来。
边界:
无则为0。
1 1 1
1 2 3
初始化一个 x * x 的列表，并将0, 0设置为1。
之后将每个点的路径数等于左+上。
效率为O(mn)。
2
数学方法暂且跳过。
测试地址:
https://leetcode.com/problems/unique-paths/description/
beat 100%.
"""
class Solution(object):
    def uniquePaths(self, m, n):
        """
        :type m: int
        :type n: int
        :rtype: int
        """
        _map = [[0 for _ in range(m)] for _ in range(n)]
        # _map[0][0] = 1
        for i in range(n):
            for j in range(m):
                x = _map[i-1][j] if i - 1 >= 0 else 0
                y = _map[i][j-1] if j - 1 >= 0 else 0
                if x + y == 0:
                    _map[i][j] = 1
                else:
```

```
                    _map[i][j] = x + y
        return _map[n-1][m-1]
```

DP/UniquePathII.py

```
"""
A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).
The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).
Now consider if some obstacles are added to the grids. How many unique paths would there be?
An obstacle and empty space is marked as 1 and 0 respectively in the grid.
Note: m and n will be at most 100.
Example 1:
Input:
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
Output: 2
Explanation:
There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right
与Unique Path 相比就是多了不通的路，这个和边界处理的效果是一样的。不过不通所使用的数字1，所以预处理一下。
其他的不同之处：
在1中会将 0 0 变为1，但如果此处也直接使用的话会造成下面情况的错误：
[[1]]时预处理为[['x']]，但随后如果不做处理会直接变为path为1，但结果是0。
同样的还有[[1,0]]，[[0,1]]
[0, 1]
[1, 0]
需要注意的点：
1. 只要判断开始/结束点为 1 可以直接返回 0。
2. 如果此点不通则不必进行此点的加减运算。
优化小结：
1. 尽量避免多个判断。
    if x + y == 0 and i + j == 0:
        _map[i][j] = 1
    else:
        _map[i][j] = x + y
每次都需要两次判断，
    if x + y != 0:
        _map[i][j] = x + y
    elif i + j == 0:
        _map[i][j] = 1
改成这样效果是一样的,但减少了每次所必须进行的判断数量。
改动后的效果提升明显：
第一个平均耗时 36ms。
第二个只有 24ms。
2. 先完成后优化。
不要想太多，先完成后在优化速度。
最开始的做法是 两个判断 + 预处理 1 + 只判断入口，走完了再判断出口。
跑通之后，发现可以直接判断出口是不是也为 1。
两个判断也可以缩为1~2个。
```

预处理1更是不必要。
但如果反过来，我不进行预处理 1，不先写两个判断。做起来是很难的，很多开荒的地方都是模糊的。
现在这个版本已经可以了，还可以优化的地方：
预处理第一横排和竖排，这样可以不需要边界判断，也就会减少3~4个判断数量。
测试地址：
https://leetcode.com/problems/unique-paths-ii/description/
beat 76%
"""

```python
class Solution(object):
    def uniquePathsWithObstacles(self, _map):
        """
        :type obstacleGrid: List[List[int]]
        :rtype: int
        """
        if _map[0][0] == 1:
            return 0
        if _map[-1][-1] == 1:
            return 0
        n, m = len(_map), len(_map[0])
        for i in range(n):
            for j in range(m):
                if _map[i][j] == 1:
                    _map[i][j] = 'x'
                    continue
                x = _map[i-1][j] if i - 1 >= 0 and _map[i-1][j] != 'x' else 0
                y = _map[i][j-1] if j - 1 >= 0 and _map[i][j-1] != 'x' else 0
                if x + y != 0:
                    _map[i][j] = x + y
                elif i + j == 0:
                    _map[i][j] = 1
        return _map[n-1][m-1]
    # def uniquePathsWithObstacles(self, _map):
    #     """
    #     :type obstacleGrid: List[List[int]]
    #     :rtype: int
    #     """
    #     if _map[0][0] == 1:
    #         return 0
    #     if _map[-1][-1] == 1:
    #         return 0
    #     for i, d in enumerate(_map):
    #         for j, d2 in enumerate(d):
    #             if d2 == 1:
    #                 _map[i][j] = 'x'
    #     n, m = len(_map), len(_map[0])
    #     for i in range(n):
    #         for j in range(m):
    #             if _map[i][j] == 'x':
    #                 continue
    #             x = _map[i-1][j] if i - 1 >= 0 and _map[i-1][j] != 'x' else 0
    #             y = _map[i][j-1] if j - 1 >= 0 and _map[i][j-1] != 'x' else 0
    #             if x + y == 0 and i + j == 0:
    #                 _map[i][j] = 1
    #             else:
    #                 _map[i][j] = x + y
    #     return _map[n-1][m-1]
```

DP/WordBreak.py

```
"""
Given a non-empty string s and a dictionary wordDict containing a list of non-
empty words, determine if s can be segmented into a space-separated sequence of
one or more dictionary words.
Note:
The same word in the dictionary may be reused multiple times in the
segmentation.
You may assume the dictionary does not contain duplicate words.
Example 1:
Input: s = "leetcode", wordDict = ["leet", "code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
Example 2:
Input: s = "applepenapple", wordDict = ["apple", "pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen
apple".
             Note that you are allowed to reuse a dictionary word.
Example 3:
Input: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]
Output: false
给一个非空字符串和一个包含单词的非空字典。判断是否能用字典里的单词组合成给定的字符串。
思路:
Dp:
从0开始，若此分隔存在于给定字典中，则可以断开。
s = "leetcode", wordDict = ["leet", "code"]
leetcode
  l e e t c o d e
T F F F F F F F
leet
s[0:0+4] in wordDict
s[0+4] = True
  l e e t c o d e
T F F F T F F F F
        当搜索到这里时会再次进行重复的搜索。
---
emmm, 写法待改进。
这个写法思路一样，不过效率会低。
beat 3%.
测试地址:
https://leetcode.com/problems/word-break/description/
"""
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: List[str]
        :rtype: bool
        """
        dp = [True] + [False] * len(s)
        for i in range(len(s)):
            for j in range(i+1):
                if dp[j] == True:
                    for x in wordDict:
```

```
                    if x == s[j:j+len(x)]:
                        dp[j+len(x)] = True
        return dp[-1]
```

Heap/FindMedianFromDataStream.py

```
"""
Median is the middle value in an ordered integer list. If the size of the list is
even, there is no middle value. So the median is the mean of the two middle
value.
For example,
[2,3,4], the median is 3
[2,3], the median is (2 + 3) / 2 = 2.5
Design a data structure that supports the following two operations:
void addNum(int num) - Add a integer number from the data stream to the data
structure.
double findMedian() - Return the median of all elements so far.
Example:
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
Follow up:
If all integer numbers from the stream are between 0 and 100, how would you
optimize it?
If 99% of all integer numbers from the stream are between 0 and 100, how would
you optimize it?
找出数据流中的中位数。
这个中位数要求的是排序后的中位数。
1.
    所以最暴力的方法即是：
    1. 每新加一个数据就放到列表尾，然后进行一次排序。
     1.1 当然如果用这个思路到是可以优化，用 O(n) 的搜索确定插入到哪里，插入到列表也是O(n)的操
作。
    2. 随后输出 这一步到是 O(1)。
    没写这个的代码，太暴力...
2.
    基于 1. 的改进，查找我们用二分来代替，这样查找可以降低到 O(log n)，但插入仍然是 O(n)。
    输出索引的话就是 O(1) 没什么好说的。
    写了这个的思路，出乎意料的是直接跑赢了 97% 的代码，不过随后几次平均变得低了。
    能跑赢这么多的原因也无非是测试数据过小，小到几乎可以忽略 O(n) 的插入耗时。
    这个代码的话很好写：
    一个用于放元素的列表：
    self.stream_data = []
    def addNum(self, num):
        bisect.insort_right(self.stream_data, num)
    def findMdian(self):
        '''
            判断奇偶。
        '''
        pass
3.
    Discuss 里的思路基本是用到了堆。这个技巧感觉很棒：
    堆的插入和查询都是 O(log n) 级别的，用堆即可克服列表插入的 O(n) 的耗时情况：
    基本骨架是：
        如果我们将一个数据分成左右两部分，那么左边最大和右边最小即为我们找中位的基础元素。
    left          right
    [1,2,3,       4,5,6]
    左边这个符合大顶堆，右边这个则是小顶堆。
```

```
    1. 建立一大一小两个堆：
        大顶堆用于存放左边的元素。
        小顶堆用于存放右边的元素。
    2. 要面对的问题是如何在新数据来临时插入：
       1. 若两个堆都为空，那么插入 left 即可。
       2. 若right为空，left不为空：
          >= left[0] 若新数据大于 left[0] 表明应该插入到right里。
          < left[0] 新数据小于 left[0] 应该插入到left，这时要先将left的0弹出放到right
里，在插入left。
       3. 都不为空时判断：
          长度相等时：
             > right[0] 表明不需要插入到 left 中，将 right[0] 弹出并插入left，然后插到
right 中即可。
             < right[0] 表明直接插入到 left 中即可。
          不等时：
             left多：
                不大于right[0]
                   left 弹出，加入到 right。
                   新加入的压入 left。
                大于：
                   直接加入 right
             right多：
                不大于right[0]:
                   直接加入 left.
                大于:
                   right 弹出，加入到 left。
                   新加入的加入到 right。
测试数据的效率不等：
100ms~500ms 都有跑过...
就巨大量的数据来说：
使用堆应该是最好的选择，每一个操作都是 O(log n) 与 O(1) 级别的。
进阶条件的思考：
1. 如果全部都是 0~100 之间的数据，完全可以建立一个哈希表，以数字为键，新加入的在它的数量上累
积，寻找中位也简单。
测试地址：
https://leetcode.com/problems/find-median-from-data-stream/description/
"""
# import bisect
import heapq
class MedianFinder(object):
    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stream_data_left = []
        heapq.heapify(self.stream_data_left)
        self.stream_data_right = []
        heapq.heapify(self.stream_data_right)
    def addNum(self, num):
        """
        :type num: int
        :rtype: void
        """
#         heapq.heappush(self.stream_data_right, num)
#         if len(self.stream_data_right) > len(self.stream_data_left):
#             pop = heapq.heappop(self.stream_data_right)
#             heapq.heappush(self.stream_data_left, -pop)
        if not self.stream_data_left:
```

```
            heapq.heappush(self.stream_data_left, -num)
            return
        if not self.stream_data_right:
            if num > -self.stream_data_left[0]:
                heapq.heappush(self.stream_data_right, num)
            else:
                pop = heapq.heappop(self.stream_data_left)
                heapq.heappush(self.stream_data_right, -pop)
                heapq.heappush(self.stream_data_left, -num)
            return
        if len(self.stream_data_right) == len(self.stream_data_left):
            if num > self.stream_data_right[0]:
                heapq.heappush(self.stream_data_right, num)
                pop = heapq.heappop(self.stream_data_right)
                heapq.heappush(self.stream_data_left, -pop)
            else:
                heapq.heappush(self.stream_data_left, -num)
        elif len(self.stream_data_left) > len(self.stream_data_right):
            if num > self.stream_data_right[0]:
                heapq.heappush(self.stream_data_right, num)
            else:
                heapq.heappush(self.stream_data_left, -num)
                pop = heapq.heappop(self.stream_data_left)
                heapq.heappush(self.stream_data_right, -pop)
        else:
            if num < self.stream_data_right[0]:
                heapq.heappush(self.stream_data_left, -num)
            else:
                heapq.heappush(self.stream_data_right, num)
                pop = heap.heappop(self.stream_data_right)
                heapq.heappush(self.stream_data_left, -pop)
        # bisect.insort_right(self.stream_data, num)
    def findMedian(self):
        """
        :rtype: float
        """
        length = len(self.stream_data_left) + len(self.stream_data_right)
        if length % 2 == 0:
            return float((-self.stream_data_left[0] +
self.stream_data_right[0])) / 2.0
        else:
            return -self.stream_data_left[0]
# Your MedianFinder object will be instantiated and called as such:
# obj = MedianFinder()
# obj.addNum(num)
# param_2 = obj.findMedian()
```

Heap/KthSmallestElementInASortedMatrix.py

```
"""
Given a n x n matrix where each of the rows and columns are sorted in ascending
order, find the kth smallest element in the matrix.
Note that it is the kth smallest element in the sorted order, not the kth
distinct element.
Example:
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,
return 13.
Note:
You may assume k is always valid, 1 ≤ k ≤ n2.
找到第k小个数。
测试用例：
https://leetcode.com/problems/kth-smallest-element-in-a-sorted-
matrix/description/
思路是用了堆：
Python 有内置的堆模块，需要进行研究自写。 2018/08/02待跟进。
堆：
堆是一个完全二叉树，完全二叉树是除了最底层，其他层都是铺满的。
                                    0
                  1                                   2
          3               4                   5               6
      7       8       9       10      11      12      13      14
    15 16   17 18   19 20   21 22   23 24   25 26   27 28   29 30
堆又分为最大堆与最小堆，最小堆是根节点是整个堆中最小的，最大堆则是最大的。
堆的操作分为：插入，取顶。
大部分情况下插入时的数据都是无序的，所以要保证最大堆与最小堆需要的操作肯定要有上浮与下沉。
上浮：
最小堆中：
如果父节点比自己大则自己上浮。
如果子节点比自己小则自己下沉。
也就是做数据交换，一直上浮或下沉到符合条件为止。
代码待添加...
"""
'''
内置heapq的第一个版本：
运行时间长，但并未TLE.passed...
import heapq
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        heap = []
        for i in matrix:
            for j in i:
                heapq.heappush(heap, j)
        for i in range(k-1):
```

```
            heapq.heappop(heap)
        return heap[0]
'''
'''
```

第二个版本：
使用sorted. 每次都根据每个列表的0进行排序，然后取出。直到k=0.
排序时间复杂度为平均为 O(log len(n))。需要进行k次，所以是 O(klog len(n))。
理想情况下应该是最快的，但在书写时sorted在数据量过大时都会重新生成数组，所以导致很慢。
改进版本是直接打散列表，然后一次性sorted,这样的时间复杂度为 O(nlogn),也很容易写，但不想这样。

```
# 1
```

原以为会TTL，居然没有，跑了1778ms，也是最慢的。

```
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        while k:
            # matrix = sorted(matrix, key=lambda x: x[0])
            matrix.sort(key=lambda x: x[0])
            pop = matrix[0][0]
            matrix[0] = matrix[0][1:]
            if not matrix[0]:
                matrix = matrix[1:]
            k -= 1
        return pop
'''
'''
```

基于第二版的改进：
第二版的性能瓶颈应该是在排序时会重新移动大量的元素，如果每次仅对len(n)个元素排序呢？
每次都会for 一遍 len(n)，生成一个新的 列表，列表中的元素是 ([0], index)，然后排序这个列表。
也就是说不在排序原列表，而是排序一个新的列表。
根据index去修正原列表。
这版本TTl.这样排序会导致时间复杂度为(k*len(n)*log len(n))

```
'''
'''
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        length = len(matrix)
        while k:
            # matrix = sorted(matrix, key=lambda x: x[0])
            # matrix.sort(key=lambda x: x[0])
            # 测试用例是Python 2.
            newMatrix = sorted([(matrix[i][0], i)for i in xrange(length)],
key=lambda x: x[0])
            pop = matrix[newMatrix[0][1]][0]
            matrix[0] = matrix[newMatrix[0][1]][1:]
            if not matrix[0]:
                matrix = matrix[1:]
            k -= 1
        return pop
'''
```

```
'''
第四版思路:
基于第二版改进:
第二版每次都只取一个，如果取多个呢?
比如第一次做归并排序时，
我是 [[1]+[2]]+[3]]+[4]]+[5]]+[6]]
而实际上应该是
[[1]+[2]] + [[3]+[4]] + [[5]+[6]]
[[1, 2, 3, 4]] + [[5, 6]]。
如果每次排序后去除t个，需要保证 n[0][-1] < n[1][-1]。
比如
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8。
第二版中每次排序会去除一个最小的。
matrix = [
   [5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 7
matrix = [
   [9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 6
matrix = [
   [10, 11, 13],
   [12, 13, 15]
],
k = 5
---
如果加入一条
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8
# 进入此条的条件是 k > len(matrix[0])
if matrix[0][-1] < matrix[1][0]:
    index1 = len(matrix[0])
    # index2 = 0
    # for t in xrange(matrix[1]):
        # if matrix[1][t] > matrix[0][-1]:
            # index2 = t
            # break
那么此时matrix[0] 将全部去除。k也相应的减去。
还有一种情况是 0[-1] 并不小于 1[0]
此时需要寻找 0 中小于 1[0]的数。
    for t in xrange(len(matrix[0])-1, -1, -1):
        if matrix[0][t] < matrix[1][0]:
            index1 = t
那么此时将去除maxtrix[0][:t+1], k也减去相应的len.
```

```
< len 则退化为第二版。
仍然TLE....
'''
'''
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        while k:
            matrix.sort(key=lambda x: x[0])
            if k >= len(matrix[0]):
                if len(matrix) >= 2:
                    if matrix[0][-1] <= matrix[1][0]:
                        index1 = len(matrix[0])
                    else:
                        for t in range(len(matrix[0])-1, -1, -1):
                            if matrix[0][t] <= matrix[1][0]:
                                index1 = t+1
                                break
                    popedMatrix = matrix[0][:index1]
                    matrix[0] = matrix[0][index1:]
                    if not matrix[0]:
                        matrix = matrix[1:]
                    pop = popedMatrix[-1]
                    k -= len(popedMatrix)
                    # print(matrix, index1)
                else:
                    return matrix[0][-1]
            else:
                pop = matrix[0][0]
                matrix[0] = matrix[0][1:]
                if not matrix[0]:
                    matrix = matrix[1:]
                k -= 1
        return pop
'''
'''
目前为止，通过的是第一版和第二版，第三和第四基于2的改进都以失败告终。
第四版的性能瓶颈在于，如果1[0]一直小于0[-1] 那么就需要一直迭代，如果恰好每次0中只有[0]比1[0]小，那么就会迭代过多的次数。
从而导致 TLE.
'''
```

Heap/MergeKSortedLists.py

```
"""
Merge k sorted linked lists and return it as one sorted list. Analyze and
describe its complexity.
Example:
Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6
合并 k 个有序链表。
用了最小堆：
把所有的链表节点入堆，然后出堆形成新的链表即可。
依然依靠了内置模块，待自己书写堆。
测试地址：
https://leetcode.com/problems/merge-k-sorted-lists/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
import heapq
class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        a = []
        heapq.heapify(a)
        for i in lists:
            while i:
                heapq.heappush(a, i.val)
                i = i.next
        if not a:
            return None
        root = ListNode(heapq.heappop(a))
        head = root
        while a:
            root.next = ListNode(heapq.heappop(a))
            root = root.next
        return head
```

Number/DivideTwoIntegers.py

```
"""
Given two integers dividend and divisor, divide two integers without using
multiplication, division and mod operator.
Return the quotient after dividing dividend by divisor.
The integer division should truncate toward zero.
Example 1:
Input: dividend = 10, divisor = 3
Output: 3
Example 2:
Input: dividend = 7, divisor = -3
Output: -2
Note:
Both dividend and divisor will be 32-bit signed integers.
The divisor will never be 0.
Assume we are dealing with an environment which could only store integers within
the 32-bit signed integer range: [−231,  231 − 1]. For the purpose of this
problem, assume that your function returns 231 − 1 when the division result
overflows.
除以某数，其结果要尽可能趋向0的，且为整数。
直接用Python 中的地板除 // 即可。
判断两个数的符号是否相同和处理 2**31-1 和 -2**31 即可。
测试链接：
https://leetcode.com/problems/divide-two-integers/description/
beat 100% 28ms.
这个题没什么意思...应该作为第二天的零启动任务来做，但已经做了，换一个零启动任务吧。
"""
class Solution(object):
    def divide(self, dividend, divisor):
        """
        :type dividend: int
        :type divisor: int
        :rtype: int
        """
        x = abs(dividend) // abs(divisor)
        if (dividend < 0 and divisor > 0) or (dividend > 0 and divisor < 0):
            if -x < -2**31:
                return -2**31
            return -x
        if x > 2**31-1:
            return 2**31-1
        return x
```

Number/Sqrt(x).py

```
"""
Implement int sqrt(int x).
Compute and return the square root of x, where x is guaranteed to be a non-
negative integer.
Since the return type is an integer, the decimal digits are truncated and only
the integer part of the result is returned.
Example 1:
Input: 4
Output: 2
Example 2:
Input: 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since
             the decimal part is truncated, 2 is returned.
顺手一写，直接用库...
测试地址：
https://leetcode.com/problems/sqrtx/description/
"""
import math
class Solution(object):
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        return int(math.sqrt(x))
```

Sorted/SortList.py

```
"""
Sort a linked list in O(n log n) time using constant space complexity.
Example 1:
Input: 4->2->1->3
Output: 1->2->3->4
Example 2:
Input: -1->5->3->4->0
Output: -1->0->3->4->5
O (n log n) 用归并排序比较好。
难点在于常量空间复杂度...
暂不解决这个问题，再回顾下归并排序。
归并排序的核心思路是分治，将大问题分成小问题，再将已经解决的小问题不断合并成一个解决方案。
所以归并排序的话先分割，进行对半分割即可。
_left = list[:middle]
_right = list[middle:]
分割成 _left 和 _right 后，要做的是归并。
merge_sort(_left, _right)
这里就归并用到底不做剪枝优化了。
merge_sort
的关键词是：
1. _left 和 _right 分别挑剩余的最小的合并到一个集合里。
2. 若 _left 耗尽，直接合并 _right。
3. _right 同理。
def merge_sort(_left, _right):
    _l_index = 0
    _r_index = 0
    _l_length = len(l)
    _r_length = len(r)
    while _l_i < _l_l and _r_i < _r_l:
        if l[_l_i] < r[_r_l]:
            l[_l_i]
            _l_i += 1
        else:
            r
    if _l_i == _l_l:
        extend _r
    if _r_i == _r_l:
        extend _l
    return result
这是最原始的 split ，后面要扩展一下才能用。
def split(list):
    _left = list[:middle]
    _right = list[middle:]
    return merge_sort(_left, _right)
在初次分割之后，_left 和 _right 确实分成了两份，但都是未经过排序的，直接用merge_sort的话是不行的（这里展开思考的话，会想到另一种排序——堆排序）。
我们需要对_left和_right分别再次进行 分割-合并。
def split(list):
    _left = split(list[:middle])
    _right = split(list[middle:])
    return merge_sort(_left, _right)
写到这里还有点问题，这样会无限分割下去，在加一个判断用于结束递归。
def split(list):
    if len(list) <= 1:
```

```
        return list
    _left = split(list[:middle])
    _right = split(list[middle:])
    return merge_sort(_left, _right)
元素<=1个时就可以返回了。
   [1, 2, 4, 5]
   left  ↓  right
  [1, 2]   [4, 5]
left↓right left↓right
 [1] [2]   [4] [5]
到这里
不在分割 [1] [2] 分别返回，然后执行了
merge_sort([1], [2])
merge_sort([4], [5])
然后再返回...
beat:
84%
测试地址：
https://leetcode.com/problems/sort-list/description/
"""
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution(object):
    def sortList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        lists = []
        while head:
            lists.append(head.val)
            head = head.next
        def merge_sort(l, r):
            _l = 0
            _r = 0
            _l_length = len(l)
            _r_length = len(r)
            result = []
            while _l < _l_length and _r < _r_length:
                if l[_l] < r[_r]:
                    result.append(l[_l])
                    _l += 1
                else:
                    result.append(r[_r])
                    _r += 1
            if _l == _l_length:
                while _r < _r_length:
                    result.append(r[_r])
                    _r += 1
            else:
                while _l < _l_length:
                    result.append(l[_l])
                    _l += 1
            return result
        def split(l):
```

```
            if len(l) <= 1:
                return l
            _l = split(l[:len(l)//2])
            _r = split(l[len(l)//2:])
            return merge_sort(_l, _r)
        lists = split(lists)
        if not lists:
            return None
        head = ListNode(lists[0])
        _head = head
        for i in lists[1:]:
            head.next = ListNode(i)
            head = head.next
        return _head
```

Sorted/sotred.py

```
# 各种排序。
import random
test_list = (
    list(range(10)),
    list(range(100)),
    list(range(1000)),
    list(range(10001)),
    list(range(50000))
)
list(map(random.shuffle, test_list))
result = (
    list(range(10)),
    list(range(100)),
    list(range(1000)),
    list(range(10001)),
    list(range(50000))
)
# 选择排序:
# 每次选取一个最小的值。
def selectionSort(shuffledList):
    new = []
    for i in range(len(shuffledList)):
        currentMinValue = min(shuffledList)
        new.append(currentMinValue)
        shuffledList.remove(currentMinValue)
    return new
# 插入排序。
# 例:
# 插入排序一次只换一格。
# 4，  2，  3，  5，  9，  6，  1
# 2，  4，  3，  5，  9，  6，  1
# def  insertingSort(shuffledList):
#     new = shuffledList.copy()
#     length = len(new)
#     for i in range(1, length):
#         if new[i] < new[i-1]:
#             new[i], new[i-1] = new[i-1], new[i]
#     return new
# def insertingSort(lst):
#         for i  in range(1,len(lst)):
#                 j = 0
#                 while lst[i] > lst[j]:
#                         j += 1
#                 results = lst[i]
#                 lst.pop(i)
#                 lst.insert(j,results)
#         return lst
# 归并排序
# 归并排序是将一个大问题分解成多个小问题来解决。
# 归并排序包含两个步骤，其一是分解，其二是合并。
# 在排序中分解的过程就是将一整个数组分解为多个小数组。
# 合并则是两两合并，每次都分别从一个数组中提取一个进行比较，将较小的放入新的数组中。
def combined(list1, list2):
        new = []
```

```
        indexOne = 0
        indexTwo = 0
        lengthOne, lengthTwo = len(list1), len(list2)
        while indexOne < lengthOne and indexTwo < lengthTwo:
            valueOne = list1[indexOne]
            valueTwo = list2[indexTwo]
            if valueOne > valueTwo:
                new.append(valueTwo)
                indexTwo += 1
            else:
                new.append(valueOne)
                indexOne +=1
        if indexOne == lengthOne:
            new.extend(list2[indexTwo:])
            return new
        if indexTwo == lengthTwo:
            new.extend(list1[indexOne:])
            return new
def makeValueInList(value):
    return [value]
def reduce(splitedList):
    length = len(splitedList)
    if length == 1:
        return splitedList[0]
    middle = length // 2
    left = reduce(splitedList[:middle])
    right = reduce(splitedList[middle:])
    return combined(left, right)
def mergeSort(shuffledList):
    splitedList = list(map(makeValueInList, shuffledList))
    result = reduce(splitedList)
    return result
# 快速排序
# 快速排序的思路与归并排序一样都是基于分治，将一个大问题变成小问题再组合起来。
# 快排的平均时间消耗是O(logn)，当然最差也有n²。
# 快排的思路是，选取一个元素，将大于它的放在左边，小于的放在右边。然后将左边右边再次进行相同的操作。
def fastSort(shuffledList):
    if len(shuffledList) <= 1:
        return shuffledList
    right = [i for i in shuffledList[1:] if i < shuffledList[0]]
    left = [i for i in shuffledList[1:] if i >= shuffledList[0]]
    return fastSort(right) + [shuffledList[0]] + fastSort(left)
for i, j in zip(test_list, result):
    assert fastSort(i) == j
    # assert mergeSort(i) == j
    # break
    # assert insertingSort(i) == j
    # assert selectionSort(i) == j
    # assert sorted(i) == j
```

Sorted/WiggleSortII.py

```
"""
Given an unsorted array nums, reorder it such that nums[0] < nums[1] > nums[2] <
nums[3]....
Example 1:
Input: nums = [1, 5, 1, 1, 6, 4]
Output: One possible answer is [1, 4, 1, 5, 1, 6].
Example 2:
Input: nums = [1, 3, 2, 2, 3, 1]
Output: One possible answer is [2, 3, 1, 3, 1, 2].
Note:
You may assume all input has valid answer.
Follow Up:
Can you do it in O(n) time and/or in-place with O(1) extra space?
1 锁了，暂不能做。
II 的话规则：
    单数位 大于 双数位。
思路：
一开始的话想到的是排序后，取中间，然后后半部分按从大到小铺在单数位上。
                                                前半部分按从大到小从末尾的双数位开始向前铺。
这个思路一开始是可以的，不过忽略了相等的部分，因为有可能铺的时候相遇。所以注意下，相等的等上面的
过程铺完之后，用于补剩下的部分。
进阶要求是：
 O(n) 时间，O(1) 空间。
排序的话 O(n log n)，不符合要求。
这个思路的关键点是找到中位数，但怎么在 O(n) 时间 O(1) 空间找到中位数呢？
下面是我的思考，没有最终实现。
1. 先找到里面最小的元素并获取出总长度。
2. 过一遍，找出 总长度 // 2 个比它大的元素，顺便记录出这里面的最大值。
3. 再过一遍，找出 比 2. 中最大值大的 元素。
4. 再找出比它大的元素个数 - 总长度 // 2
5. 若刚好有 总长度 // 2 个那么它就是中位数，否则它就是第 k（比它大的元素个数 - 总长度 // 2）
大个元素。
6. 缩小了范围。重复步骤...
时间复杂度非常依赖运气...
下面直接用了排序，没有达成进阶条件，待改进。
beat
68%
测试地址：
https://leetcode.com/problems/wiggle-sort-ii/description/
"""
class Solution(object):
    def wiggleSort(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place instead.
        """
        nums.sort()
        middle = nums[len(nums) // 2]
        larger = [i for i in nums[len(nums)//2:] if i != middle]
        smaller = [i for i in nums[:len(nums)//2] if i != middle]
        equal = [i for i in nums if i == middle]
        larger.reverse()
        smaller.reverse()
        length = len(nums)
```

```
        odd = 1
        even = length - 1 if (length-1) % 2 == 0 else length - 2
        for i in larger:
            # try:
            nums[odd] = i
            odd += 2
        for i in smaller:
            nums[even] = i
            even -= 2
        while even >= 0:
            nums[even] = equal.pop()
            even -= 2
        while odd < length:
            nums[odd] = equal.pop()
            odd += 2
```

Stack/GetMinStack.py

```
"""
设计一个可以 Get Min的栈结构。
栈是一个先进后出的数据结构。
要设计这样的一个需要两个栈来完成，一个栈用于正常的存储，另一个用于只存储最小的。
这样push/pop get min都是O(1) 时间完成。
非常基础。
其实在Python中可以用类直接写min属性，不过限定了只能用栈了。
"""
# deque 是一个Python实现的双端队列。
# 由底层C代码实现。
# 使用它可以保证在左右两端进行添加和删除操作时间都是O(1)。
from collections import deque
testStack = range(10)
class Stack:
    """
        一个先进后出的栈。
    """
    def __init__(self):
        """
            这里用实例对象。
        """
        self.stored = deque()
    def __repr__(self):
        # 3.6 +
        return f'<{self.stored}>'
    def push(self, value):
        self.stored.append(value)
    def pop(self):
        return self.stored.pop()
    def get_top(self):
        """
            查看栈顶的数据但不压出。
        """
        return self.stored[-1]
    def empty(self):
        if self.stored:
            return False
        return True
class MinStack(Stack):
    """
        一个可以在O(1)时间内获取出最小值的栈。
    """
    def __init__(self):
        super().__init__()
        self.minStack = Stack()
    def push(self, value):
        super().push(value)
        if self.minStack.empty():
            self.minStack.push(value)
        else:
            if self.minStack.get_top() <= value:
                self.minStack.push(self.minStack.get_top())
            else:
                self.minStack.push(value)
```

```
    def pop(self):
        super().pop()
        self.minStack.pop()
    def get_min(self):
        """
            返回栈顶但不压出。
        """
        return self.minStack.get_top()
a = Stack()
for i in testStack:
    a.push(i)
print(a)
a.pop()
print(a)
a = MinStack()
a.push(1)
print(a.get_min()) # 1
a.push(2)
print(a.get_min()) # 1
a.push(0)
print(a.get_min()) # 0
a.pop()
print(a.get_min()) # 1
```

Stack/ImplementQueueUsingStack.py

```
"""
使用栈来实现队列。
队列是一个先进先出的数据结构，栈则是一个先进后出的数据结构。
当然在 Python 中可以直接使用 Queue 队列，deque 双端队列。
不过还是要学习思考下如何用栈来实现。
用栈实现有两个难点：
栈只能是先进后出。所以要想实现的话务必需要另一个栈。
将当前栈里所有的数据压入另一个栈然后另一个栈里的先进后出就会变成原本的先进先出了。
难点1： 何时将数据压入栈2？
难点2：压入栈2后如果又有新的数据追加进来呢？
第一个问题我想到的是在栈2为空，且栈1数据为两个及以上时。
第二个问题我想到的是如果栈2有数据，则在栈1追加。
这时pop()，直到条件为难点1时清空栈1，压入栈2。
1. push 时，如果两个都没有数据则直接压入栈2.
2. push 时，如果栈2有数据则压入栈1.
3. pop 时，如果栈2有数据则 pop 栈2.
4. pop 栈2之后若栈2为空，则pop栈1，直至空。
测试用例：
https://leetcode.com/problems/implement-queue-using-stacks/
"""
# 要使用3遍以上时再抽象到一个单独的类中。
from GetMinStack import Stack
class Queue:
    def __init__(self):
        self.stack_one = Stack()
        self.stack_two = Stack()
    def push(self, value):
        if self.stack_two.empty():
            self.stack_two.push(value)
        else:
            self.stack_one.push(value)
    def pop(self):
        pop_result = self.stack_two.pop()
        if self.stack_two.empty():
            while 1:
                if self.stack_one.empty():
                    break
                self.stack_two.push(self.stack_one.pop())
        return pop_result
    def peek(self):
        return self.stack_two.get_top()
queue = Queue()
queue.push(1)
queue.push(2)
queue.push(3)
print(queue.pop())
queue.push(4)
print(queue.pop())
print(queue.pop())
print(queue.pop())
```

Stack/ReverseAStackByRecursive.py

```
"""
只能用递归不能用其他数据结构来逆序一个栈。
可以用另一个新栈的话比较容易。
"""
from GetMinStack import Stack
def reverseStack(stacks):
    new_stack = Stack()
    def helper(stacks):
        if stacks.empty():
            return
        new_stack.push(stacks.pop())
        helper(stacks)
    helper(stacks)
    return new_stack
# 测试用例。
a = range(10)
b = Stack()
for i in a:
    b.push(i)
# 不用assert了。
print(b)
print(reverseStack(b))
```

Stack/TrappingRainWater.py

```
"""
Given n non-negative integers representing an elevation map where the width of
each bar is 1, compute how much water it is able to trap after raining.
The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In
this case, 6 units of rain water (blue section) are being trapped. Thanks Marcos
for contributing this image!
Example:
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
图看链接：
https://leetcode.com/problems/trapping-rain-water/description/
思路：
1. 这里想到用单调栈来做，给的例子基本上是单调递减，然后有一个不符合的就开始处理。
不过随之而来的遇到的第一个不合理的地方：
            |
|           |
|  |     |  |
-----------
2 1  0 1  3
在 2 1 0 之后，出现了第一个不是单调递增的值，1，然后开始处理，处理之后得到值 1（1-0）。可是用后面的 3 能装更多的水。
2. 对上面问题的修复是片面的：
    我的想法是再开始判断单调递增，直到不合理的出现。这样修复后可以处理这一类问题，不过随之而来的又一个新的问题：
|                 |
|            |    |
|  |      |  | | |
|  |  |   |  | | |
------------------
4  2  1  2  3 2 4
显然用两边的 4 可以装更多的水，但用之前的单调递增+单调递减是不能达到这样的效果的。
3. 基于 2 的再改进。
   两次思考问题都过于局限，只看到了眼前的一点。
   这次的改进依然基于单调栈，遇到非单调递减的值，不会直接清空已经存储的点，而是全部覆盖记录。
   比如
   4 2 1 2 当遇到第二个 2 时，会进行 2-1 2-2 的记录，留下 4，因为 4此时是 > 2 的，若是 2 1 4 就没必要保留了。
   所以放置一个用于覆盖记录的列表。
   result = [0] * length
   每次非单调递减后就执行一次 stack[0] - i 与最小值的差覆盖，这样覆盖到最后结果就会是正确的。
例：
 4  2  1  2  3  2  4
[0, 0, 0, 0, 0, 0, 0]
 0  1  2  3  4  5  6
第一次单调递减打破后：
会执行一次 0 - 3 的判断。
[0, 0, 1, 0, 0, 0, 0]
此时栈中剩余的是记录 4 的下标 0 和 记录第二个2的下标 3。
第二次单调递减打破后：
会执行一次 0 - 4 的判断
[0, 0, 1, 0, 0, 0, 0] -> [0, 1, 2, 1, 0, 0, 0]
此时栈中剩余的是 0 和 4
第三次打破后：
会执行一次 0 - 6 的判断
```

```
[0, 1, 2, 1, 0, 0, 0] -> [0, 2, 3, 2, 1, 2, 0]
这样的时间复杂度 最差是 O(n²) 最好是 O(n)。
beat
0%
太慢，待改进。
测试地址：
https://leetcode.com/problems/trapping-rain-water/description/
"""
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        stack = [0]
        i = 1
        length = len(height)
        result = [0] * length
        while i < length:
            if height[i] <= height[stack[-1]]:
                stack.append(i)
                i += 1
            else:
                mins = min(height[stack[0]], height[i])
                index = 0
                for j in xrange(stack[0], i):
                    _ = mins - height[j]
                    if _ > 0 and _ > result[j]:
                        result[j] = _
                if height[stack[0]] <= height[i]:
                    stack = []
                else:
                    stack = [stack[0]]
                stack.append(i)
                i += 1
        return sum(result)
```

String/DetectCapital.py

```
"""
Given a word, you need to judge whether the usage of capitals in it is right or
not.
We define the usage of capitals in a word to be right when one of the following
cases holds:
1. All letters in this word are capitals, like "USA".
2. All letters in this word are not capitals, like "leetcode".
3. Only the first letter in this word is capital if it has more than one letter,
like "Google".
Otherwise, we define that this word doesn't use capitals in a right way.
Example 1:
Input: "USA"
Output: True
Example 2:
Input: "FlaG"
Output: False
题目中将三种情况定义为 capitals：
1. 全是大写。（"USA"）
2. 全是小写。("leetcode")
3. 多于1个字符只在第一位用了大写。("Google")
思路O(n)的遍历和内置。
遍历的判断：
ASCII码中：
65-90 是 A-Z.
97-122 是 a-z.
若第一个字符是大写，则判断：
剩下的是否全是大写，剩下的是否全是小写。
若第一个字符是小写，则判断：
剩下的是否全是小写。
只有一个字符时，无论如何都是capital.
测试用例：
https://leetcode.com/problems/detect-capital/description/
内置最快，自写慢个4ms，考虑到内置是c写的。4ms完全可以接受。
"""
class Solution(object):
    def detectCapitalUse(self, word):
        """
        :type word: str
        :rtype: bool
        """
        if len(word) < 2:
            return True
        letter_one = ord(word[0])
        # letter_two = word[1]
        if 65 <= letter_one <= 90:
            return self.letter_all_capital_or_lower(word[1:])
        return self.letter_all_lower(word[1:])
        # for i in word:
    def letter_all_capital_or_lower(self, word):
        #
        lower = False
        capital = False
        # A-Z
        left = 65
```

```
        right = 90
        if 97 <= ord(word[0]) <= 122:
            left = 97
            right = 122
        for i in word:
            if not left <= ord(i) <= right:
                return False
        return True
    def letter_all_lower(self, word):
        for i in word:
            if not 97 <= ord(i) <= 122:
                return False
        return True
```

String/FindAndReplacePattern.py

```
"""
You have a list of words and a pattern, and you want to know which words in words
matches the pattern.
A word matches the pattern if there exists a permutation of letters p so that
after replacing every letter x in the pattern with p(x), we get the desired
word.
(Recall that a permutation of letters is a bijection from letters to letters:
every letter maps to another letter, and no two letters map to the same letter.)
Return a list of the words in words that match the given pattern.
You may return the answer in any order.
Example 1:
Input: words = ["abc","deq","mee","aqq","dkd","ccc"], pattern = "abb"
Output: ["mee","aqq"]
Explanation: "mee" matches the pattern because there is a permutation {a -> m, b
-> e, ...}.
"ccc" does not match the pattern because {a -> c, b -> c, ...} is not a
permutation,
since a and b map to the same letter.
Note:
1 <= words.length <= 50
1 <= pattern.length = words[i].length <= 20
判断是否与给定的字符串模式是一样的。
思路是把所有的字符串转换为同一种编码的格式之后与pattern对比即可。
用的 str 直接就过了，beat 95%。 换成 set 取交集应该更快一些。
测试地址：
https://leetcode.com/contest/weekly-contest-98/problems/find-and-replace-
pattern/
beat:
95% 24ms.
"""
class Solution(object):
    def findAndReplacePattern(self, words, pattern):
        """
        :type words: List[str]
        :type pattern: str
        :rtype: List[str]
        """
        def translate_pattern(word):
            result = ['0']
            current = '0'
            patterns = {word[0]: '0'}
            for i in word[1:]:
                if patterns.get(i) is not None:
                    result.append(patterns.get(i))
                else:
                    current = str(int(current)+1)
                    patterns[i] = current
                    result.append(current)
            return ''.join(result)
        x = translate_pattern(pattern)
        result = []
        for i in words:
            if x == translate_pattern(i):
                result.append(i)
```

```
        return result
```

String/FirstUniqueCharacterInAString.py

```
"""
Given a string, find the first non-repeating character in it and return it's
index. If it doesn't exist, return -1.
Examples:
s = "leetcode"
return 0.
s = "loveleetcode",
return 2.
Note: You may assume the string contain only lowercase letters.
给定一个字符串，找到第一个不重复的字符，输出索引，如不存在输出 -1。
思路是直接使用字典，都是O(1)。
Discuss 中使用的 count 方法居然比这种方式快...
beat 72%
测试地址：
https://leetcode.com/problems/first-unique-character-in-a-string/description/
"""
class Solution(object):
    def firstUniqChar(self, s):
        """
        :type s: str
        :rtype: int
        """
        x = {}
        for i in s:
            try:
                x[i] += 1
            except:
                x[i] = 1
        for i in x.keys():
            if x[i] > 1:
                x.pop(i)
        for i in range(len(s)):
            if s[i] in x:
                return i
        return -1
```

String/HammingDistance.py

```
"""
汉明距离：
Input: x = 1, y = 4
Output: 2
Explanation:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑
The above arrows point to positions where the corresponding bits are different.
相异的部分就是汉明距离。
应用：
搜索引擎中的搜图：
https://blog.csdn.net/liudongdong19/article/details/80541216
基本思路是：
    将图片转换灰度后会有64级，每级对应一个整数，两两对比整数。也就是取汉明距离。
测试用例：
https://leetcode.com/problems/hamming-distance/description/
思路：
利用异或运算相同取0，相异取1，最后计算出
"""
class Solution(object):
    def hammingDistance(self, x, y):
        """
        :type x: int
        :type y: int
        :rtype: int
        """
        return str(bin(x ^ y)).count('1')
```

String/ImplementStrStr().py

```
"""
Implement strStr().
Return the index of the first occurrence of needle in haystack, or -1 if needle
is not part of haystack.
Example 1:
Input: haystack = "hello", needle = "ll"
Output: 2
Example 2:
Input: haystack = "aaaaa", needle = "bba"
Output: -1
Clarification:
What should we return when needle is an empty string? This is a great question to
ask during an interview.
For the purpose of this problem, we will return 0 when needle is an empty string.
This is consistent to C's strstr() and Java's indexOf().
strStr() 等同于 Python 中的 find()
想要通过直接：
```

str.find()

```
直接 beat 100%...
测试用例：
https://leetcode.com/problems/implement-strstr/description/
自己的实现思路：
记录两者len，needle的长度大于haystack，则一定不存在 -1.
接下来遍历 haystack 若此时剩余长度小于 needle 则一定不存在 -1.
若[0]与[0]相同，则进一步进行对比，一样则返回下标，不一样则继续。
built-in 20ms
myself 24ms
考虑到built-in 是 c++，这样效率也基本一致了。
"""
class Solution(object):
    def strStr(self, haystack, needle):
        """
        :type haystack: str
        :type needle: str
        :rtype: int
        """
        # bulit-in 20ms
        # return haystack.find(needle)
        # myself
        if not needle:
            return 0
        # 24ms.
        lengthHaystack = len(haystack)
        lengthNeedle = len(needle)
        if lengthNeedle > lengthHaystack:
            return -1
        if lengthNeedle == lengthHaystack:
            return 0 if haystack == needle else -1
        for i, d in enumerate(haystack):
            if lengthHaystack - i < lengthNeedle:
                return -1
```

```
            if d == needle[0]:
                if haystack[i:i+lengthNeedle] == needle:
                    return i
```

String/IsomorphicStrings.py

```
"""
Given two strings s and t, determine if they are isomorphic.
Two strings are isomorphic if the characters in s can be replaced to get t.
All occurrences of a character must be replaced with another character while
preserving the order of characters. No two characters may map to the same
character but a character may map to itself.
Example 1:
Input: s = "egg", t = "add"
Output: true
Example 2:
Input: s = "foo", t = "bar"
Output: false
Example 3:
Input: s = "paper", t = "title"
Output: true
Note:
You may assume both s and t have the same length.
给定两个字符串，判断是否属于相同的模式。
这个没想到什么好办法，就是根据字典进行替换然后对比是否一致。
在Discuss里看到一个 one line 版。
先上自己的思路：
迭代字符串a，如果遇到的是a_d中的字符串，则根据a_d替换，否则将其根据出现的顺序添加到a_d中，
这样做让每个单词根据字符所出现的顺序进行字典替换。
测试用例：
https://leetcode.com/problems/isomorphic-strings/description/
"""
class Solution(object):
    def isIsomorphic(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        """
        这里可以缩写为一个函数，但不需要复用，直接复制粘贴了。
        """
        s_k = 0
        t_k = 0
        s_d = {}
        t_d = {}
        new_s = ''
        new_t = ''
        for i in s:
            if s_d.get(i):
                new_s += s_d.get(i)
            else:
                s_d[i] = str(s_k)
                s_k += 1
        for i in t:
            if t_d.get(i):
                new_t += t_d.get(i)
            else:
                t_d[i] = str(t_k)
                t_k += 1
```

```
        return new_s == new_t
"""
接下来是one line 版，虽然并不是很高效，但思路不错。
这个问题的核心问题是 重复的数据，如果每个单词都不想同那么也是同一种模式，关键就是找出相同的单词。
zip会将每个部位的单词两两对应。
`zip` `rar`
('z', 'r'), ('i', 'a'), ('p', 'r')
如果两个单词的模式相同，那么一定会出现多组同样的数据，且数量与原单词去重后一致。
t,s与s,t的效果应该是一致的，都不会影响判断。
return len(set(zip(s, t))) == len(set(s)) and len(set(zip(t, s))) == len(set(t))
"""
"""
第三版：
根据 one line 的思路：
zip可以有效检测分组：
one line 虽然简单易写，不过有3次set，时间复杂度为 O(3n)，set是c++写的所以很快，感觉不出来。
就是把第一版变成了zip的的形式，理论上应该会快的，但实际运行时间与第一版一致。
"""
class Solution(object):
    def isIsomorphic(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        s_k = 0
        t_k = 0
        s_d = {}
        t_d = {}
        for i in zip(s, t):
            if s_d.get(i[0]):
                s_k2 = s_d.get(i[0])
            else:
                s_k2 = s_d[i[0]] = str(s_k)
                s_k += 1
            if t_d.get(i[1]):
                t_k2 = t_d.get(i[1])
            else:
                t_k2 = t_d[i[1]] = str(t_k)
                # t_k2 = t_d.get(i[1])
                t_k += 1
            if s_k2 != t_k2:
                return False
        return True
```

String/JewelsAndStones.py

```
"""
Example 1:
Input: J = "aA", S = "aAAbbbb"
Output: 3
Example 2:
Input: J = "z", S = "ZZ"
Output: 0
Note:
S and J will consist of letters and have length at most 50.
The characters in J are distinct.
查找J中的每个字符在 S 出现的次数的总和。
改进：
J有可能有重复的数。
测试数据：
https://leetcode.com/problems/jewels-and-stones/description/
"""
class Solution(object):
    def numJewelsInStones(self, J, S):
        """
        :type J: str
        :type S: str
        :rtype: int
        """
        S_dict = {i:S.count(i) for i in set(S)}
        return sum((S_dict.get(i, 0) for i in J))
```

String/LengthOfLastWord.py

```
"""
Input: "Hello World"
Output: 5
每个单词都会被 ' ' 分割。
所以要做的是从尾向前找第一个空格。当然Python 下用 strip(' ')分割后取[-1]是最好写出来的。
当然还要确保下尾部开始不能是' '，要找到第一个单词才行。
测试用例：
https://leetcode.com/problems/length-of-last-word/description/
"""
class Solution(object):
    def lengthOfLastWord(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0
        result = 0
        flag = False
        for i in s[::-1]:
            if i == ' ' and not flag:
                continue
            elif i == ' ' and flag:
                return result
            else:
                result += 1
                flag = True
                continue
            return result
        return result
```

String/LetterCombinationsOfAPhoneNumber.py

```
"""
Given a string containing digits from 2-9 inclusive, return all possible letter
combinations that the number could represent.
A mapping of digit to letters (just like on the telephone buttons) is given
below. Note that 1 does not map to any letters.
Example:
Input: "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
Note:
Although the above answer is in lexicographical order, your answer could be in
any order you want.
图看链接：
https://leetcode.com/problems/letter-combinations-of-a-phone-number/description/
老式手机键盘一样，返回所有的组合。
使用的递归思路：
迭代每一个组合，返回所有的组合...
应该容易理解。时间复杂度上应该只有 O(xn)的解法。
测试链接：
https://leetcode.com/problems/letter-combinations-of-a-phone-number/description/
beat 100% 20ms。
应该会有浮动吧，一次通过，不测第二次了。
"""
class Solution(object):
    def letterCombinations(self, digits):
        """
        :type digits: str
        :rtype: List[str]
        """
        maps = {
            '2': 'abc',
            '3': 'def',
            '4': 'ghi',
            '5': 'jkl',
            '6': 'mno',
            '7': 'pqrs',
            '8': 'tuv',
            '9': 'wxyz'
        }
        result = []
        def reduce_abc(strs, currentStr=""):
            if not strs:
                return currentStr
            else:
                for i in maps[strs[0]]:
                    x = reduce_abc(strs[1:], currentStr=currentStr+i)
                    if x:
                        result.append(x)
        reduce_abc(digits)
        return result
```

String/LongesSubstrings.py

```
"""
Given a string, find the length of the longest substring without repeating
characters.
Examples:
Given "abcabcbb", the answer is "abc", which the length is 3.
Given "bbbbb", the answer is "b", with the length of 1.
Given "pwwkew", the answer is "wke", with the length of 3. Note that the answer
must be a substring, "pwke" is a subsequence and not a substring.
找到最长的子字符串。
基本思路是逐个遍历若不在new里则添加进里面，如果在new里则表示已经发生了重复，那么会对比目前的new
与之前保存的子字符串长度哪个较大，
保存较大的。新的new则为发生重复的字符串直到最后+新数据。`pkwk`遍历到第二个k时new此时会变成
`wk`。
因为是子字符串并不是子序列，所以这样做是可行的。
时间复杂度 O(n)
测试用例：
https://leetcode.com/problems/longest-substring-without-repeating-
characters/description/
"""
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s:
            return 0
        longestSubstringLength = 1
        new = s[0]
        for data in s[1:]:
            if data not in new:
                new += data
                continue
            # repeated.
            if len(new) > longestSubstringLength:
                longestSubstringLength = len(new)
            new = new[new.index(data)+1:] + data
        if len(new) > longestSubstringLength:
            return len(new)
        return longestSubstringLength
```

String/LongestCommonPrefix.py

```
"""
Write a function to find the longest common prefix string amongst an array of
strings.
If there is no common prefix, return an empty string "".
Example 1:
Input: ["flower","flow","flight"]
Output: "fl"
Example 2:
Input: ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
Note:
All given inputs are in lowercase letters a-z.
所有单词的共同的前缀。
没什么技巧，从每个单词的第一个开始对比就好了。
Python 中有一个 zip 可以做这些事情。
beat 90%
测试地址：
https://leetcode.com/problems/longest-common-prefix/description/
今日的零启动任务。
"""
class Solution(object):
    def longestCommonPrefix(self, strs):
        """
        :type strs: List[str]
        :rtype: str
        """
        result = ""
        for i in zip(*strs):
            a = i[0]
            for j in i[1:]:
                if j != a:
                    return result
            else:
                result += a
        return result
```

String/LongestPalindromicSubstrings.py

```
"""
Given a string s, find the longest palindromic substring in s. You may assume
that the maximum length of s is 1000.
Example 1:
Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
Example 2:
Input: "cbbd"
Output: "bb"
最长的回文子串。
1. 本来想用正则的，发现正则不重复搜。
sorted(re.findall(r'((?P<letter>.{1}).*(?P=letter))', "abacab"), key=lambda x:
len(x[0]))
unpassed, wrong answer.
2. 回文串的定义是，翻转过来也相同，也就是从中间分开，从各自的中间开始走，到各自的头相同就对了。
思路:
 "cbbd"
 遍历一遍，找到所有字符的位置，放到hash中。
 {"c": [0], "b": [1, 2], "d": [3]}
 若有一个大于len() 1的，就开始判断:
   1. 判断从大到小。
   1.1 制造从大到小的字典{2: [(1, 3), (2, 4)]} 可以用itertools.combinations()
 。
 若全不大于，返回s[0].
效率有点低...大概率 TLE.
WTF, passed，而且还 beat 33%!
"""
import itertools
class Solution(object):
    # beat 33%
    def longestPalindrome(self, s):
        if len(s) <= 1:
            return s
        # {"c": [0], "b": [1, 2], "d": [3]}
        s_dict = {}
        for i, d in enumerate(s):
            try:
                s_dict[d].append(i)
            except KeyError:
                s_dict[d] = [i]
        # print(s_dict)
        # {2: [(1, 3), (2, 4)]}
        value_dict = {}
        for i in s_dict:
            if len(s_dict[i]) >= 2:
                for j in self.makeCombinations(s_dict[i]):
                    try:
                        value_dict[j[1]-j[0]].append(j)
                    except KeyError:
                        value_dict[j[1]-j[0]] = [j]
        # print(value_dict)
        for i in sorted(value_dict, reverse=True):
            for j in value_dict[i]:
```

```
                x = s[j[0]:j[1]+1]
                if x == x[::-1]:
                    return x
        return s[0]
    def makeCombinations(self, split_list):
        return itertools.combinations(split_list, 2)
    # wrong
    # def longestPalindrome(self, s):
    #     """
    #     :type s: str
    #     :rtype: str
    #     """
    #     # if len(s) == 1:
    #     #     return s
    #     findall = re.findall(r'((?P<letter>.{1}).*(?P=letter))', s)
    #     print(findall)
    #     for i in sorted(findall, reverse=True, key=lambda x: len(x[0])):
    #         if i[0] == i[0][::-1]:
    #             return i[0]
    #     if not s:
    #         return ""
    #     return s[0]
s = Solution()
print(s.longestPalindrome("babad"*200))
```

String/LongPressedName.py

```
"""
Your friend is typing his name into a keyboard.  Sometimes, when typing a
character c, the key might get long pressed, and the character will be typed 1 or
more times.
You examine the typed characters of the keyboard.  Return True if it is possible
that it was your friends name, with some characters (possibly none) being long
pressed.
Example 1:
Input: name = "alex", typed = "aaleex"
Output: true
Explanation: 'a' and 'e' in 'alex' were long pressed.
Example 2:
Input: name = "saeed", typed = "ssaaedd"
Output: false
Explanation: 'e' must have been pressed twice, but it wasn't in the typed
output.
Example 3:
Input: name = "leelee", typed = "lleeelee"
Output: true
Example 4:
Input: name = "laiden", typed = "laiden"
Output: true
Explanation: It's not necessary to long press any character.
Note:
name.length <= 1000
typed.length <= 1000
The characters of name and typed are lowercase letters.
判断 typed 里 是否存在 name，保证顺序的。
思路：
typed 和 name 各一个指针，命中了就一起走，没命中就typed走。最后判断指针是否指向了name尾。
测试地址：
https://leetcode.com/contest/weekly-contest-107/problems/long-pressed-name/
"""
class Solution(object):
    def isLongPressedName(self, name, typed):
        """
        :type name: str
        :type typed: str
        :rtype: bool
        """
        x = 0
        y = 0
        x_length = len(name)
        y_length = len(typed)
        while x < x_length and y < y_length:
            if typed[y] == name[x]:
                y += 1
                x += 1
            else:
                y += 1
        if x == x_length:
            return True
        return False
```

String/MinimumAddToMakeParenthesesValid.py

```
"""
Given a string S of '(' and ')' parentheses, we add the minimum number of
parentheses ( '(' or ')', and in any positions ) so that the resulting
parentheses string is valid.
Formally, a parentheses string is valid if and only if:
It is the empty string, or
It can be written as AB (A concatenated with B), where A and B are valid strings,
or
It can be written as (A), where A is a valid string.
Given a parentheses string, return the minimum number of parentheses we must add
to make the resulting string valid.
Example 1:
Input: "())"
Output: 1
Example 2:
Input: "((("
Output: 3
Example 3:
Input: "()"
Output: 0
Example 4:
Input: "()))(("
Output: 4
Note:
S.length <= 1000
S only consists of '(' and ')' characters.
给一个字符串只包含 "(" 和 ")"。
返回至少补多少个可以达成全部有效。
思路:
去除原来有效的，剩余多少个即为需要多少个。
"""
class Solution(object):
    def minAddToMakeValid(self, S):
        """
        :type S: str
        :rtype: int
        """
        if not S:
            return 0
        t = [S[0]]
        for i in S[1:]:
            if i == ')':
                if not t:
                    t.append(i)
                    continue
                if t[-1] == '(':
                    t.pop()
                else:
                    t.append(i)
            else:
                t.append(i)
        return len(t)
```

String/MinimumWindowSubstring.py

```
"""
Given a string S and a string T, find the minimum window in S which will contain
all the characters in T in complexity O(n).
Example:
Input: S = "ADOBECODEBANC", T = "ABC"
Output: "BANC"
Note:
If there is no such window in S that covers all characters in T, return the empty
string "".
If there is such window, you are guaranteed that there will always be only one
unique minimum window in S.
在S中找到包含T所有字符的长度最小的子字符串。
要求在 O(n) 时间复杂度内完成。
思路:
1. emmm.本来是一个 O（n）的但是最后出了点问题...总体思路没错，待改进。passed的了，但效率很
低。
首先是 哈希 T .
哈希后的结果放到两个变量里。一个用于对比一个用于删除判断。
返回迭代S:
    如果这个元素存在于 T 中，记录，并在T中删除对应的字符。
        如果此时用于删除的 T 没有了。ok，找出记录的下标的最大与最小。记录为结果。
        用于删除的T没有了也不要停止，继续迭代，此后不断更新重复出现的字符的下标，重复对比此时记录
的长度。
为什么可行呢?
S = ADOBE A CODEBANC T = ABC
在这里加一个A进去。
当迭代到ADOBE 时，记录的A = 0 B = 3。此时如果记录的话为 0:3 包含了 A与B.
不管此后与谁重复，只要找里面的最小与最大的下标，即可找到所有想要的值。
这个思路看起来是 O(n) 的。只需迭代一遍，中途就记录几个min，max就可以。
可实际实施起来...
发现这个min与max还不是很好记录...
用堆吧只能很快速的获取一个。
用二叉搜索树吧删除还有点麻烦。
想用红黑吧...还不会写。
到是可以用set暂时代替红黑，喔对，可以用set试试。
啊哈，现在什么都没用，直接用了生成器生成。
以后再试试，先记下来。
下面这个pass了喔。
600+ms beat 7% = =.
测试地址:
https://leetcode.com/problems/minimum-window-substring/description/
"""
from collections import deque
class Solution(object):
    def minWindow(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: str
        """
        # b = set(t)
        b = {}
        for i in t:
            try:
```

```
                b[i] += 1
            except:
                b[i] = 1
        a = b.copy()
        # [(mins_index, maxs_index), (mins_index, maxs_index), (mins_index,
maxs_index)]
        x = {}
        mins = ""
        # t_min = 0
        # t_max = 0
        for i, d in enumerate(s):
            if d in b:
                try:
                    x[d].append(i)
                except:
                    x[d] = deque([i], maxlen=b[d])
                if a.get(d):
                    a[d] -= 1
                    if not a[d]:
                        a.pop(d)
                if not a:
                    values = x.values()
                    if not mins:
                        mins = s[min((q[0] for q in values)):max((q[-1] for q in
values))+1]
                    else:
                        mins = min(mins, s[min((q[0] for q in
values)):max((q[-1] for q in values))+1], key=len)
        if a:
            return ""
        return mins
```

String/NumberOfSegmentsInString.py

```
"""
Count the number of segments in a string, where a segment is defined to be a
contiguous sequence of non-space characters.
Please note that the string does not contain any non-printable characters.
Example:
Input: "Hello, my name is John"
Output: 5
深刻的体会到了什么是，当我要用正则解决一个问题时，那么就有了两个问题。
cry= =.
其实本来用不着正则的。
就是根据空白分割，然后统计数量，没什么难度。O(n).
Python中可以直接 len(s.split()).
测试用例：
https://leetcode.com/problems/number-of-segments-in-a-string/description/
"""
import re
class Solution(object):
    def countSegments(self, s):
        """
        :type s: str
        :rtype: int
        """
        s = s.strip()
        if not s.strip():
            return 0
        result = re.split(r'\s+', s)
        return len(result)
```

String/PermutationInString.py

```
"""
Given two strings s1 and s2, write a function to return true if s2 contains the
permutation of s1. In other words, one of the first string's permutations is the
substring of the second string.
Example 1:
Input:s1 = "ab" s2 = "eidbaooo"
Output:True
Explanation: s2 contains one permutation of s1 ("ba").
Example 2:
Input:s1= "ab" s2 = "eidboaoo"
Output: False
Note:
The input strings only contain lower case letters.
The length of both given strings is in range [1, 10,000].
类似于 Find All Anagrams in a String 难度应该颠倒过来。
这个的测试用例更丰富，发现了没想到的一个盲点。
思路请看
https://github.com/HuberTRoy/leetCode/blob/master/DP/FindAllAnagramsInAString.py
beat 79%
测试地址：
https://leetcode.com/problems/permutation-in-string/description/
"""
class Solution(object):
    def checkInclusion(self, s1, s2):
        """
        :type s1: str
        :type s2: str
        :rtype: bool
        """
        if len(s1) > len(s2):
            return False
        counts = {}
        for i in s1:
            try:
                counts[i] += 1
            except:
                counts[i] = 1
        pre = counts.copy()
        for c in range(len(s2)):
            i = s2[c]
            if i in pre:
                pre[i] -= 1
                if not pre[i]:
                    pre.pop(i)
                if not pre:
                    return True
            else:
                if i in counts:
                    if i != s2[c-len(s1)+sum(pre.values())]:
                        for t in s2[c-len(s1)+sum(pre.values()):c]:
                            if t == i:
                                break
                            try:
                                pre[t] += 1
```

```
                        except:
                            pre[t] = 1
                    continue
                pre = counts.copy()
                if i in pre:
                    pre[i] -= 1
                    if not pre[i]:
                        pre.pop(i)
        return False
```

String/ReplaceWords.py

```
"""
In English, we have a concept called root, which can be followed by some other
words to form another longer word - let's call this word successor. For example,
the root an, followed by other, which can form another word another.
Now, given a dictionary consisting of many roots and a sentence. You need to
replace all the successor in the sentence with the root forming it. If a
successor has many roots can form it, replace it with the root with the shortest
length.
You need to output the sentence after the replacement.
Example 1:
Input: dict = ["cat", "bat", "rat"]
sentence = "the cattle was rattled by the battery"
Output: "the cat was rat by the bat"
测试用例：
https://leetcode.com/problems/replace-words/description/
思路：
第一个思路是直接迭代，然后用in查看是否在某个单词中，是的话就直接替换。
这样做有违要求，successor是root+followed。不能是 xxx+root+followed.
当然即使不违反题目要求也基本是不行的，这样做的时间复杂度会变为 O(m*n)其中的n为字典中字符串的总长度。in这个操作符应该是将每个字符都一一对应一遍。
第二个思路是用正则。直接排序后全替换一遍，易写，但效率非常低。与上一个一致。不过提供了一种小思路。
第三种是根据前缀，学习到前缀树 `Trie`。
前缀树用来保存一系列字符串，增加这些字符串公共部分的复用率，达到快速检索的目的。
参考：
https://segmentfault.com/a/1190000008877595
不过，在这里题目直接将字典给了出来，也无需再构建前缀树。
使用前缀树的目的是进行高效的检索。
比如
"the cattle was rattled by the battery"
字典是
["cat", "bat", "rat", "rain"]
那么前缀树的构成是
    c    b     r
    a    a     a
    t    t   t   i
                 n
在这种思路下，我们在搜索the的时候也是要先匹配t是否在第一层中，然后再逐个向下匹配。
既然已经给了字典，直接利用字符串索引即可。
1. 将原句根据' '分离。
2. 迭代分离后的句子，然后迭代字典。
3. 将字典的第一个字符与迭代的句子的第一个字符相比较，如果相同则进入判断，判断成功则结束此次字典迭代进入下一个单词。
4. 判断的逻辑，将字典中的此单词与此单词传入，迭代字典中的此单词，逐个字符比较相同则不断比较，一直到字典中的此单词耗尽或此单词耗尽。
"""
class Solution(object):
    def replaceWords(self, dict, sentence):
        """
        :type dict: List[str]
        :type sentence: str
        :rtype: str
        """
        def judge(dict_word, word):
```

```
            for i in dict_word:
                if not word:
                    return False
                if i == word[0]:
                    word = word[1:]
                    continue
                return False
            return True
        dicts = sorted(dict, key=len)
        sentence = sentence.split(' ')
        for i, d in enumerate(sentence):
            for j in dicts:
                if d[0] == j[0]:
                    if judge(j[1:], d[1:]):
                        sentence[i] = j
                        break
        return ' '.join(sentence)
#         最开始的正则思路，用\s保证一定是root+followed的形式。
#         sentence = ' ' + sentence
#         for i in dicts:
#             sentence = re.sub(r'\s{}\w*'.format(i), ' '+i, sentence)
#         return sentence[1:]
```

String/ReverseInteger.py

```
"""
Given a 32-bit signed integer, reverse digits of an integer.
Example 1:
Input: 123
Output: 321
Example 2:
Input: -123
Output: -321
Example 3:
Input: 120
Output: 21
Note:
Assume we are dealing with an environment which could only store integers within
the 32-bit signed integer range: [−231,  231 − 1]. For the purpose of this
problem, assume that your function returns 0 when the reversed integer
overflows.
翻转一个整数。
我直接用了转成字符串，然后取反然后转成整数的方法。
效率过得去。
beat 45%.
测试地址：
https://leetcode.com/problems/reverse-integer/description/
"""
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        # str_x = str(x)
        if x < 0:
            x = -int(str(abs(x))[::-1])
        else:
            x = int(str(abs(x))[::-1])
        if x > 2**31 or x < -2**31:
            return 0
        return x
```

String/ReverseString.py

```
"""
Write a function that takes a string as input and returns the string reversed.
Example 1:
Input: "hello"
Output: "olleh"
Example 2:
Input: "A man, a plan, a canal: Panama"
Output: "amanaP :lanac a ,nalp a ,nam A"
倒序字符串，在 Python 中应该是非常简单的，Python 的字符串支持分片操作。
基本有这么几种方法：
1. 创建新字符串，从后向前迭代，然后与新字符串合并。
2. 直接用分片 [::-1]
3. 转换成列表，然后翻转列表。
经测试 2 和 3的效率差不多，没看源码，底层2应该用的1的思路实现的吧。
测试用例：
https://leetcode.com/problems/reverse-string/description/
"""
class Solution(object):
    def reverseString(self, s):
        """
        :type s: str
        :rtype: str
        """
        # return s[::-1]
        return ''.join(reversed(list(s)))
```

String/ShiftingLetters.py

```
"""
We have a string S of lowercase letters, and an integer array shifts.
Call the shift of a letter, the next letter in the alphabet, (wrapping around so
that 'z' becomes 'a').
For example, shift('a') = 'b', shift('t') = 'u', and shift('z') = 'a'.
Now for each shifts[i] = x, we want to shift the first i+1 letters of S, x
times.
Return the final string after all such shifts to S are applied.
Example 1:
Input: S = "abc", shifts = [3,5,9]
Output: "rpl"
Explanation:
We start with "abc".
After shifting the first 1 letters of S by 3, we have "dbc".
After shifting the first 2 letters of S by 5, we have "igc".
After shifting the first 3 letters of S by 9, we have "rpl", the answer.
Note:
1 <= S.length = shifts.length <= 20000
0 <= shifts[i] <= 10 ^ 9
给一个字符串，不断经过转换，得出最终字符串。
每一轮的转换都是对这之前的所有字符串而言的。
思路：
1.
 从后向前，得到索引 % 26.
2.
 转换字符到 ascii，相加然后处理超过的量。
测试地址：
https://leetcode.com/contest/weekly-contest-88/problems/shifting-letters/
"""
class Solution(object):
    def shiftingLetters(self, S, shifts):
        """
        :type S: str
        :type shifts: List[int]
        :rtype: str
        """
        letters = "abcdefghijklmnopqrstuvwxyz"
        new_s = []
        old_index = 0
        for i, j in zip(S[::-1], shifts[::-1]):
            index = letters.index(i)
            new_index = (index+j+old_index)%26
            new_s.append(letters[new_index])
            old_index += j
        return ''.join(new_s[::-1])
```

String/SimplifyPath.py

```
"""
Given an absolute path for a file (Unix-style), simplify it.
For example,
path = "/home/", => "/home"
path = "/a/./b/../../c/", => "/c"
path = "/a/../../b/../c//.//", => "/c"
path = "/a//b////c/d//././/..", => "/a/b/c"
In a UNIX-style file system, a period ('.') refers to the current directory, so
it can be ignored in a simplified path. Additionally, a double period ("..")
moves up a directory, so it cancels out whatever the last directory was. For more
information, look here:
https://en.wikipedia.org/wiki/Path_(computing)#Unix_style
Corner Cases:
Did you consider the case where path = "/../"?
In this case, you should return "/".
Another corner case is the path might contain multiple slashes '/' together, such
as "/home//foo/".
In this case, you should ignore redundant slashes and return "/home/foo".
给一个 UNIX 风格的文件系统字符串，修剪成标准格式。
思路：
1. 先用正则多斜线变一斜线。
2. 一个用于存放结果的新列表：
    遇到 . 不管，遇到 .. 就弹出尾部的一个，其他的均加入。
3. 最后用 '/' 合并起来。
beat 73%
测试了多次最高是 28ms，和 24ms大致一样，24ms的没用正则，直接 split('/') 然后 判断非空的组成新的。
测试地址：
https://leetcode.com/problems/simplify-path/description/
"""
import re
class Solution(object):
    def simplifyPath(self, path):
        """
        :type path: str
        :rtype: str
        """
        path = re.sub(r'/+', '/', path)
        if path[-1] == '/':
            path = path[:-1]
        path = path.split('/')
        new_path = []
        for i in path:
            if i == '..':
                try:
                    new_path.pop()
                    continue
                except:
                    continue
            if i =='.':
                continue
            new_path.append(i)
        x = '/'.join(new_path)
        if x and x[0] == '/':
```

```
            return x
        return '/' + x
```

String/SortCharactersByFrequency.py

```
"""
Given a string, sort it in decreasing order based on the frequency of
characters.
Example 1:
Input:
"tree"
Output:
"eert"
Explanation:
'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid
answer.
Example 2:
Input:
"cccaaa"
Output:
"cccaaa"
Explanation:
Both 'c' and 'a' appear three times, so "aaaccc" is also a valid answer.
Note that "cacaca" is incorrect, as the same characters must be together.
Example 3:
Input:
"Aabb"
Output:
"bbAa"
Explanation:
"bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
给定一个字符串，以字符出现的频率进行排序。
思路:
1. 用一个字典记录每个字符出现的频率。
2. 根据出现的频率排序。
3. 因为直接堆在一起即可，直接构建一个列表。
4. 在组合起来。
beat 95% 36ms.
测试地址:
https://leetcode.com/problems/sort-characters-by-frequency/description/
"""
class Solution(object):
    def frequencySort(self, s):
        """
        :type s: str
        :rtype: str
        """
        x = {}
        for i in s:
            try:
                x[i] += 1
            except:
                x[i] = 1
        b = sorted(x, key=lambda t: x[t], reverse=True)
        return ''.join([i*x[i] for i in b])
```

String/StringToInteger.py

```
"""
Implement atoi which converts a string to an integer.
The function first discards as many whitespace characters as necessary until the
first non-whitespace character is found. Then, starting from this character,
takes an optional initial plus or minus sign followed by as many numerical digits
as possible, and interprets them as a numerical value.
The string can contain additional characters after those that form the integral
number, which are ignored and have no effect on the behavior of this function.
If the first sequence of non-whitespace characters in str is not a valid integral
number, or if no such sequence exists because either str is empty or it contains
only whitespace characters, no conversion is performed.
If no valid conversion could be performed, a zero value is returned.
Note:
Only the space character ' ' is considered as whitespace character.
Assume we are dealing with an environment which could only store integers within
the 32-bit signed integer range: [−231,  231 − 1]. If the numerical value is out
of the range of representable values, INT_MAX (231 − 1) or INT_MIN (−231) is
returned.
Example 1:
Input: "42"
Output: 42
Example 2:
Input: "   -42"
Output: -42
Explanation: The first non-whitespace character is '-', which is the minus sign.
             Then take as many numerical digits as possible, which gets 42.
Example 3:
Input: "4193 with words"
Output: 4193
Explanation: Conversion stops at digit '3' as the next character is not a
numerical digit.
Example 4:
Input: "words and 987"
Output: 0
Explanation: The first non-whitespace character is 'w', which is not a numerical

             digit or a +/- sign. Therefore no valid conversion could be
performed.
Example 5:
Input: "-91283472332"
Output: -2147483648
Explanation: The number "-91283472332" is out of the range of a 32-bit signed
integer.
             Thefore INT_MIN (−231) is returned.
实现一个从字符串到整数的函数。
给出的例子有这么多。
主要思路：
1. 先是去掉两边的空白。
2. 确定出首位符号（有则按有的，无则按+。）
3. 之后判断剩下个每一位是否是一个十进制数。
4. 最后判断 -2**31 < x < 2**31-1，输出。
还发现了一个Python 2 与 3 的不同之处：
在处理整数的时候：
2 中 int("-     321") 可以处理为 -321.
```

3 中则会报错。
测试地址：
https://leetcode.com/problems/string-to-integer-atoi/description/
beat 95% 36ms.
"""

```python
class Solution(object):
    def myAtoi(self, strs):
        """
        :type str: str
        :rtype: int
        """
        nums = '1234567890'
        signs = '+-'
        strs = strs.strip()
        if not strs or len(strs) == 1 and strs in signs:
            return 0
        if strs[0] in signs:
            sign = strs[0]
            strs = strs[1:]
        else:
            sign = '+'
        str_num = '0'
        x = 0
        for i in strs:
            if i == ' ' and str_num == '0':
                return 0
            if i not in nums:
                if sign == '+':
                    x = int(str_num)
                else:
                    x = -int(str_num)
                break
            else:
                str_num += i
        else:
            if sign == '+':
                x = int(str_num)
            else:
                x = -int(str_num)
        if x > 2**31-1:
            return 2**31-1
        elif x < -2**31:
            return -2**31
        else:
            return x
```

String/ToLowerCase.py

```
"""
将所有字符转换成小写字符。
Python 自带此函数。
若要自己写的话，我想到的是一个是建立起哈希表。
{'A': 'a'....}
这样查表，时间复杂度是 O(n)。
还有可以利用ASCII。
ord('a')
97
chr('A')
65
中间差了32个。
"""
class Solution(object):
    def toLowerCase(self, str):
        """
        :type str: str
        :rtype: str
        """
        return str.lower()
```

String/UniqueEmailAddresses.py

```
"""
Every email consists of a local name and a domain name, separated by the @ sign.
For example, in alice@leetcode.com, alice is the local name, and leetcode.com is
the domain name.
Besides lowercase letters, these emails may contain '.'s or '+'s.
If you add periods ('.') between some characters in the local name part of an
email address, mail sent there will be forwarded to the same address without dots
in the local name.  For example, "alice.z@leetcode.com" and "alicez@leetcode.com"
forward to the same email address.  (Note that this rule does not apply for
domain names.)
If you add a plus ('+') in the local name, everything after the first plus sign
will be ignored. This allows certain emails to be filtered, for example
m.y+name@email.com will be forwarded to my@email.com.  (Again, this rule does not
apply for domain names.)
It is possible to use both of these rules at the same time.
Given a list of emails, we send one email to each address in the list.  How many
different addresses actually receive mails?
Example 1:
Input:
["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com","testemail+
david@lee.tcode.com"]
Output: 2
Explanation: "testemail@leetcode.com" and "testemail@lee.tcode.com" actually
receive mails
Note:
1 <= emails[i].length <= 100
1 <= emails.length <= 100
Each emails[i] contains exactly one '@' character.
替换 . 和 + 即可。
测试地址:
https://leetcode.com/contest/weekly-contest-108/problems/unique-email-addresses/
"""
import re
class Solution(object):
    def numUniqueEmails(self, emails):
        """
        :type emails: List[str]
        :rtype: int
        """
        result = 0
        # local = {}
        _emails = set()
        ignore = re.compile(r'\+(.*)')
        for i in emails:
            x = i.split('@')
            if len(x) > 2:
                continue
            if len(x) == 1:
                continue
            local = x[0]
            domain = x[1]
            local = local.replace('.', '')
            local = re.sub(ignore, '', local)
            _emails.add(local + '@' + domain)
```

```
        return len(_emails)
```

String/ValidAnagram.py

```
"""
Given two strings s and t , write a function to determine if t is an anagram of
s.
Example 1:
Input: s = "anagram", t = "nagaram"
Output: true
Example 2:
Input: s = "rat", t = "car"
Output: false
Note:
You may assume the string contains only lowercase alphabets.
Follow up:
What if the inputs contain unicode characters? How would you adapt your solution
to such case?
判断 s 与 t 是否为变位词。
思路是排序 s 和 t，若相等则确定是变位词。
直接用 sorted 了。
效率 O(nlogn)
beat 61%.
前面都是用的 dict 的思路，dict 查询是 O(1) 总体也就是 O(n) 了。
dict 的思路基本是记录每个单词出现的次数，到0就剔除，最后若不剩余的话则表示是变位词，不写了，没有
难点。
测试地址：
https://leetcode.com/problems/valid-anagram/description/
"""
class Solution(object):
    def isAnagram(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        if sorted(s) == sorted(t):
            return True
        return False
```

String/ValidNumber.py

```
"""
Validate if a given string can be interpreted as a decimal number.
Some examples:
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
" -90e3   " => true
" 1e" => false
"e3" => false
" 6e-1" => true
" 99e2.5 " => false
"53.5e93" => true
" --6 " => false
"-+3" => false
"95a54e53" => false
Note: It is intended for the problem statement to be ambiguous. You should gather
all requirements up front before implementing one. However, here is a list of
characters that can be in a valid decimal number:
Numbers 0-9
Exponent - "e"
Positive/negative sign - "+"/"-"
Decimal point - "."
Of course, the context of these characters also matters in the input.
Update (2015-02-10):
The signature of the C++ function had been updated. If you still see your
function signature accepts a const char * argument, please click the reload
button to reset your code definition.
判断可否化为数字。
虽然使用float是作弊行为，但还是忍不住用了...
当做零启动任务吧。
测试地址：
https://leetcode.com/problems/valid-number/description/
"""
class Solution(object):
    def isNumber(self, s):
        """
        :type s: str
        :rtype: bool
        """
        try:
            float(s)
            return True
        except:
            return False
```

String/ValidPalindrome.py

```
"""
Given a string, determine if it is a palindrome, considering only alphanumeric
characters and ignoring cases.
Note: For the purpose of this problem, we define empty string as valid
palindrome.
Example 1:
Input: "A man, a plan, a canal: Panama"
Output: true
Example 2:
Input: "race a car"
Output: false
忽略特殊字符，空白，大小写。判断是否为回文字符串，空的话也为有效的。
关键字: re.
测试地址:
https://leetcode.com/problems/valid-palindrome/description/
"""
import re
class Solution(object):
    def isPalindrome(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if not s:
            return True
        s = s.lower()
        x = ''.join(re.findall(r'[a-z0-9]{1}', s))
        return x == x[::-1]
```

String/ValidParentheses.py

```
"""
Given a string containing just the characters '(', ')', '{', '}', '[' and ']',
determine if the input string is valid.
An input string is valid if:
Open brackets must be closed by the same type of brackets.
Open brackets must be closed in the correct order.
Note that an empty string is also considered valid.
Example 1:
Input: "()"
Output: true
Example 2:
Input: "()[]{}"
Output: true
Example 3:
Input: "(]"
Output: false
Example 4:
Input: "([)]"
Output: false
Example 5:
Input: "{[]}"
Output: true
用栈即可。
测试地址:
https://leetcode.com/problems/valid-parentheses/description/
beat 85%.
"""
class Solution(object):
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """
        left = "({["
        left_key = {')': '(', ']': '[', '}': '{'}
        stack = []
        for i in s:
            if i in left:
                stack.append(i)
            else:
                try:
                    if stack[-1] == left_key[i]:
                        stack.pop()
                    else:
                        return False
                except:
                    return False
        if stack:
            return False
        return True
```

String/WordPattern.py

```
"""
Given a pattern and a string str, find if str follows the same pattern.
Here follow means a full match, such that there is a bijection between a letter
in pattern and a non-empty word in str.
Example 1:
Input: pattern = "abba", str = "dog cat cat dog"
Output: true
Example 2:
Input:pattern = "abba", str = "dog cat cat fish"
Output: false
Example 3:
Input: pattern = "aaaa", str = "dog cat cat dog"
Output: false
Example 4:
Input: pattern = "abba", str = "dog dog dog dog"
Output: false
Notes:
You may assume pattern contains only lowercase letters, and str contains
lowercase letters separated by a single space.
与 IsomorphicString 一样，此题来自于做完IsomorphicString的相关推荐。
思路:
直接 one line 的思路:
转换成列表，然后一致。判断下字符位数的不同。
同样的思路换成这个题就能跑20ms..
beat 99%.
"""
class Solution(object):
    def wordPattern(self, pattern, string):
        """
        :type pattern: str
        :type str: str
        :rtype: bool
        """
        pattern = list(pattern)
        string = string.split(' ')
        if len(pattern) != len(string):
            return False
        temp = len(set(zip(pattern, string)))
        return temp == len(set(pattern)) and temp == len(set(string))
```

Tree/BinarySearchTree.py

```
"""
包括生成树，二叉搜索树的前后中遍历。
二叉搜索树在比较优质的情况下搜索和插入时间都是 O(logn) 的。在极端情况下会退化为链表 O(n)。
将无序的链表塞进二叉搜索树里，按左根右中序遍历出来就是排序好的有序链表。
"""
# 生成树操作。
class TreeNode(object):
    def __init__(self , val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
class binarySearchTree(object):
    """
        二叉搜索树，
        它的性质是左边节点都小于根节点，右边的都大于根节点。
        而且一般来说它是不存在重复元素的。
    """
    def __init__(self, root):
        if isinstance(root, TreeNode):
            print(1)
            self.root = root
        else:
            self.root = TreeNode(root)
    def add(self, value):
        # 从顶点开始遍历，找寻其合适的位置。
        root = self.root
        while 1:
            if root.val < value:
                if root.right is None:
                    if self.search(value):
                        break
                    root.right = TreeNode(value)
                    break
                else:
                    root = root.right
                    continue
            if root.val > value:
                if root.left is None:
                    if self.search(value):
                        break
                    root.left = TreeNode(value)
                    break
                else:
                    root = root.left
                    continue
            if root.val == value:
                break
    def search(self, value):
        # 查找一个值是否存在于这颗树中。
        return self._search(self.root, value)
    def _search(self, root, value):
        if root.val == value:
            return True
        if root.right:
```

```
            if root.val < value:
                return self._search(root.right, value)
        if root.left:
            if root.val > value:
                return self._search(root.left, value)
        return False
    def delete(self):
        pass
    def prevPrint(self):
        # 根左右
        nodes = [self.root]
        result = []
        while 1:
            if not nodes:
                return result
            node = nodes.pop()
            result.append(node.val)
            if node.right:
                nodes.append(node.right)
            if node.left:
                nodes.append(node.left)
    def _middlePrint(self, root, result):
        if root.left:
            self._middlePrint(root.left, result)
        result.append(root.val)
        if root.right:
            self._middlePrint(root.right,result)
    def middlePrint(self):
        # 左根右
        result = []
        self._middlePrint(self.root, result)
        return result
    def _suffPrint(self, root, result):
        if root.left:
            self._suffPrint(root.left, result)
        if root.right:
            self._suffPrint(root.right,result)
        result.append(root.val)
    def suffPrint(self):
        # 左右根
        result = []
        self._suffPrint(self.root, result)
        return result
oneTree = binarySearchTree(5)
for i in range(-5, 10):
    oneTree.add(i)
print(oneTree.middlePrint())
print(oneTree.suffPrint())
```

Tree/BinaryTreeMaximumPathSum.py

```
"""
Given a non-empty binary tree, find the maximum path sum.
For this problem, a path is defined as any sequence of nodes from some starting
node to any node in the tree along the parent-child connections. The path must
contain at least one node and does not need to go through the root.
Example 1:
Input: [1,2,3]
       1
      / \
     2   3
Output: 6
Example 2:
Input: [-10,9,20,null,null,15,7]
   -10
   / \
  9  20
    /  \
   15   7
Output: 42
给定一棵树：
找到一条路径，该路径所经过的节点之和是此树中最大的。
测试用例：
https://leetcode.com/problems/binary-tree-maximum-path-sum/description/
"""
"""
第一版：
此版本实现思路：
从底到顶，每次都会比对是否是最大值。
此版本的问题是输出的不是最大的路径而是所有节点中最大的相加和。
比如此树：
        5
       / \
      4   8
     /  /  \
   11  13   4
  / \        \
 7   2        1
从全部相加得出的是 55，而测试要求是48。
测试中所定义的*路径*是一条连通的线，可以 7 -> 11 -> 2，但不能 7 -> 11 -> 2 -> 4
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def maxPathSum(self, root):
        '''
        :type root: TreeNode
        :rtype: int
        '''
        self.maxes = -float('inf')
        def helper(root):
            myValue = root.val
```

```
            if not root.left and not root.right:
                self.maxes = myValue if myValue > self.maxes else self.maxes
                return myValue
            if root.left:
                value = helper(root.left)
                if value > 0:
                    myValue += value
                if myValue < value:
                    self.maxes = value if value > self.maxes else self.maxes
                else:
                    self.maxes = myValue if myValue > self.maxes else self.maxes
            if root.right:
                value = helper(root.right)
                if value > 0:
                    myValue += value
                if myValue < value:
                    self.maxes = value if value > self.maxes else self.maxes
                else:
                    self.maxes = myValue if myValue > self.maxes else self.maxes
            return myValue
        helper(root)
        return self.maxes
"""
"""
Passed! And Beat 94.66%.
第二版思路：
每一个点有两种决策：
1. 以此点为中转站的 left to right 的值，此值确定的是以此点为轴心的局部范围内是否有最大值。
   1
 /  \
2    3
2. 向上返回的值。此值确定的是以父节点为轴心的局部范围是否有最大值。
例子：
   10
 /   \
11   -20
     /  \
    15   5
在-20这个点它要向上返回的值是 15 -> -20 这一条线。
以-20为轴心的值是 15 -> -20 -> 5
1.中要进行的判断是，left + val + right, left + val, right + val, val 和 已记录的最大值哪个最大。4
2.要一直返回，返回之后的判断与1一致，2要返回的内容是 val, left + val和right + val 最大的那个。
最后叶节点比对下最大值直接返回即可。
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def maxPathSum(self, root):
        '''
        :type root: TreeNode
        :rtype: int
        '''
```

```
        self.maxes = -float('inf')
        def helper(root):
            myValue = root.val
            if not root.left and not root.right:
                self.maxes = myValue if myValue > self.maxes else self.maxes
                return myValue
            valueLeft, valueRight = -float('inf'), -float('inf')
            if root.left:
                valueLeft = helper(root.left)
            if root.right:
                valueRight = helper(root.right)
            # judge left to right is max or not.
            self.maxes = max([myValue + valueLeft, myValue + valueRight, myValue
+ valueLeft + valueRight, myValue, self.maxes])
            # return to parent
            return max(myValue + max(valueLeft, valueRight), myValue)
        helper(root)
        return self.maxes
```

Tree/BinaryTreeRightSideView.py

```
"""
Given a binary tree, imagine yourself standing on the right side of it, return
the values of the nodes you can see ordered from top to bottom.
Example:
Input: [1,2,3,null,5,null,4]
Output: [1, 3, 4]
Explanation:
   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
给一颗二叉树，返回从右边看能看到的第一个。
思路BFS，返回最后的一个即可。
beat 94%。
测试地址：
https://leetcode.com/problems/binary-tree-right-side-view/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def rightSideView(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if not root:
            return []
        result = [root.val]
        current = [root]
        next_nodes = []
        while current or next_nodes:
            for i in current:
                if i.left:
                    next_nodes.append(i.left)
                if i.right:
                    next_nodes.append(i.right)
            if next_nodes:
                result.append(next_nodes[-1].val)
            current = next_nodes
            next_nodes = []
        return result
```

Tree/CompleteBinaryTreeInserter.py

```
"""
A complete binary tree is a binary tree in which every level, except possibly the
last, is completely filled, and all nodes are as far left as possible.
Write a data structure CBTInserter that is initialized with a complete binary
tree and supports the following operations:
CBTInserter(TreeNode root) initializes the data structure on a given tree with
head node root;
CBTInserter.insert(int v) will insert a TreeNode into the tree with value
node.val = v so that the tree remains complete, and returns the value of the
parent of the inserted TreeNode;
CBTInserter.get_root() will return the head node of the tree.
Example 1:
Input: inputs = ["CBTInserter","insert","get_root"], inputs = [[[1]],[2],[]]
Output: [null,1,[1,2]]
Example 2:
Input: inputs = ["CBTInserter","insert","insert","get_root"], inputs =
[[[1,2,3,4,5,6]],[7],[8],[]]
Output: [null,3,4,[1,2,3,4,5,6,7,8]]
Note:
The initial given tree is complete and contains between 1 and 1000 nodes.
CBTInserter.insert is called at most 10000 times per test case.
Every value of a given or inserted node is between 0 and 5000.
一只完全二叉树是除最后一层外，每一层的节点都是满的，而且尽量靠左。
实现一个 CBTInserter，这个东西会初始化一颗二叉树，并支持以下操作：
1. CBTInserter(TreeNode root) 初始化这个结构。
2. CBTInserter.insert (int v) 会插入v到这个结构中，并且返回它的父节点。
3. CBTInserter.get_root() 返回树的根节点。
思路：
初始化思路：
根据 BFS 的路线：
    一个root，一个当前节点的集合，一个下一层节点的集合。
    初始化问题，初始化时只给了一颗完整的树，要自己找出当前节点的集合和下一层节点的集合，
    直接用了 BFS 层级遍历然后依次调用 insert 了。
    这样初始化是 O（n）之后都是 O（1）。
insert 思路：
    使用 list + reverse 代替的先进先出queue了。
    选当前节点最先插入的一个，然后判断 left 和 right，插入right后把它从当前层里删除，
    若当前层不存在节点就替换为_next_nodes。
    这样插入是 O(1) 的。
get_root：
    return self.root。
contest，还没有beat。
run time 52ms.
测试地址：
https://leetcode.com/contest/weekly-contest-105/problems/complete-binary-tree-
inserter/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class CBTInserter(object):
```

```
    def __init__(self, root):
        """
        :type root: TreeNode
        """
        self._root = root
        self.root = TreeNode(root.val)
        self.current_node = [self.root]
        self._next_node = []
        self.get_init_nodes()
    def get_init_nodes(self):
        result = []
        current_nodes = [self._root]
        _next_node = []
        while current_nodes or _next_node:
            for i in current_nodes:
                if i.left:
                    result.append(i.left.val)
                    _next_node.append(i.left)
                if i.right:
                    result.append(i.right.val)
                    _next_node.append(i.right)
            current_nodes = _next_node
            _next_node = []
        for i in result:
            self.insert(i)
    def insert(self, v):
        """
        :type v: int
        :rtype: int
        """
        node = self.current_node[-1]
        if not node.left:
            node.left = TreeNode(v)
            parent = node
            self._next_node.append(node.left)
        elif not node.right:
            node.right = TreeNode(v)
            parent = node
            self._next_node.append(node.right)
            self.current_node.pop()
        if not self.current_node:
            self._next_node.reverse()
            self.current_node = self._next_node
            self._next_node = []
        return parent.val
    def get_root(self):
        """
        :rtype: TreeNode
        """
        return self.root
# Your CBTInserter object will be instantiated and called as such:
# obj = CBTInserter(root)
# param_1 = obj.insert(v)
# param_2 = obj.get_root()
```

Tree/ConstructBinaryTreeFromInorderAndPostorderTraversal.py

```
"""
Given inorder and postorder traversal of a tree, construct the binary tree.
Note:
You may assume that duplicates do not exist in the tree.
For example, given
inorder = [9,3,15,20,7]
postorder = [9,15,7,20,3]
Return the following binary tree:
    3
   / \
  9  20
    /  \
   15   7
这个的思路与之前的大同小异。
inorder:
左 根 右
postorder:
左 右 根
postorder 中找根，
inorder 中找左右。
下面是一个递归实现。
left_inorder
left_postorder
和
right_inorder
right_postorder
的处理。
一开始全部中规中矩的定义清晰，然后root.left, root.right。
完成所有测试大概需要 200ms 左右。
后面发现并不需要：
postoder 是 左 右 根。
根完了就是右，所以直接可以postorder.pop()，然后先进行 right 的查找，相当于
right_postorder 带了一些另一颗树的东西，不过无关紧要。
都是些优化的步骤。
测试地址：
https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def buildTree(self, inorder, postorder):
        """
        :type inorder: List[int]
        :type postorder: List[int]
        :rtype: TreeNode
        """
        def makeTree(inorder,
                     postorder):
```

```
            if not inorder or not postorder:
                return None
            root = TreeNode(postorder.pop())
            index = inorder.index(root.val)
            # left_inorder = inorder[:inorder.index(root.val)]
            # left_postorder = postorder[:len(left_inorder)]
            # right_inorder = inorder[len(left_inorder)+1:]
            # right_postorder = postorder[len(left_postorder):-1]
            root.right = makeTree(inorder[index+1:], postorder)
            root.left = makeTree(inorder[:index], postorder)
            return root
        return makeTree(inorder, postorder)
```

Tree/ConstructBinaryTreeFromPreorderAndInorderTraversal.py

```
"""
Given preorder and inorder traversal of a tree, construct the binary tree.
Note:
You may assume that duplicates do not exist in the tree.
For example, given
preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]
Return the following binary tree:
    3
   / \
  9  20
    /  \
   15   7
给定中序和前序遍历返回完整的二叉树。
就思路上来说比较容易理解：
1. 前序是 根 左 右。
2. 中序是 左 根 右。
也就是在 前序中找到根，然后在中序中找到根的左右两颗子树。不断的重复左右两颗子树这样的过程。
下面是一个递归实现，效率并不是非常高：
beat 40% ~ 50%.
测试地址：
https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def buildTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: TreeNode
        """
        def make(preorder, inorder):
            if not preorder:
                return None
            root = TreeNode(preorder[0])
            left_in = inorder[:inorder.index(root.val)]
            left_pre = preorder[1:len(left_in)+1]
            right_in = inorder[len(left_in)+1:]
            right_pre = preorder[len(left_in)+1:]
            root.left = make(left_pre, left_in)
            root.right = make(right_pre, right_in)
            return root
        return make(preorder, inorder)
```

Tree/ConstructBinaryTreeFromPreorderAndPostorderTraversal.py

```
"""
Return any binary tree that matches the given preorder and postorder traversals.
Values in the traversals pre and post are distinct positive integers.
Example 1:
Input: pre = [1,2,4,5,3,6,7], post = [4,5,2,6,7,3,1]
Output: [1,2,3,4,5,6,7]
Note:
1 <= pre.length == post.length <= 30
pre[] and post[] are both permutations of 1, 2, ..., pre.length.
It is guaranteed an answer exists. If there exists multiple answers, you can
return any of them.
根据二叉树的 前序和后序遍历，返回一颗完整的二叉树。
不唯一，返回随便一个即可。
思路：
1. 二叉树的前序是 根左右。
2. 二叉树的后序是 左右根。
在前序中确定 根 ，然后在后序中找左右子树。
pre = [1,2,4,5,3,6,7]
post = [4,5,2,6,7,3,1]
总根是 1
1的左右其中一个是 2，就当它是左子树好了，因为是 根 左右 所以假设的话就先假设为左子树，如果只有一边的话，左右其实无所谓。
   1
 /
2
然后在 post 中找属于 2 这个子树的节点。
找到 4 5 2，那么剩下的 6 7 3 就是与之相对的 1 的右子树了。
把 pre 分成 [2, 4, 5] [3, 6, 7]
   post 分为 [4, 5, 2] [6, 7 ,3]
这样 作为1的左右两颗子树已经出来了。
pre[left] 和 pre[right] 的 0 分别为 左右两棵子树的根。
之后就是分别把左右两边的这两个代替原来的 pre post，根也同样代替，然后递归直到没有即可。
测试链接：
https://leetcode.com/contest/weekly-contest-98/problems/construct-binary-tree-from-preorder-and-postorder-traversal/
beat 50% 40ms.
这应该是自己的极限了。
4道题 一个半小时做 3 道题，1 easy 2 medium 0 hard.
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def constructFromPrePost(self, pre, post):
        """
        :type pre: List[int]
        :type post: List[int]
        :rtype: TreeNode
        """
        def getLeftAndRight(pre, post):
```

```
            # no more node.
            if not pre:
                return None
            # Get the index of the left root.
            index = post.index(pre[0])
            # post left tree
            val_children = post[:index+1]
            # post right tree
            val_brother = post[index+1:-1]
            # pre left tree
            # Get the left tree pre list
            # if left tree post list contains 3 elements
            # then we will get equal in pre list.
            t = len(val_children)
            pre_val_children = pre[:t]
            # there is right tree.
            # The elements are the rest of pre list.
            pre_val_brother = pre[t:]
            left = pre[0]
            right = val_brother[-1] if val_brother else None
            # left, right, pre, post
            return (left, right, pre_val_children, val_children,
pre_val_brother, val_brother)
        def construct(root, pre, post):
            x = getLeftAndRight(pre[1:], post)
            if root and x:
                if x[0] is not None:
                    root.left = TreeNode(x[0])
                if x[1] is not None:
                    root.right = TreeNode(x[1])
                if root.left:
                    construct(root.left, x[2], x[3])
                if root.right:
                    construct(root.right, x[4], x[5])
        allRoot = TreeNode(pre[0])
        construct(allRoot, pre, post)
        return allRoot
```

Tree/ConvertSortedArrayToBinarySearchTree.py

```
"""
Given an array where elements are sorted in ascending order, convert it to a
height balanced BST.
For this problem, a height-balanced binary tree is defined as a binary tree in
which the depth of the two subtrees of every node never differ by more than 1.
Example:
Given the sorted array: [-10,-3,0,5,9],
One possible answer is: [0,-3,9,-10,null,5], which represents the following
height balanced BST:
      0
     / \
   -3   9
   /   /
 -10  5
 给定一个已排序过的数组，将它转换为一颗高度平衡的二叉搜索树。
 也就是两颗子树的高度差不超过1。
 因为是排序的数组，相对来说也异常简单，可以将它看做是一颗二叉搜索树中序遍历后的结果。
 按照此结果转换回去就是了。
 每次都二分：
 [-10,-3,0,5,9]
        mid
                  5//2 = 2 mid = 2
             0
[-10, -3]       [5, 9]   2 // 2 = 1 mid = 1
    ↓             ↓
    -3            9
[-10]  []     [5]    []
为空则不进行下面的操作。
beat 98%
测试地址：
https://leetcode.com/problems/convert-sorted-array-to-binary-search-
tree/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def sortedArrayToBST(self, nums):
        """
        :type nums: List[int]
        :rtype: TreeNode
        """
        if not nums:
            return None
        def makeBinarySearchTree(nums):
            mid = len(nums) // 2
            root = TreeNode(nums[mid])
            left = nums[:mid]
            right = nums[mid+1:]
            if left:
                root.left = makeBinarySearchTree(left)
```

```
            if right:
                root.right = makeBinarySearchTree(right)
            return root
        root = makeBinarySearchTree(nums)
        return root
```

Tree/ConvertSortedListToBinarySearchTree.py

```
"""
Given a singly linked list where elements are sorted in ascending order, convert
it to a height balanced BST.
For this problem, a height-balanced binary tree is defined as a binary tree in
which the depth of the two subtrees of every node never differ by more than 1.
Example:
Given the sorted linked list: [-10,-3,0,5,9],
One possible answer is: [0,-3,9,-10,null,5], which represents the following
height balanced BST:
      0
     / \
   -3   9
   /   /
 -10  5
 这次给的是一个排序过的链表，链表和数组有所不同，链表的话无法使用索引，或者说使用索引所需要的时间是 O(n) 并非  O(1)。
 当然可以把链表转换成一个数组然后按照数组的方法去解，这样不会出错，时间复杂度上也是同样的，就是空间复杂度上要高一些。
 我自己的话没想到其他的思路：
 在Discuss里看到一个 Java 的思路，觉得非常棒：
 前面我们分析过这其实就是个中序遍历的结果，按照这个思路，如果能按照中序遍历逆推回去，即可得到一颗完整的高度平衡的二叉搜索树。
 这个思路也是这样的，在做二叉树的中序遍历时用递归一般这样写：
```

```
if root.left:
  recursive(root.left)
root.val
if root.right:
  recursive(root.right)
```

```
 如果我们能找到left的头，并一直持续到right的尾，即可得到一颗二叉搜索树，这棵树可能并不会与原来的相同。
 如：
  中序结果是：
  [-1, 1, 2, 3]
  这颗树可能是：
       1
    /   \
 -1      2
           \
            3
也可以是：
         2
       /  \
     1     3
   /
-1
 那就按原来的数组中的方法：
 如果要从中序遍历的结果生成二叉树，首先需要获取的是 mid 中位，找到它的根。剩下的也是不断找到根。
 [-10,-3,0,5,9]
 1. 第一步先找一下链表的长度。
 2. 第二步则给函数说左有几个，右有几个。
 左边有几个的话很简单：
```

直接 length // 2即可，地板除的话会舍弃。 比如如果有4个数据。 4//2之后左边的还剩下 两个 [0,1]
右边的话：
需要原来的长度 减去左边的长度 再减去 这个的根得知。
这样不断递归至 size 为 0 即为左子树的头，与右子树的尾。
[-10,-3,0,5,9]
1. size = 5
left = 5//2 = 2 [-10,-3]
right = 5 - left - 1 = 5 - 2 - 1 = 2 [5, 9]
2. size = 1.left = 2
left = 2//1 = 1 [-10]
right = 2 - left - 1 = 2 - 1 -1 = 0 []
3. size = 2.left = 1
left = 1 // 2 = 0
right = 1 - 0 - 1 = 0
这一步开始返回链表的第一个值 -10 作为 2 里左节点的val.之后返回到2
2. 则把自己的节点值覆盖为链表中下一个值 -3。 之后返回到1.
1. 则把自己的节点值覆盖为链表中的下一个值 0。 之后开始right的递归。同样的操作。
---
4. size = 1.right = 2
left = 2//1 = 1 [5]
right = 2 - left - 1 = 2 -1 -1 =0 []
5. size = 4.left = 1
left = 1 // 2 = 0 []
right = 1- 0 -1 = 0 []
...
关键词：
中序遍历，左右子树节点的个数。
beat 99%
测试地址:
https://leetcode.com/problems/convert-sorted-list-to-binary-search-tree/description/
"""

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def sortedListToBST(self, head):
        """
        :type head: ListNode
        :rtype: TreeNode
        """
        size = 0
        self.c_head = head
        while head:
            size += 1
            head = head.next
        def makeBSTByInorder(size):
            if not size:
                return
```

```
            root = TreeNode(None)
            root.left = makeBSTByInorder(size//2)
            root.val = self.c_head.val
            self.c_head = self.c_head.next
            root.right = makeBSTByInorder(size-size//2-1)
            return root
        return makeBSTByInorder(size)
```

Tree/KthSmallestElementInABST.py

```
"""
Given a binary search tree, write a function kthSmallest to find the kth smallest
element in it.
Note:
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.
Example 1:
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
Example 2:
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
Follow up:
What if the BST is modified (insert/delete operations) often and you need to find
the kth smallest frequently? How would you optimize the kthSmallest routine?
取出二叉搜索树中第 k 小的数据。
就这一条来看用 inorder 即可。
进阶条件是如果这颗树经常进行 插入/删除操作如何去优化它呢?
以下是思考:
一颗相对平衡的 BST 的优势在于:可以在 O(log n) 时间内查找/插入/删除某些数据。
就查找这一个条件来说构建一个 排序过的数组是可以达到 O(log n) 的需求的。 但是插入和删除对于数组
来说都是 O（n）级别的。
Discuss里有讨论说可以记录 第 k ，第 k-1 个。若插入的比 k 大那么不变，否则k就变为 k-1，然后在
重新计算 k - 1。这样也是部分优化。
待解决。
非进阶的:
beat 78% 48ms。
测试地址:
https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def find_data(self, root: TreeNode):
        if root is None:
            return
        Solution.find_data(self,root.left)
        self.data.append(root.val)
        Solution.find_data(self,root.right)
```

```
        return
    def kthSmallest(self, root: TreeNode, k: int) -> int:
        self.data = []
        Solution.find_data(self, root)
        return self.data[k-1]
```

Tree/LowestCommonAncestorOfABinarySearchTree.py

```
"""
Given a binary search tree (BST), find the lowest common ancestor (LCA) of two
given nodes in the BST.
According to the definition of LCA on Wikipedia: “The lowest common ancestor is
defined between two nodes p and q as the lowest node in T that has both p and q
as descendants (where we allow a node to be a descendant of itself).”
Given binary search tree:  root = [6,2,8,0,4,7,9,null,null,3,5]
        _______6______
       /              \
    ___2__          ___8__
   /      \        /      \
   0      _4       7       9
         /  \
         3   5
Example 1:
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
Example 2:
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of
itself
             according to the LCA definition.
Note:
All of the nodes' values will be unique.
p and q are different and both values will exist in the BST.
给定一颗 BST，找到两个子节点的最小公共祖先。
用普通的树的方法也是可以的，不过可以做个剪枝优化。
因为 BST 我们知道每个节点的左右子节点的范围。
1. 如果处于 root 左右，那么直接返回即可。
2. 如果都小于，那么去找左子树。
3. 如果都大，那么去找右子树。
在寻找过程中，
4. 只要有一个命中了，那么直接返回当前节点即可。因为剩下的那个节点只有可能在它的子树中，如果不在
它的子树中也不会执行到这一步。
beat 99%
测试地址：
https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-
tree/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
```

```
        """
        if p.val == root.val or q.val == root.val:
            return root
        if p.val < root.val and q.val > root.val:
            return root
        elif p.val > root.val and q.val < root.val:
            return root
        if p.val > root.val and q.val > root.val:
            return self.lowestCommonAncestor(root.right, p, q)
        else:
            return self.lowestCommonAncestor(root.left, p, q)
```

Tree/LowestCommonAncestorOfABinaryTree.py

```
"""
Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in
the tree.
According to the definition of LCA on Wikipedia: “The lowest common ancestor is
defined between two nodes p and q as the lowest node in T that has both p and q
as descendants (where we allow a node to be a descendant of itself).”
Given the following binary tree:  root = [3,5,1,6,2,0,8,null,null,7,4]
        _______3______
       /              \
    ___5__          ___1__
   /      \        /      \
   6      _2       0       8
         /  \
         7   4
Example 1:
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of of nodes 5 and 1 is 3.
Example 2:
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of
itself
             according to the LCA definition.
Note:
All of the nodes' values will be unique.
p and q are different and both values will exist in the binary tree.
给一颗二叉树，找出某两个子节点的最小公共祖先。
思路：
用递归：
   1. 用递归找到符合条件的子节点。 找到后的子节点会返回为一个具体的 TreeNode，找不到的话则是
None。
   2. 之后判断 是不是两个都找到了，最先知道两个都找到的点即为最小公共祖先。
   3. q为p子节点，或p为q子节点的情况：
          由于是唯一的，所以出现这种情况一定有一边返回是None，所以返回不是None的一边即可。
普通的二叉树要递归的话是这样：
# do something
if root.right:
    right = recursive(root.right)
if root.left:
    left = recursive(root.left)
# do something
按照上面的思路加工一下即可。
测试地址：
https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-
tree/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
```

```
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        if root.val == p.val or root.val == q.val:
            return root
        right = None
        left = None
        if root.right:
            right = self.lowestCommonAncestor(root.right, p, q)
        if root.left:
            left = self.lowestCommonAncestor(root.left, p, q)
        if right and left:
            return root
        if right:
            return right
        if left:
            return left
```

Tree/PathSumII.py

```
"""
Given a binary tree and a sum, find all root-to-leaf paths where each path's sum
equals the given sum.
Note: A leaf is a node with no children.
Example:
Given the below binary tree and sum = 22,
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \    / \
7    2  5   1
Return:
[
   [5,4,11,2],
   [5,8,4,5]
]
PathSum 的进阶版，输出所有符合条件的路径。所以这里直接用遍历，有负数加上不是二叉搜索树应该没太
多需要优化的地方。
测试用例：
https://leetcode.com/problems/path-sum-ii/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def pathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: List[List[int]]
        """
        if not root:
            return []
        result = []
        def helper(prev, root, sum, path):
            if prev + root.val == sum:
                if not root.left and not root.right:
                    result.append(list(map(int, path.split(' ')[1:]))+
[root.val])
                    return True
            if root.left:
                helper(prev + root.val, root.left, sum, path=path+"
"+str(root.val))
                    # return True
            if root.right:
                helper(prev + root.val, root.right, sum, path=path+"
"+str(root.val))
            return False
        helper(0, root, sum, "")
```

```
        return result
```

Tree/PopulatingNextRightPointersInEachNode.py

```
"""
Given a binary tree
struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
Populate each next pointer to point to its next right node. If there is no next
right node, the next pointer should be set to NULL.
Initially, all next pointers are set to NULL.
Note:
You may only use constant extra space.
Recursive approach is fine, implicit stack space does not count as extra space
for this problem.
You may assume that it is a perfect binary tree (ie, all leaves are at the same
level, and every parent has two children).
Example:
Given the following perfect binary tree,
     1
   /  \
  2    3
 / \  / \
4  5  6  7
After calling your function, the tree should look like:
     1 -> NULL
   /  \
  2 -> 3 -> NULL
 / \  / \
4->5->6->7 -> NULL
给一颗二叉树，二叉树是满二叉树，每个节点默认 next 为 None，将每个节点的 next 指向它右边的一个。
需要使用 O（1）空间..
O(1) 空间没什么思路，目前用的两个列表存放Node.
也就是BFS.
beat 88%
测试地址：
https://leetcode.com/problems/populating-next-right-pointers-in-each-
node/description/
"""
# Definition for binary tree with next pointer.
# class TreeLinkNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None
class Solution:
    # @param root, a tree link node
    # @return nothing
    def connect(self, root):
        if not root:
            return
        current = [root]
        next_nodes = []
```

```
        while current or next_nodes:
            for i in current:
                if i.left:
                    if next_nodes:
                        next_nodes[-1].next = i.left
                    next_nodes.append(i.left)
                if i.right:
                    if next_nodes:
                        next_nodes[-1].next = i.right
                    next_nodes.append(i.right)
            current = next_nodes
            next_nodes = []
```

Tree/PopulatingNextRightPointersInEachNodeII.py

```
"""
Given a binary tree
struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
Populate each next pointer to point to its next right node. If there is no next
right node, the next pointer should be set to NULL.
Initially, all next pointers are set to NULL.
Note:
You may only use constant extra space.
Recursive approach is fine, implicit stack space does not count as extra space
for this problem.
Example:
Given the following binary tree,
     1
   /  \
  2    3
 / \    \
4   5    7
After calling your function, the tree should look like:
     1 -> NULL
   /  \
  2 -> 3 -> NULL
 / \    \
4-> 5 -> 7 -> NULL
使用 BFS 和列表的额外空间的话 I 和 II没有任何区别...
待添加 O(1) 空间算法。
beat 72%.
测试地址:
https://leetcode.com/problems/populating-next-right-pointers-in-each-node-
ii/description/
"""
# Definition for binary tree with next pointer.
# class TreeLinkNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None
class Solution:
    # @param root, a tree link node
    # @return nothing
    def connect(self, root):
        if not root:
            return
        current = [root]
        next_nodes = []
        while current or next_nodes:
            for i in current:
                if i.left:
                    if next_nodes:
                        next_nodes[-1].next = i.left
```

```
                next_nodes.append(i.left)
            if i.right:
                if next_nodes:
                    next_nodes[-1].next = i.right
                next_nodes.append(i.right)
        current = next_nodes
        next_nodes = []
```

Tree/serializeAndDeserialize.py

```
"""
Given the root to a binary tree, implement serialize(root),
which serializes the tree into a string, and deserialize(s),
which deserializes the string back into the tree.
给定一颗二叉树，
serialize(root) 方法可以将此树弄成字符串，
deserialize()则可以将转换成的字符串还原为树。
这个要求让我想到翻译的一章Json，里有一个序列化自定义对象。
序列化的时候，弄出它的 __class__，__dict__。
在这里，root 是字符串，不用做特殊处理，left和right要么是None，要么是Node。
但在Python的魔法方法中，有一种更好用的方式，思路还是Json，也要用到Json。
因为是要转换为字符串，直接定义 __str__方法，返回
"{{'val': {}, 'left': {}, 'right': {}}}".format(self.val, self.left,
self.right)"
这样只要调用一次 str，剩下的如果left和right是Node，则也会调用同样的 __str__方法，最终形成一个嵌套字典。
标准的Json，要转换下引号。
在解包的时候，用Json处理一下，然后循环，如果left/right是字典，就写成Node，直到left或right是None。
这一步用递归比较容易。同时也要更改下Node，在构造left的时候，如果是字典，就要用Node封装，如果是Node或者None，则不管。
遇到的问题：
在转换为Json的过程中，
```

```
def _serialize(self):
    return {"val": self.val, "left": self.left or self.left._serialize(),
"right": self.right or self.right._serialize()}
def serialize(self):
    # 会提示不是可序列化的目标。
    # self._serialize()
    # 返回的是个Dict.
    return json.dumps(self._serialize())
```

```
Ok, a silly wrong. The statement `or` will return the first if the first is True
or return the second when the first is False.
So, if self.left is not None it will return <class.__main__.Node ....> but not
self.left._serialize().
The key of the question is I rewrite the `__str__` and `__repr__`,
so it will show the same output of `_serialize()` when I printed it...
To solve it just replace `or` to `and`. `and` statement will return the second
when the first is True.
"""
import json
class Node:
    def __init__(self, val, left=None, right=None):
        self.val = val
        if isinstance(left, Node) or left == None:
            self.left = left
        else:
            self.left = self.construct(left)
        if isinstance(right, Node) or right == None:
            self.right = right
        else:
```

```
            self.right = self.construct(right)
    def __str__(self):
        return '{{"val": "{}", "left": {}, "right": {}}}'.format(self.val,
self.left, self.right)
    def __repr__(self):
        return str(self)
    def _serialize(self):
        return {"val": self.val, "left": self.left and self.left._serialize(),
"right": self.right and self.right._serialize()}
    def construct(self, constructDict):
        return Node(**constructDict)
    def serialize(self):
        return json.dumps(self._serialize())
def deserialize(string):
    constructDict = json.loads(string)
    def construct(treeDict):
        return Node(treeDict.get('val'), treeDict.get('left'),
treeDict.get('right'))
    return construct(constructDict)
# test
node = Node('root', Node('left', Node('left.left')), Node('right'))
print(node.serialize())
assert deserialize(node.serialize()).left.left.val == 'left.left'
```

Tree/SerializeAndDeserializeBinaryTree.py

```
"""
Serialization is the process of converting a data structure or object into a
sequence of bits so that it can be stored in a file or memory buffer, or
transmitted across a network connection link to be reconstructed later in the
same or another computer environment.
Design an algorithm to serialize and deserialize a binary tree. There is no
restriction on how your serialization/deserialization algorithm should work. You
just need to ensure that a binary tree can be serialized to a string and this
string can be deserialized to the original tree structure.
Example:
You may serialize the following tree:
    1
   / \
  2   3
     / \
    4   5
as "[1,2,3,null,null,4,5]"
Clarification: The above format is the same as how LeetCode serializes a binary
tree. You do not necessarily need to follow this format, so please be creative
and come up with different approaches yourself.
Note: Do not use class member/global/static variables to store states. Your
serialize and deserialize algorithms should be stateless.
二叉树的“结”构与解构，在很久以前的一个同样的思路中用了 JSON，因为那个没有标明字符串的输出形式，
所以用JSON可以直接用字典来写明每个点的 left, right, val。
Leetcode中的这个需要是列表形式的，所以不用 JSON 的思路了。
按照这个形式，可以以 层 来分级。
也就是 BFS 的思路：
    1
   / \
  2   3
     / \
    4   5
第一层是
1
第二层是
2 3
第三层是
None None 4 5
第四层全是 None。
serialize 的话比较好写：
如果节点不为 None，把val加入到result中，left和right都加到下一层节点里。
       为 None 的话就加 None 到result里。
最后处理下尾部的None即可。
deserialize 的话：
目前也是同样的思路
1 2 4 8 16 这样的递增，除了最后一层肯定是可以满满的排满的。
一个是 roots，上一层的roots，一个是nodes，要给roots加 right 和 left 的nodes。
有一个点需要注意：
上一层serialize之后的 None 的处理，要处理成 'null'，否则 Leetcode 不通过。
beat 62%
测试地址：
https://leetcode.com/problems/serialize-and-deserialize-binary-tree/description/
---
如果只为了通过测试：
```

```
serialize 直接 return root
deserialize 直接 return data...
因为它的测试是
# codec = Codec()
# codec.deserialize(codec.serialize(root))
排在最前面的几个是这样做的 = =...
前面的大神写的也有很厉害的：
class Codec:
    def serialize(self, root):
        '''Encodes a tree to a single string.
        :type root: TreeNode
        :rtype: str
        '''
        def doit(node):
            if node:
                vals.append(node.val)
                doit(node.left)
                doit(node.right)
            else:
                vals.append('#')
        vals = []
        doit(root)
        return vals
    def deserialize(self, data):
        '''Decodes your encoded data to tree.
        :type data: str
        :rtype: TreeNode
        '''
        def doit():
            val = next(vals)
            if val == '#':
                return None
            else:
                node = TreeNode(val)
                node.left = doit()
                node.right = doit()
                return node
        vals = iter(data)
        return doit()
思路非常流畅，看过之后也是恍然大悟的感觉，比上面的这个思路要快 20ms 左右。
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Codec:
    def serialize(self, root):
        """Encodes a tree to a single string.
        :type root: TreeNode
        :rtype: str
        """
        if not root:
            return None
        result = []
        def _serialize(roots):
            _next = []
```

```
            for i in roots:
                if i:
                    result.append(i.val)
                    _next.append(i.left)
                    _next.append(i.right)
                else:
                    result.append(None)
            return _next
        base = _serialize([root])
        while any(base):
            base = _serialize(base)
        while 1:
            if result[-1] == None:
                result.pop()
            else:
                break
        return str(result)
    def deserialize(self, data):
        """Decodes your encoded data to tree.
        :type data: str
        :rtype: TreeNode
        """
        if not data:
            return []
        data = data[1:-1].split(',')
        root = TreeNode(data[0])
        length = 2
        data.pop(0)
        leaves = [root]
        def _deserialize(roots, nodes):
            _next = []
            for i in roots:
                if not i:
                    continue
                if nodes:
                    val = nodes.pop(0)
                    if val == ' None':
                        val = 'null'
                    else:
                        val = int(val)
                    i.left = TreeNode(val)
                    _next.append(i.left)
                if nodes:
                    val = nodes.pop(0)
                    if val == ' None':
                        val = 'null'
                    else:
                        val = int(val)
                    i.right = TreeNode(val)
                    _next.append(i.right)
            return _next
        base = _deserialize(leaves, data[:length])
        data = data[length:]
        length *= 2
        while data:
            base = _deserialize(base, data[:length])
            data = data[length:]
            length *= 2
```

```
        return root
# Your Codec object will be instantiated and called as such:
# codec = Codec()
# codec.deserialize(codec.serialize(root))
```

Tree/SumRootToLeafNumbers.py

```
"""
Given a binary tree containing digits from 0-9 only, each root-to-leaf path could
represent a number.
An example is the root-to-leaf path 1->2->3 which represents the number 123.
Find the total sum of all root-to-leaf numbers.
Note: A leaf is a node with no children.
Example:
Input: [1,2,3]
    1
   / \
  2   3
Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.
Example 2:
Input: [4,9,0,5,1]
    4
   / \
  9   0
 / \
5   1
Output: 1026
Explanation:
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
给一只二叉树，输出所有从根到叶路径的总和。
O(n) 遍历。 字符串 -> 数字 -> 求和。
测试用例：
https://leetcode.com/problems/sum-root-to-leaf-numbers/description/
24 ms beat 75%
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def sumNumbers(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        result = []
        if not root:
            return 0
        def helper(root, string):
            if not root.left and not root.right:
                result.append(int(string+str(root.val)))
                return
```

```
            if root.left:
                helper(root.left, string+str(root.val))
            if root.right:
                helper(root.right, string+str(root.val))
        helper(root, '')
        return sum(result)
```

Tree/SymmetricTree.py

```
"""
Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).
For example, this binary tree [1,2,2,3,4,4,3] is symmetric:
    1
   / \
  2   2
 / \ / \
3  4 4  3
But the following [1,2,2,null,3,null,3] is not:
    1
   / \
  2   2
   \   \
   3    3
Note:
Bonus points if you could solve it both recursively and iteratively.
判断左子树是否与右子树互为镜像。
思路:
遍历左子树，把结果放入队列中。
按照相反的左右遍历右子树，同时让压出队列顶的一项作对比。
不匹配或队列中没有足够的数据都可判为False.
效率 O(n)
beat 100%.
测试地址:
https://leetcode.com/problems/symmetric-tree/description/
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def isSymmetric(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
            return True
        left = root.left
        right = root.right
        left_node = [left]
        right_node = [right]
        left_node_value = []
        while left_node:
            # if left_node:
            _left = left_node.pop()
            if _left:
                left_node_value.append(_left.val)
            else:
                left_node_value.append(None)
            if _left:
```

```
                left_node.append(_left.right)
                left_node.append(_left.left)
        left_node_value.reverse()
        while right_node:
            _right = right_node.pop()
            if left_node_value:
                left_value = left_node_value.pop()
            else:
                return False
            if _right:
                if left_value != _right.val:
                    return False
            else:
                if left_value != None:
                    return False
            if _right:
                right_node.append(_right.left)
                right_node.append(_right.right)
        return True
```

Tree/Trie.py

```
"""
前缀树又叫字典树，该树会覆盖多个相同的字符以形成空间上的优势。
如：
rat 与 rain
  r
  a
t  i
   n
最终会形成这样的树。
字典树有多种实现方式，下面直接用了列表（数组）来实现。
测试用例：
https://leetcode.com/problems/implement-trie-prefix-tree/description/
使用Python 中的字典可以直接形成这种树，所以弃用这种方式，用类的思路实现了一下。
"""
class TrieNode(object):
    # __slots__ 考虑到TrieNode会大量创建，使用 __slot__来减少内存的占用。
    # 在测试的15个例子中：
    # 使用 __slots__会加快创建，平均的耗时为290ms-320ms。
    # 而不使用则在 340ms-360ms之间。
    # 创建的越多效果越明显。
    # 当然，使用字典而不是类的方式会更加更加更加高效。
    __slots__ = {'value', 'nextNodes', 'breakable'}
    def __init__(self, value, nextNode=None):
        self.value = value
        if nextNode:
            self.nextNodes = [nextNode]
        else:
            self.nextNodes = []
        self.breakable = False
    def addNext(self, nextNode):
        self.nextNodes.append(nextNode)
    def setBreakable(self, enable):
        self.breakable = enable
    def __eq__(self, other):
        return self.value == other
class Trie(object):
    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.root = []
    def insert(self, word):
        """
        Inserts a word into the trie.
        :type word: str
        :rtype: void
        """
        self.makeATrieNodes(word)
    def search(self, word):
        """
        Returns if the word is in the trie.
        :type word: str
        :rtype: bool
        """
```

```
        for i in self.root:
            if i == word[0]:
                return self._search(i, word[1:])
        return False
    def _search(self, root, word):
        if not word:
            if root.breakable:
                return True
            return False
        if not root.nextNodes:
            return False
        for i in root.nextNodes:
            if i == word[0]:
                return self._search(i, word[1:])
        return False
    def startsWith(self, prefix):
        """
        Returns if there is any word in the trie that starts with the given
prefix.
        :type prefix: str
        :rtype: bool
        """
        for i in self.root:
            if i == prefix[0]:
                return self._startWith(i, prefix[1:])
        return False
    def _startWith(self, root, prefix):
        if not prefix:
            return True
        if not root.nextNodes:
            return False
        for i in root.nextNodes:
            if i == prefix[0]:
                return self._startWith(i, prefix[1:])
        return False
    def makeATrieNodes(self, word):
        for j in self.root:
            if word[0] == j:
                rootWord = j
                break
        else:
            rootWord = TrieNode(word[0])
            self.root.append(rootWord)
            for i in word[1:]:
                nextNode = TrieNode(i)
                rootWord.addNext(nextNode)
                rootWord = nextNode
            rootWord.setBreakable(True)
            return
        # has the letter.
        word = word[1:]
        while 1:
            if not word:
                rootWord.setBreakable(True)
                break
            for i in rootWord.nextNodes:
                if i == word[0]:
                    rootWord = i
```

```
                    word = word[1:]
                    break
            else:
                for i in word:
                    nextNode = TrieNode(i)
                    rootWord.addNext(nextNode)
                    rootWord = nextNode
                rootWord.setBreakable(True)
                break
```

Tree/ValidateBinarySearchTree.py

```
"""
Given a binary tree, determine if it is a valid binary search tree (BST).
Assume a BST is defined as follows:
The left subtree of a node contains only nodes with keys less than the node's
key.
The right subtree of a node contains only nodes with keys greater than the node's
key.
Both the left and right subtrees must also be binary search trees.
Example 1:
Input:
    2
   / \
  1   3
Output: true
Example 2:
    5
   / \
  1   4
     / \
    3   6
Output: false
Explanation: The input is: [5,1,4,null,null,3,6]. The root node's value
             is 5 but its right child's value is 4.
验证是否为有效的 二叉搜索树。
二叉搜索树的定义是：
右边的小于父节点，左边的大于父节点，对于每一个节点都是同样的规则。
思路：
直接中序遍历，中序遍历的二叉搜索树会以排序好的形式返回，返回的同时判断是否比上一个要大，若小于或
相等，那么就表示这不是一颗二叉搜索树。
递归..O(n) 时间复杂度。
测试链接：
https://leetcode.com/problems/validate-binary-search-tree/description/
beat 100% 36ms.
"""
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def isValidBST(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
            return True
        self.prev = -float('inf')
        def inOrderTraversal(root):
            if root.left:
                if inOrderTraversal(root.left) == -1:
                    return -1
            if root.val <= self.prev:
```

```
                return -1
            self.prev = root.val
            if root.right:
                if inOrderTraversal(root.right) == -1:
                    return -1
        if inOrderTraversal(root) == -1:
            return False
        return True
```