目录

数组

数组的遍历

26. Remove Duplicates from Sorted Array

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        cur = -1
        val = -101
        for i in range(len(nums)):
            if nums[i] != val:
                cur += 1
                nums[cur] = nums[i]
                val = nums[i]
        return cur + 1
```

27. Remove Element

```python
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        # 如果当前元素 x 与移除元素 val 相同，那么跳过该元素。
        # 如果当前元素 x 与移除元素 val 不同，那么我们将其放
        # 到下标 idx 的位置，并让 idx 自增右移。
        idx = 0
        for i in range(len(nums)):
            if nums[i] != val:
                nums[idx] = nums[i]
                idx += 1
        return idx
```

41. First Missing Positive

```python
class Solution:
    def firstMissingPositive(self, nums: List[int]) -> int:
        # 直接把数组看做哈希表，元素1放到位置0，元素k+1放到位置k
        # 对于每一个位置如果当前是正数，且位置与正数值不对应，那么就把
        # 当前元素交换到正确的地方，如果换过来的元素还是正数且不符合
        # 那么就再次交换直到符合；如果发现当前不符合而且目标位置是
        # 已经符合的，那么就不交换

        # 为什么这个while一定会停下?
        # (1) 如果新来的是负数 -> 停下
        # (2) 如果新来的是新位置 -> 继续换
        # (3) 如果新来的是旧位置 -> nums[nums[i]-1] == nums[i]
        for i in range(len(nums)):
            while 1 <= nums[i] <= len(nums) and nums[nums[i]-1] != nums[i]:
                # 这种写法是错的，因为nums[i]会变
                # nums[i], nums[nums[i]-1] = nums[nums[i]-1], nums[i]
                nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]
        for i in range(len(nums)):
            if nums[i]-1 != i:
                return i + 1
        return len(nums) + 1
```

80. Remove Duplicates from Sorted Array II

```python
# 每个元素最多出现2次
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        val = -10001
        idx = 0
        # 填入的位置只用idx控制，同26/27
        for i in range(len(nums)):
            # 前面没有出现，新元素直接填
            if nums[i] != val:
                nums[idx] = nums[i]
                val = nums[i]
                idx += 1
            else:
                # 出现过了就检查，如果往前看两个不是
                # 说明还没到2个，所以可以填
                if idx < 2 or nums[idx-2] != val:
                    nums[idx] = val
                    idx += 1
        return idx
```

88. Merge Sorted Array

```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int):
        """
        Do not return anything, modify nums1 in-place instead.
        """
        ptr = m-1 # 表示这个位置之前的nums1是要处理的
        idx = len(nums1)-1
        while nums2:
            # 注意判断ptr是不是到头了
            if ptr < 0:
                # 已经处理完了nums1，直接把nums2拿过来
                nums1[idx] = nums2.pop(-1)
            else:
                if nums2[-1] >= nums1[ptr]:
                    nums1[idx] = nums2.pop(-1)
                else:
                    nums1[idx] = nums1[ptr]
                    ptr -= 1
            idx -= 1
```

189. Rotate Array

```
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        # 3 reverse: left + right + all

        # remember k could be larger than nums
        k %= len(nums)

        nums[:-k] = nums[:-k][::-1]
        nums[-k:] = nums[-k:][::-1]
        i = 0
        # pay attention! "<=" is wrong!
        while i < (len(nums) // 2):
            nums[i], nums[-i-1] = nums[-i-1], nums[i]
            i += 1
```

251. Flatten 2D Vector

```
class Vector2D:

    def __init__(self, vec: List[List[int]]):
        self.inner = 0
        self.outer = 0
        self.vec = vec

    def next(self) -> int:
        val = self.vec[self.outer][self.inner]
        if self.inner == len(self.vec[self.outer])-1:
            self.outer += 1
            self.inner = 0
        else:
            self.inner += 1
        return val

    def hasNext(self) -> bool:
        """
        bad case
        ["Vector2D","hasNext","next","hasNext"]
        [[[[],[3]]],[],[],[]]
        ["Vector2D", "hasNext"]
        [[[]],[]]
        """
        # 跳过空数组
        while self.outer < len(self.vec) and len(self.vec[self.outer]) == 0:
            self.outer += 1
        if self.outer < len(self.vec)-1:
            return True
        if self.outer == len(self.vec)-1:
            if self.inner < len(self.vec[-1]):
                return True
        return False
```

268. Missing Number

```
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        for i in range(len(nums)):
            while nums[i] < len(nums) and nums[i] != i:
                n = nums[i]
                temp = nums[n]
                nums[n] = n
                nums[i] = temp
        for i in range(len(nums)):
            if nums[i] != i:
                return i
        return len(nums)
"""
# bit
class Solution:
    def missingNumber(self, nums):
        missing = len(nums)
        for i, num in enumerate(nums):
            missing ^= i ^ num
        return missing
# guass sum
class Solution:
    def missingNumber(self, nums):
        expected_sum = len(nums)*(len(nums)+1)//2
        actual_sum = sum(nums)
        return expected_sum - actual_sum
"""
```

274. H-Index

```
class Solution:
    def hIndex(self, citations: List[int]) -> int:
        # O(n) solution
        # 因为『H指数』一定小于等于论文的数量n
        # 所以我们把引用量大于论文数量的放在一起
        # 准备n+1个桶（桶代表引用数），然后反向遍历
        n = len(citations)
        bucket = [0] * (n+1)
        for i in citations:
            if i >= n:
                bucket[n] += 1
            else:
                bucket[i] += 1
        h = 0
        for i in range(n, -1, -1):
            # i代表i次引用，h是h篇论文
            h += bucket[i]
            if h >= i:
                # 有h篇论文有i或i以上次引用
                # 所以至少有i篇论文有i或i次以上引用
                return i
"""
    citations.sort(reverse=True)
    h = 0
    i = 0
    # 至少有h篇引用>=h
    while i < len(citations):
        # 例如i=0，这个时候实际上是在看第一篇，所以比较的应当是h+1
        if citations[i] >= h+1:
            h += 1
            i += 1
        else:
            break
    return h
"""
```

283. Move Zeroes

```python
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        # 碰到了非零的就往前填，填的位置用指针控制
        ptr = 0
        for i in range(len(nums)):
            if nums[i] != 0:
                nums[ptr] = nums[i]
                ptr += 1
        for i in range(ptr, len(nums)):
            nums[i] = 0
```

315. Count of Smaller Numbers After Self

```
class Solution:
    def countSmaller(self, nums: List[int]) -> List[int]:

        # 为什么要index，原因是要record count，如果改变原数组则不能操作
        def merge_sort(start, end):
            if start >= end:
                return
            mid = start + (end - start) // 2
            merge_sort(start, mid)
            merge_sort(mid+1, end)

            temp = []
            i, j = start, mid+1
            while i <= mid and j <= end:
                # 这里是因为当小于的时候，j前面的元素就是比i小的，这个时候要记录
                if nums[index[i]] <= nums[index[j]]:
                    res[index[i]] += j - (mid+1)
                    temp.append(index[i])
                    i += 1
                else:
                    temp.append(index[j])
                    j += 1
            while i <= mid:
                res[index[i]] += j - (mid+1)
                temp.append(index[i])
                i += 1
            while j <= end:
                temp.append(index[j])
                j += 1

            # 排序后的index，覆盖
            index[start: end+1] = temp

        if len(nums) == 1:
            return [0]
        index = list(range(len(nums)))
        res = [0] * len(nums)
        merge_sort(0, len(nums)-1)
        return res
```

324. Wiggle Sort II

```python
class Solution:
    def wiggleSort(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        # 桶排序，然后正序逆序间隔填充
        max_num = max(nums)
        b = [0] * (max_num+1)
        for n in nums:
            b[n] += 1
        ptr = max_num
        for i in range(1, len(nums), 2):
            while b[ptr] == 0:
                ptr -= 1
            nums[i] = ptr
            b[ptr] -= 1
        for i in range(0, len(nums), 2):
            while b[ptr] == 0:
                ptr -= 1
            nums[i] = ptr
            b[ptr] -= 1
```

376. Wiggle Subsequence

```python
class Solution:
    def wiggleMaxLength(self, nums: List[int]) -> int:
        # this is a really clever solution
        # we only consider the currest val and last val
        # and to update down and up count accordingly
        down, up = 1, 1
        for i in range(1, len(nums)):
            if nums[i] > nums[i-1]:
                up = down + 1
            elif nums[i] < nums[i-1]:
                down = up + 1
        return max(down, up)
```

396. Rotate Function

```python
class Solution:
    def maxRotateFunction(self, nums: List[int]) -> int:
        # 错位相减法
        # F(k+1) = F(k) + S − n ∗ arr_k[n−1]
        s = sum(nums)
        val = sum(i*j for i,j in zip(nums, range(len(nums))))
        res = val
        for i in range(1, len(nums)):
            val += s - len(nums) * nums[-i]
            res = max(res, val)
        return res
```

414. Third Maximum Number

```python
class Solution:
    def thirdMax(self, nums: List[int]) -> int:
        nums = list(set(nums))
        if len(nums) == 3:
            return min(nums)
        elif len(nums) == 2:
            return max(nums)
        elif len(nums) == 1:
            return nums[0]
        a, b, c = tuple(sorted(nums[:3], reverse=True))
        for i in nums[3:]:
            if i > a:
                c = b
                b = a
                a = i
            elif i >= b and i < a:
                c = b
                b = i
            elif i >= c and i < b:
                c = i
        return c
"""
# use float("-inf") to unify the process
class Solution(object):
    def thirdMax(self, nums):
        v = [float('-inf'), float('-inf'), float('-inf')]
        for num in nums:
            if num not in v:
                if num > v[0]: v = [num, v[0], v[1]]
                elif num > v[1]: v = [v[0], num, v[1]]
                elif num > v[2]: v = [v[0], v[1], num]
        return max(nums) if float('-inf') in v else v[2]
"""
```

453. Minimum Moves to Equal Array Elements

```python
class Solution:
    def minMoves(self, nums: List[int]) -> int:
        # n-1个数同时加一，就好比每次有一个数自身减一
        # 因为只能做减法，所以数组最后的数只能是最小值
        min_val = min(nums)
        return sum(nums[i]-min_val for i in range(len(nums)))
```

457. Circular Array Loop

```python
class Solution:
    def circularArrayLoop(self, nums: List[int]) -> bool:
        self.BIG = 1001 # 用一个大数做标记
        # 每个节点都出发看看能到哪些节点
        n = len(nums)
        for i in range(n):
            if nums[i] >= self.BIG: # 已标记
                continue
            cur = i # 当前位置
            tag = self.BIG + i # 当前出发圈可以到达的标记tag
            while True:
                nxt = (cur + nums[cur]) % n # 下一个位置，正数和负数通用
                if nxt == cur: # k=1，只有自己
                    break
                if nums[nxt] == tag: # 已访问过
                    return True
                if nums[cur] * nums[nxt] < 0: # 保证同号
                    break
                nums[cur] = tag
                cur = nxt
        return False
```

462. Minimum Moves to Equal Array Elements II

```python
class Solution:
    def minMoves2(self, nums: List[int]) -> int:
        # MAE最小值在中位数时候取到
        if len(nums) <= 1:
            return 0
        nums.sort()
        import statistics
        median = statistics.median(nums)
        return sum(abs(nums[i] - int(median)) for i in range(len(nums)))

"""
def median(lst):
    n = len(lst)
    s = sorted(lst)
    return (s[n//2-1]/2.0+s[n//2]/2.0, s[n//2])[n % 2] if n else None
"""
```

485. Max Consecutive Ones

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        count, res = 0, 0
        for i in nums:
            if i == 0:
                res = max(count, res)
                count = 0
            else:
                count += 1
        return max(count, res)
```

493. Reverse Pairs

```
class Solution:
    def reversePairs(self, nums: List[int]) -> int:
        self.res = 0
        self.merge_sort(nums)
        return self.res

    def merge_sort(self, nums):
        if len(nums) > 1:

            left_arr = nums[:len(nums)//2]
            right_arr = nums[len(nums)//2:]

            self.merge_sort(left_arr)
            self.merge_sort(right_arr)

            i, j = 0, 0
            while i < len(left_arr) and j < len(right_arr):
                if left_arr[i] > 2 * right_arr[j]:
                    self.res += len(left_arr) - i
                    j += 1
                else:
                    i += 1

            i, j, k = 0, 0, 0
            while i < len(left_arr) and j < len(right_arr):
                if left_arr[i] > right_arr[j]:
                    nums[k] = right_arr[j]
                    j += 1
                    k += 1
                else:
                    nums[k] = left_arr[i]
                    i += 1
                    k += 1

            while i < len(left_arr):
                nums[k] = left_arr[i]
                k += 1
                i += 1
            while j < len(right_arr):
                nums[k] = right_arr[j]
                k += 1
                j += 1
```

495. Teemo Attacking

```
class Solution:
    def findPoisonedDuration(self, timeSeries: List[int], duration: int):

        # Please use this version!!! Extremely clear!

        a = len(timeSeries)
        n = duration
        for i in range(1 , a):
            b = timeSeries[i] - timeSeries[i-1]
            if b < duration:
                n += b
            else:
                n += duration
        return n
```

628. Maximum Product of Three Numbers

```python
class Solution:
    def maximumProduct(self, nums: List[int]) -> int:
        # 只要求出数组中最大的三个数以及最小的两个数
        min2, min1 = float("inf"), float("inf")
        max1, max2, max3 = float("-inf"), float("-inf"), float("-inf")
        for i in nums:
            # 学习写法!
            if i < min2:
                min1 = min2
                min2 = i
            elif i < min1:
                min1 = i

            if i > max3:
                max1 = max2
                max2 = max3
                max3 = i
            elif i > max2:
                max1 = max2
                max2 = i
            elif i > max1:
                max1 = i
        return max(min2 * min1 * max3, max1 * max2 * max3)
```

665. Non-decreasing Array

```python
class Solution:
    def checkPossibility(self, nums: List[int]) -> bool:
        if len(nums) == 1:
            return True
        count = 0
        """
        # 3, 4, 2, 5 is a bad case
        for i in range(1, len(nums)):
            if nums[i] < nums[i-1]:
                count += 1
        """
        for i in range(1, len(nums)):
            if nums[i] < nums[i-1]:
                count += 1
                if i == 1 or nums[i] >= nums[i-2]:
                    # 2 5 (3) -> 2 3 3
                    # 尽量改前一个
                    nums[i-1] = nums[i]
                else:
                    # 迫不得已，改当前的
                    nums[i] = nums[i-1]
        return count <= 1
```

674. Longest Continuous Increasing Subsequence

```python
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:
        max_count = 0
        count = 0
        last = float("-inf")
        for i in nums:
            if i > last:
                count += 1
            else:
                max_count = max(count, max_count)
                count = 1
            last = i
        return max(count ,max_count)
```

926. Flip String to Monotone Increasing

```python
class Solution:
    def minFlipsMonoIncr(self, s: str) -> int:
        # 决定的是在原字符串中选择哪一个位置的1，使其作为最优解中的第一个1
        # 只要一直记录着当前位置的前面有多少个1，后面有多少个0即可
        cnt_0 = s.count("0") # 最前面的位置，后面有所有的0
        cnt_1 = 0 # 最前面的位置，前面还没有1
        res = len(s) - cnt_0 # 最恶劣的情况，把所有1给反过来
        for c in s:
            if c == "0":
                cnt_0 -= 1
            else:
                # 把当前的1看做最优解的第一个1
                res = min(res, cnt_0 + cnt_1)
                cnt_1 += 1
        return res
```

1570. Dot Product of Two Sparse Vectors

```python
class SparseVector:
    def __init__(self, nums: List[int]):
        self.d = dict()
        for i, n in enumerate(nums):
            self.d[i] = n

    # Return the dotProduct of two sparse vectors
    def dotProduct(self, vec: 'SparseVector') -> int:
        # 如果只有一个稀疏，可以比较非零元素个数，用稀疏的非零元素来乘
        idx = set(self.d).intersection(set(vec.d))
        return sum(self.d[i] * vec.d[i] for i in idx)
```

1762. Buildings With an Ocean View

```
class Solution:
    def findBuildings(self, heights: List[int]) -> List[int]:
        max_h = 0
        L = []
        for i in range(len(heights)-1, -1, -1):
            if heights[i] > max_h:
                L.append(i)
                max_h = heights[i]
        return L[::-1]
```

1775. Equal Sum Arrays With Minimum Number of Operations

```python
class Solution:
    def minOperations(self, nums1: List[int], nums2: List[int]) -> int:
        # 把每一个位置可能的贡献值算出来，然后排序，贪心进行贡献
        s1, s2 = sum(nums1), sum(nums2)
        d = abs(s1 - s2)
        if d == 0:
            return 0
        if s1 > s2:
            nums1, nums2 = nums2, nums1
        contrib1 = [6-i for i in nums1]
        contrib2 = [i-1 for i in nums2] # 最小要保证1
        ctb = contrib1 + contrib2
        ctb.sort(reverse=True)
        count = 0
        for c in ctb:
            count += 1
            d -= c
            if d <= 0:
                return count
        return -1
```

1822. Sign of the Product of an Array

```
class Solution:
    def arraySign(self, nums: List[int]) -> int:
        neg = 0
        for i in nums:
            if i < 0:
                neg += 1
            elif i == 0:
                return 0
        return 1 if neg % 2 == 0 else -1
```

二维数组

36. Valid Sudoku

```python
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        for i in range(9):
            s = "".join(board[i]).replace(".", "")
            if len(s) != len(set(s)):
                return False
        for j in range(9):
            s = "".join([board[i][j] for i in range(9)]).replace(".", "")
            if len(s) != len(set(s)):
                return False
        for i in range(3):
            for j in range(3):
                s = "".join([
                    board[3*i+m][3*j+n]
                    for m in range(3)
                    for n in range(3)
                ]).replace(".", "")
                if len(s) != len(set(s)):
                    return False
        return True
```

48. Rotate Image

```python
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix)
        for i in range(n//2):
            for j in range(i, i+n-2*i-1):
                (
                    matrix[i][j],
                    matrix[j][n-i-1],
                    matrix[n-i-1][n-j-1],
                    matrix[n-j-1][i],
                ) = (
                    matrix[n-j-1][i],
                    matrix[i][j],
                    matrix[j][n-i-1],
                    matrix[n-i-1][n-j-1],
                )
```

54. Spiral Matrix

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        res = []
        up, right, down, left = 0, len(matrix[0])-1, len(matrix)-1, 0
        while True:
            # 关键是想清楚哪里add和哪里break
            for i in range(left, right+1):
                res.append(matrix[up][i])
            up += 1
            if up > down:
                break
            for i in range(up, down+1):
                res.append(matrix[i][right])
            right -= 1
            if left > right:
                break
            for i in range(right, left-1, -1):
                res.append(matrix[down][i])
            down -= 1
            if up > down:
                break
            for i in range(down, up-1, -1):
                res.append(matrix[i][left])
            left += 1
            if left > right:
                break
        return res
```

59. Spiral Matrix II

```python
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        matrix = [[0] * n for _ in range(n)]
        num = 1
        up, right, down, left = 0, n-1, n-1, 0
        while True:
            # 同54
            for i in range(left, right+1):
                matrix[up][i] = num
                num += 1
            up += 1
            if up > down:
                break
            for i in range(up, down+1):
                matrix[i][right] = num
                num += 1
            right -= 1
            if left > right:
                break
            for i in range(right, left-1, -1):
                matrix[down][i] = num
                num += 1
            down -= 1
            if up > down:
                break
            for i in range(down, up-1, -1):
                matrix[i][left] = num
                num += 1
            left += 1
            if left > right:
                break
        return matrix
```

73. Set Matrix Zeroes

```python
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        r, c = len(matrix), len(matrix[0])
        # 借用第一行和第一列做整行或整列是否有0的标记
        # 但事先要判断本身第一行或第一列是否有0
        ROW_FLAG, COL_FLAG = False, False
        for i in range(r):
            if matrix[i][0] == 0:
                ROW_FLAG = True
                break
        for j in range(c):
            if matrix[0][j] == 0:
                COL_FLAG = True
                break

        # 标记
        for i in range(1, r):
            if not all(matrix[i]):
                matrix[i][0] = 0
        for j in range(1, c):
            if not all([matrix[i][j] for i in range(r)]):
                matrix[0][j] = 0

        # 根据标记归零
        for i in range(1, r):
            if matrix[i][0] == 0:
                for j in range(c):
                    matrix[i][j] = 0
        for j in range(1, c):
            if matrix[0][j] == 0:
                for i in range(r):
                    matrix[i][j] = 0

        # 处理第一行和第一列
        if ROW_FLAG:
            for i in range(r):
                matrix[i][0] = 0
        if COL_FLAG:
            for j in range(c):
                matrix[0][j] = 0
```

118. Pascal's Triangle

```
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        res = []
        cur = [1]
        while numRows:
            res.append(cur.copy())
            if len(cur) == 1:
                cur = [1, 1]
            else:
                cur = [1]
                      + [cur[i]+cur[i+1] for i in range(len(cur)-1)]
                      + [1]
            numRows -= 1
        return res
```

119. Pascal's Triangle II

```
class Solution:
    def getRow(self, rowIndex: int) -> List[int]:
        cur = [1]
        while rowIndex:
            if len(cur) == 1:
                cur = [1, 1]
            else:
                cur = [1]
                      + [cur[i]+cur[i+1] for i in range(len(cur)-1)]
                      + [1]
            rowIndex -= 1
        return cur
```

289. Game of Life

```
class Solution:
    def gameOfLife(self, board: List[List[int]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        # 2表示原来是死的，但是现在活了
        # 3表示原来是活的，但是现在死了
        r, c = len(board), len(board[0])
        for i in range(r):
            for j in range(c):
                count = 0
                for m in range(-1,2):
                    for n in range(-1,2):
                        if (m==0 and n==0) or i+m<0 or i+m>r-1 or j+n<0 or j+n>
                            continue
                        count += board[i+m][j+n] == 1 or board[i+m][j+n] == 3
                if board[i][j] == 0:
                    if count == 3:
                        board[i][j] = 2
                else:
                    if count > 3 or count < 2:
                        board[i][j] = 3
        for i in range(r):
            for j in range(c):
                if board[i][j] == 2:
                    board[i][j] = 1
                elif board[i][j] == 3:
                    board[i][j] = 0
        return board
```

348. Design Tic-Tac-Toe

```
class TicTacToe:
    def __init__(self, n: int):
        self.n = n
        self.rows = [0] * n
        self.cols = [0] * n
        self.diag = 0
        self.anti_diag = 0
    def move(self, row: int, col: int, player: int) -> int:
        i = 1 if player == 1 else -1
        self.rows[row] += i
        self.cols[col] += i
        self.diag += (row == col) * i
        self.anti_diag += (row + col == self.n-1) * i
        if (
            abs(self.rows[row]) == self.n
            or abs(self.cols[col]) == self.n
            or abs(self.diag) == self.n
            or abs(self.anti_diag) == self.n
        ):
            return player
        else:
            return 0
```

498. Diagonal Traverse

```
class Solution:
    def findDiagonalOrder(self, mat: List[List[int]]) -> List[int]:
        # 注意一共有m+n−1次遍历
        # 边界条件一个max一个min，考虑细致
        m, n = len(mat), len(mat[0])
        res = []
        for i in range(m+n-1):
            if i%2 == 0:
                for k in range(min(m-1, i), max(0, i-(n-1))-1, -1):
                    res.append(mat[k][i-k])
            else:
                for k in range(max(0, i-(n-1)), min(m-1, i)+1):
                    res.append(mat[k][i-k])
        return res
```

566. Reshape the Matrix

```
class Solution:
    def matrixReshape(self, mat: List[List[int]], r: int, c: int) -> List[Lis
        res = []
        R, C = len(mat), len(mat[0])
        if R * C != r * c:
            return mat
        for i in range(R):
            for j in range(C):
                if (j + i*C) % c == 0:
                    res.append([mat[i][j]])
                else:
                    res[-1].append(mat[i][j])
        return res
```

598. Range Addition II

```python
class Solution:
    def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:
        r, c = m, n
        for i in range(len(ops)):
            r = min(r, ops[i][0])
            c = min(c, ops[i][1])
        return r * c
```

766. Toeplitz Matrix

```python
class Solution:
    def isToeplitzMatrix(self, matrix: List[List[int]]) -> bool:
        # 一个按行，一个按列，两个分别都要算对角线坐标，用while控制越界
        m, n = len(matrix), len(matrix[0])
        for i in range(m):
            cur_i, cur_j = i, 0
            val = matrix[cur_i][cur_j]
            while cur_i >= 0 and cur_i < m and cur_j >= 0 and cur_j < n:
                if matrix[cur_i][cur_j] != val:
                    return False
                cur_i += 1
                cur_j += 1
        for j in range(n):
            cur_i, cur_j = 0, j
            val = matrix[cur_i][cur_j]
            while cur_i >= 0 and cur_i < m and cur_j >= 0 and cur_j < n:
                if matrix[cur_i][cur_j] != val:
                    return False
                cur_i += 1
                cur_j += 1
        return True
```

1267. Count Servers that Communicate

```
class Solution:
    def countServers(self, grid: List[List[int]]) -> int:
        m, n = len(grid), len(grid[0])
        row, col = [0] * m, [0] * n
        for i in range(m):
            for j in range(n):
                if grid[i][j]:
                    row[i] += 1
                    col[j] += 1
        res = 0
        for i in range(m):
            for j in range(n):
                if grid[i][j] == 1 and (row[i] >= 2 or col[j] >= 2):
                    res += 1
        return res
```

1275. Find Winner on a Tic Tac Toe Game

```python
class Solution:
    def tictactoe(self, moves: List[List[int]]) -> str:
        player = True # A plays first
        row = [0, 0, 0]
        col = [0, 0, 0]
        diag = 0
        anti_diag = 0
        count = 0

        for x, y in moves:
            row[x] += 1 if player else -1
            col[y] += 1 if player else -1
            if x == y:
                diag += 1 if player else -1
            if x + y == 2:
                anti_diag += 1 if player else -1
            if player:
                player = False
            else:
                player = True
            count += 1

        if any(i == 3 for i in row)
            or any(i == 3 for i in col)
             or diag == 3 or anti_diag == 3:
            return "A"
        if any(i == -3 for i in row)
            or any(i == -3 for i in col)
             or diag == -3 or anti_diag == -3:
            return "B"
        if count < 9:
            return "Pending"
        else:
            return "Draw"
```

1861. Rotating the Box

```python
class Solution:
    def rotateTheBox(self, box: List[List[str]]) -> List[List[str]]:
        r, c = len(box), len(box[0])
        res = [["."]*r for i in range(c)]

        for i in range(r-1, -1, -1):
            for idx, item in enumerate(self.arrange(box[i])):
                res[idx][r-1-i] = item

        return res

    def arrange(self, arr):
        L = ["."] * len(arr)
        count = 0
        for i in range(len(L)):
            if arr[i] == "#":
                count += 1
            elif arr[i] == "*":
                L[i] = "*"
                # pos是相对于当前*的向前距离
                pos = -1
                while count > 0:
                    L[i+pos] = "#"
                    pos -= 1
                    count -= 1
        if count > 0:
            pos = -1
            while count > 0:
                L[pos] = "#"
                pos -= 1
                count -= 1
        return L
```

2128. Remove All Ones With Row and Column Flips

```python
class Solution:
    def removeOnes(self, grid: List[List[int]]) -> bool:
        # 找规律，每一行之间要么相等，要么完全相反
        forward = grid[0]
        backward = [1-i for i in grid[0]]
        for row in grid:
            if row != forward and row != backward:
                return False
        return True
```

2221. Find Triangular Sum of an Array

```python
class Solution:
    def triangularSum(self, nums: List[int]) -> int:
        coef = 1
        sum = nums[0]
        n = len(nums) - 1
        for k in range(1, len(nums)):
            # 利用组合数的递推公式
            coef *= (n - k + 1)
            coef //= k # 组合数一定是整数，避免浮点
            sum += coef * nums[k] # 二项式展开
            sum %= 10
        return sum

"""
# 模拟
class Solution:
    def triangularSum(self, nums: List[int]) -> int:
        while len(nums) > 1:
            new_nums = list()
            for i in range(len(nums) - 1):
                new_nums.append((nums[i] + nums[i + 1]) % 10)
            nums = new_nums
        return nums[0]
"""
```

In the above example, this representation would evaluate to:

$$s = \sum_{k=0}^{5} \binom{4}{k} a_k = \binom{4}{0}(1) + \binom{4}{1}(2) + \binom{4}{2}(3) + \binom{4}{3}(4) + \binom{4}{4}(5)$$
$$s = (1)(1) + (4)(2) + (6)(3) + (4)(4) + (1)(5) = 48$$

Now we want a way to calculate $\binom{n}{k}$. From the binomial theorem, we have the equality

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

However, we don't want to keep calculating factorials for every `k`, so it would be nice to have a relationship between $\binom{n}{k}$ and $\binom{n}{k-1}$. manipulation:

$$\binom{n}{k} = \binom{n}{k-1} x$$

$$\frac{n!}{k!(n-k)!} = \frac{n!}{(k-1)!(n-k+1)!} x$$

$$x = \frac{(k-1)!}{k!} \cdot \frac{(n-k+1)!}{(n-k)!}$$

$$x = \frac{(k-1)!}{k(k-1)!} \cdot \frac{(n-k+1)(n-k)!}{(n-k)!}$$

$$x = \frac{n-k+1}{k}$$

$$\binom{n}{k} = \binom{n}{k-1} \frac{n-k+1}{k}$$

Now, we can calculate $\binom{n}{k}$ *as we iterate* through the list as an O(1) step.

前缀和

238. Product of Array Except Self

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        n = len(nums)
        pre = [1] * len(nums)
        suf = [1] * len(nums)
        for i in range(len(nums)):
            pre[i] = nums[i] * (pre[i-1] if i-1 >= 0 else 1)
            suf[n-i-1] = nums[n-i-1] * (suf[n-i] if n-i < len(nums) else 1)
        res = [1] * len(nums)
        for i in range(n):
            if i == 0:
                res[i] = suf[1]
            elif i == len(nums)-1:
                res[i] = pre[-2]
            else:
                res[i] = pre[i-1] * suf[i+1]
        return res
```

303. Range Sum Query - Immutable

```
class NumArray:

    def __init__(self, nums: List[int]):
        self.arr = list(accumulate(nums))

    def sumRange(self, left: int, right: int) -> int:
        return self.arr[right] - (0 if left == 0 else self.arr[left-1])


# Your NumArray object will be instantiated and called as such:
# obj = NumArray(nums)
# param_1 = obj.sumRange(left,right)
```

304. Range Sum Query 2D - Immutable

```
class NumMatrix:

    def __init__(self, matrix: List[List[int]]):
        self.m = matrix
        self.r = len(matrix)
        self.c = len(matrix[0])
        self.pf = [[0] * self.c for _ in range(self.r)]
        for i in range(self.r):
            for j in range(self.c):
                if i == 0:
                    self.pf[i][j] = self.m[i][j]
                        + (0 if j == 0 else self.pf[i][j-1])
                elif j == 0:
                    self.pf[i][j] = self.m[i][j] + self.pf[i-1][j]
                else:
                    self.pf[i][j] = self.m[i][j]
                        + self.pf[i-1][j]
                        + self.pf[i][j-1]
                        - self.pf[i-1][j-1]

    def sumRegion(self, row1: int, col1: int, row2: int, col2: int):
        left_up = self.pf[row1-1][col1-1]
                  if row1-1 >= 0 and col1-1 >= 0 else 0
        left_down = self.pf[row2][col1-1] if col1-1 >= 0 else 0
        right_up = self.pf[row1-1][col2] if row1-1 >= 0 else 0
        right_down = self.pf[row2][col2]
        return right_down - left_down - right_up + left_up



# Your NumMatrix object will be instantiated and called as such:
# obj = NumMatrix(matrix)
# param_1 = obj.sumRegion(row1,col1,row2,col2)
```

363. Max Sum of Rectangle No Larger Than K

```python
from sortedcontainers import SortedList
class Solution:
    def maxSumSubmatrix(self, matrix: List[List[int]], k: int) -> int:
        m, n = len(matrix), len(matrix[0])
        res = float("-inf")
        # 固定上下两个边界
        for high in range(m):
            arr = [0] * n
            for low in range(high, m):
                # 把这个矩阵拍扁
                for i in range(n):
                    arr[i] += matrix[low][i]
                s, pre = SortedList([0]), 0 # 前缀和
                # 初始化sortedlist包含0，因为需要sum(arr[:i])没有左边界的情况
                for i in range(n):
                    pre += arr[i]
                    # right - left <= k -> left >= right - k
                    # left越小越好
                    left_idx = s.bisect_left(pre - k)
                    if left_idx < len(s): # 存在这样的元素
                        val = pre - s[left_idx] # 利用前缀和计算左右范围
                        if val == k:
                            return k
                        if val > res:
                            res = val
                    s.add(pre) # 不是append，这个加进去自动排序的
        return res
```

523. Continuous Subarray Sum

```python
class Solution:
    def checkSubarraySum(self, nums: List[int], k: int) -> bool:
        # 思路：把前缀和%k逐个放入哈希表
        # 看看有没有重复的mod结果（为了使得长度>2，所以还要存index)
        # bad case: [23,2,4,6,6], k=7
        # 这个时候因为可能正好从开始一直加到某一个数字被k整除
        # 但是之前0没有出现，因此需要初始化补上(0, −1)
        mod_to_idx = dict(zip([0], [-1]))
        presum = 0
        for i, n in enumerate(nums):
            presum += n
            if presum % k in mod_to_idx:
                if i - mod_to_idx[presum % k] >= 2:
                    return True
            else:
                mod_to_idx[presum % k] = i
        return False
```

560. Subarray Sum Equals K

```python
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        # 使用前缀和，对于每一个元素x
        # 我们只需要查询k−x是不是在里面就可以了
        # 每次加入的新元素是累计和
        hash_cumsum = defaultdict(int)
        hash_cumsum[0] = 1 # element: freq
        count = 0
        cumsum = 0
        for x in nums:
            cumsum += x
            if cumsum-k in hash_cumsum:
                count += hash_cumsum[cumsum-k] # [1, −1, 0] bad case
            hash_cumsum[cumsum] += 1
        return count
```

1292. Maximum Side Length of a Square with Sum Less than or Equal to Threshold

```
class Solution:
    def maxSideLength(self, mat: List[List[int]], threshold: int):
        self.m = mat
        self.r = len(mat)
        self.c = len(mat[0])
        self.t = threshold
        self.pf = [[0] * self.c for _ in range(self.r)]
        for i in range(self.r):
            for j in range(self.c):
                if i == 0:
                    self.pf[i][j] = self.m[i][j]
                        + (0 if j == 0 else self.pf[i][j-1])
                elif j == 0:
                    self.pf[i][j] = self.m[i][j] + self.pf[i-1][j]
                else:
                    self.pf[i][j] = self.m[i][j]
                        + self.pf[i-1][j]
                        + self.pf[i][j-1]
                        - self.pf[i-1][j-1]

        self.res = 0
        for i in range(self.r):
            for j in range(self.c):
                self.res = max(self.search(i, j), self.res)
        return self.res

    def sumRegion(self, row1: int, col1: int, row2: int, col2: int):
        left_up = self.pf[row1-1][col1-1]
                  if row1-1 >= 0 and col1-1 >= 0 else 0
        left_down = self.pf[row2][col1-1] if col1-1 >= 0 else 0
        right_up = self.pf[row1-1][col2] if row1-1 >= 0 else 0
        right_down = self.pf[row2][col2]
        return right_down - left_down - right_up + left_up

    def search(self, i, j):
        left, right = -min(i, j), 0
        while left < right:
            mid = left + (right - left) // 2
            if self.sumRegion(i+mid, j+mid, i, j) > self.t:
                left = mid + 1
            else:
                right = mid
        if self.sumRegion(i+left, j+left, i, j) <= self.t:
            return -left + 1
        else:
            return 0
```

1314. Matrix Block Sum

```
class Solution:
    def matrixBlockSum(self, mat: List[List[int]], k: int):
        row, col = len(mat), len(mat[0])
        pf_sum = [[0] * col for _ in range(row)]
        # for every i0, j0 in pf_sum
        # it means sum_{x=0}^{x0} sum_{y=0}^{y0} mat[i][j]
        for i in range(row):
            for j in range(col):
                if i == 0 and j == 0:
                    pf_sum[i][j] = mat[i][j]
                elif i == 0 and j > 0:
                    pf_sum[i][j] = mat[i][j] + pf_sum[i][j-1]
                elif j == 0:
                    pf_sum[i][j] = mat[i][j] + pf_sum[i-1][j]
                else:
                    pf_sum[i][j] = mat[i][j]
                        + pf_sum[i-1][j]
                        + pf_sum[i][j-1]
                        - pf_sum[i-1][j-1]
        res = [[0] * col for _ in range(row)]
        for i in range(row):
            for j in range(col):
                # when too small, treat it as zero
                # when too big, use border to wrap
                right_down = pf_sum[min(i+k, row-1)][min(j+k, col-1)]
                right_up = pf_sum[i-k-1][min(j+k, col-1)]
                    if i-k-1>=0 else 0
                left_down = pf_sum[min(i+k, row-1)][j-k-1]
                    if j-k-1>=0 else 0
                left_up = pf_sum[i-k-1][j-k-1]
                    if i-k-1>=0 and j-k-1>=0 else 0
                res[i][j] = right_down - right_up - left_down + left_up
        return res
```

1423. Maximum Points You Can Obtain from Cards

```python
class Solution:
    def maxScore(self, cardPoints: List[int], k: int) -> int:
        # 前缀和和后缀和，前面最多k个元素
        # 因为有可能全不选，所以pre数组为k+1长度
        pre, suf = [0] * (k+1), [0] * (k+1)
        for i, n in enumerate(cardPoints[:k]):
            pre[i+1] = pre[i] + n
        for i, n in enumerate(cardPoints[-k:][::-1]):
            suf[-i-2] = suf[-i-1] + n
        res = 0
        # 1 2 3 4 5 6 7 k=3
        # 0 1 3 6 -> pre
        # 18 13 7 0 -> suf
        for i in range(k+1):
            val = pre[i] + suf[i]
            if val > res:
                res = val
        return res
```

2055. Plates Between Candles

```python
class Solution:
    def platesBetweenCandles(self, s: str, queries) -> List[int]:
        # 找到每个位置左右两边的蜡烛，然后搜索时候用前缀和
        left = []
        idx = -1
        for i, c in enumerate(s):
            if c == "|":
                idx = i
            left.append(idx)
        right = []
        idx = -1
        for i in range(len(s)-1, -1, -1):
            c = s[i]
            if c == "|":
                idx = i
            right.append(idx)
        right = right[::-1]

        presum = [i=="*" for i in s]
        presum = list(accumulate(presum, initial=0))

        res = []
        for start, end in queries:
            start, end = right[start], left[end]
            if start == -1 or end == -1 or start > end:
                res.append(0)
            else:
                res.append(presum[end+1] - presum[start])
        return res
```

字符串

遍历与统计

14. Longest Common Prefix

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        i = 0
        while True:
            if i == len(strs[0]):
                return strs[0]
            cur = strs[0][i]
            for w in strs:
                # 索引前永远先判断是不是有效
                if i == len(w):
                    return w
                if w[i] != cur:
                    return strs[0][:i]
            i += 1
```

49. Group Anagrams

```python
class Solution:

    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        # 奇葩思路：用质数表示26个字母，把字符串的各个字母相乘，这样可保证字母异
        # 正常思路：把字符串用唯一的字符编码表示，然后扔到哈希表里聚合

        self.d1 = dict(zip(list("abcdefghijklmnopqrstuvwxyz"), range(26)))
        self.d2 = list("abcdefghijklmnopqrstuvwxyz")
        res = dict()
        for s in strs:
            gen = self.generator(s)
            if gen not in res:
                res[gen] = [s]
            else:
                res[gen].append(s)
        return list(res.values())

    def generator(self, s):
        res = ""
        note = [0] * 26
        for i in s:
            note[self.d1[i]] += 1
        for i in range(26):
            if note[i] != 0:
                res += self.d2[i] + str(note[i])
        return res
```

58. Length of Last Word

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        # a better solution
        end = len(s)-1
        while end >= 0 and s[end] == " ":
            end -= 1
        start = end
        while start >= 0 and s[start] != " ":
            start -= 1
        return end - start
```

125. Valid Palindrome

```
# java: s = s.toLowerCase().replaceAll("[^a-z0-9]", "");
class Solution:
    def isPalindrome(self, s: str) -> bool:
        left, right = 0, len(s)-1
        while left < right:
            while left < right and (not s[left].isdigit() and not s[left].isal
                left += 1
            while left < right and (not s[right].isdigit() and not s[right].is
                right -= 1
            if s[left].lower() != s[right].lower():
                return False
            left += 1
            right -= 1
        return True
```

151. Reverse Words in a String

```
class Solution:
    def reverseWords(self, s: str) -> str:
        return " ".join(s.split()[::-1])
"""
from collections import deque
class Solution:
    def reverseWords(self, s: str) -> str:
        left, right = 0, len(s) - 1
        # remove leading spaces
        while left <= right and s[left] == ' ':
            left += 1

        # remove trailing spaces
        while left <= right and s[right] == ' ':
            right -= 1

        d, word = deque(), []
        # push word by word in front of deque
        while left <= right:
            if s[left] == ' ' and word:
                d.appendleft(''.join(word))
                word = []
            elif s[left] != ' ':
                word.append(s[left])
            left += 1
        d.appendleft(''.join(word))

        return ' '.join(d)
"""
```

186. Reverse Words in a String II

```
class Solution:
    def reverseWords(self, s: List[str]) -> None:
        """
        Do not return anything, modify s in-place instead.
        """
        # 先把整个字符串反过来，然后逐个反转
        def reverse(start, end):
            while start < end:
                tmp = s[start]
                s[start] = s[end]
                s[end] = tmp
                start += 1
                end -= 1
        reverse(0, len(s)-1)
        start = 0
        for i in range(len(s)+1):
            if i == len(s) or s[i] == " ":
                reverse(start, i-1)
                start = i+1
```

242. Valid Anagram

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        h = defaultdict(int)
        for i in s:
            h[i] += 1
        for i in t:
            h[i] -= 1
        return all(h[i]==0 for i in h)
```

249. Group Shifted Strings

```
class Solution:
    def groupStrings(self, strings: List[str]) -> List[List[str]]:
        res = defaultdict(list)
        for s in strings:
            n = len(s)
            enc = ""
            for i in range(n):
                enc += str((ord(s[i])-ord(s[0]))%26) # 和首位的差
            pair = (n, enc)
            res[pair].append(s)
        return list(res.values())
```

344. Reverse String

```
class Solution:
    def reverseString(self, s: List[str]) -> None:
        """
        Do not return anything, modify s in-place instead.
        """
        s[:] = s[::-1]
"""
class Solution:
    def reverseString(self, s):
        left, right = 0, len(s) - 1
        while left < right:
            s[left], s[right] = s[right], s[left]
            left, right = left + 1, right - 1
"""
```

383. Ransom Note

```
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        h = defaultdict(int)
        for i in magazine:
            h[i] += 1
        for i in ransomNote:
            h[i] -= 1
        return all(h[i]>=0 for i in h)
```

387. First Unique Character in a String

```
class Solution:
    def firstUniqChar(self, s: str) -> int:
        d = defaultdict(int)
        for i in s:
            d[i] += 1
        for idx, i in enumerate(s):
            if d[i] == 1:
                return idx
        return -1
```

392. Is Subsequence

```python
class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        cur = 0
        for c in t:
            # if we don't judge, s[0] could raise
            if len(s) == 0:
                return True
            if s[0] == c:
                s = s[1:]
        return len(s) == 0
```

395. Longest Substring with At Least K Repeating Characters

```python
class Solution:
    def longestSubstring(self, s: str, k: int) -> int:
        # 这题非常奇葩，滑动窗口很难做
        # 思路是，对于当前字符串如果有一个字符的数量小于k
        # 那么答案一定不包括它自己，所以搜索左边和右边的字符串，取最大结果
        if len(s) < k:
            return 0
        counter = [0] * 26
        for c in s:
            counter[ord(c) - ord("a")] += 1
        for i, c in enumerate(s):
            if counter[ord(c) - ord("a")] < k:
                res1 = self.longestSubstring(s[:i], k)
                res2 = self.longestSubstring(s[i+1:], k)
                return max(res1, res2)
        return len(s)
```

696. Count Binary Substrings

```
class Solution:
    def countBinarySubstrings(self, s: str) -> int:
        # 先统计连续的0和1分别有多少个，如：111100011000，得到4323
        # 在4323中的任意相邻两个数字，取小的一个加起来，就是3+2+2 = 7.
        # 下面用列表存，实际上只需要一个简单变量，空间复杂度可以优化
        L = []
        n = s[0]
        count = 1
        for c in s[1:]:
            if c == n:
                count += 1
            else:
                L.append(count)
                n = c
                count = 1
        L.append(count)
        if len(L) == 1:
            return 0
        cur = L[0]
        res = 0
        for i in L[1:]:
            res += min(i, cur)
            cur = i
        return res
```

937. Reorder Data in Log Files

```
class Solution:
    def reorderLogFiles(self, logs: List[str]) -> List[str]:

        def get_key(log):
            _id, rest = log.split(" ", maxsplit=1)
            # timsort is stable
            return (0, rest, _id) if rest[0].isalpha() else (1, )

        return sorted(logs, key=get_key)
```

1055. Shortest Way to Form String

```python
class Solution:
    def shortestWay(self, source: str, target: str) -> int:
        # 思路：source中的每个字母分配一个链表
        # "abac" -> # 元素是索引
        # "a": [0, 2, -1]
        # "b": [1, -1]
        # "c": [3, -1]
        # 再用src_ptr来指示当前每个字母下一个时候的位置
        # 对于target每一个字母去找，如果找到位置的索引为-1
        # 说明这个时候source要从头遍历
        src_ptr = dict()
        src_pos_list = defaultdict(list)
        for i, ch in enumerate(source):
            src_ptr[ch] = 0
            src_pos_list[ch].append(i)
        for ch in src_pos_list:
            src_pos_list[ch].append(-1)

        tgt_i = 0
        src_last_pos = -1
        src_used = set()
        res = 0

        while tgt_i < len(target):
            ch = target[tgt_i]
            if ch not in src_pos_list:
                return -1
            src_pos = src_pos_list[ch][src_ptr[ch]]
            if src_pos != -1 and src_pos < src_last_pos:
                src_ptr[ch] += 1
                continue
            if src_pos == -1:
                res += 1
                for src_c in src_used:
                    src_ptr[src_c] = 0
                src_used = set([ch])
                src_ptr[ch] = 1
                src_last_pos = src_pos_list[ch][0]
            else:
                src_used.add(ch)
                src_last_pos = src_pos
                src_ptr[ch] += 1
            tgt_i += 1

        return res + 1
```

1864. Minimum Number of Swaps to Make the Binary String Alternating

```python
class Solution:
    def minSwaps(self, s: str) -> int:
        # 1. 遍历计算字符串中0和1的个数num0,num1,奇数位上1的个数odd1.
        # 2. 如果num0与num1的差值的大于1,不满足条件,直接返回−1
        # 3. 如果字符串长度是偶数,就会有两种排序方式"0101..." "1010..."
        # 假如是0开头,奇数位上就应该都是0,我们只需要返回奇数位上的1的个数;
        # 假如是1开头,奇数位上就应该都是1,我们只需要返回奇数位上的0的个数,
        # 所以就只需要返回odd1和n/2 − odd1的最小值,就是代表哪种交换方式代价最小.
        # 4. 如果字符串长度是奇数,就只有一种排序方式"1010..."或"0101...",
        # 如果num0比num1大的话,就应该是以0开头,否则就应该是1开头,
        # 假如是0开头,奇数位上就应该都是0,我们只需要返回奇数位上的1的个数;
        # 假如是1开头,奇数位上就应该都是1,我们只需要返回奇数位上的0的个数.

        cnt_0, cnt_1 = s.count("0"), s.count("1")
        if abs(cnt_0 - cnt_1) > 1:
            return -1
        if len(s) % 2 == 0:
            odd_1 = s[::2].count("1")
            return min(odd_1, len(s) // 2 - odd_1) # 奇数1个数和奇数0个数
        else:
            if cnt_0 > cnt_1:
                return s[::2].count("1")
            else:
                return s[::2].count("0")
```

2135. Count Words Obtained After Adding a Letter

```
class Solution:
    def wordCount(self, startWords: List[str], targetWords) -> int:
        # 每个单词都字母排序，target的abc(d)ef
        # 可以对应start的abcef
        start_words_holder = set()
        for w in startWords:
            start_words_holder.add("".join(sorted(w)))
        answer = 0
        for w in targetWords:
            w = "".join(sorted(w))
            for i in range(len(w)):
                if w[:i] + w[i + 1:] in start_words_holder:
                    answer += 1
                    break
        return answer
"""
#bitmask的做法
class Solution:
    def wordCount(self, startWords: List[str], targetWords) -> int:
        seen = set()
        for word in startWords:
            m = 0
            for ch in word:
                m ^= 1 << ord(ch)-97
            seen.add(m)

        ans = 0
        for word in targetWords:
            m = 0
            for ch in word:
                m ^= 1 << ord(ch)-97
            for ch in word:
                # 只要有一位不同即可
                if m ^ (1 << ord(ch)-97) in seen:
                    ans += 1
                    break
        return ans
"""
```

2222. Number of Ways to Select Buildings

```python
class Solution:
    def numberOfWays(self, s: str) -> int:
        # 只可能是010或者101，以某个位为中心元素，如果它是0，它们看两侧1
        # 如果它是1，那么看两侧0，左右相乘法；需要优化空间复杂度到1
        n, n_one = len(s), s.count("1")
        n_zero = n - n_one
        res = 0

        cnt_one = 0 # 到目前为止1的数量
        cnt_zero = 0 # 到目前为止0的数量
        for i in s:
            if i == "1":
                res += cnt_zero * (n - n_one - cnt_zero)
                cnt_one += 1
            else:
                res += cnt_one * (n_one - cnt_one)
                cnt_zero += 1
        return res
```

2268. Minimum Number of Keypresses

```python
class Solution:
    def minimumKeypresses(self, s: str) -> int:
        # Idea: 出现频率越高，越早抢占好位置
        # 得到频率之后，每个button上面有3个字母，左中右
        # 按照如下的顺序来填进去：
        # - 1-left, 2-left, 3-left, ... 9-left | 9 in total
        # 1-mid, 2-mid, 3-mid, ... 9-mid | 9 in total
        # 1-right, 2-right, 3-right, ... 8-right | 8 in total
        c = collections.Counter(s)
        ans = cnt = 0
        for i, freq in enumerate(sorted(c.values(), reverse=True)):
            if i % 9 == 0:
                cnt += 1
            ans += cnt * freq
        return ans
```

数字转换

8. String to Integer (atoi)

```python
class Solution:
    def myAtoi(self, s: str) -> int:
        idx = 0
        res = 0
        sign = 1

        # 空串
        if len(s) == 0:
            return 0
        # 去除空格
        while idx < len(s) and s[idx] == " ":
            idx += 1
        # 判断之后还有没有字符串
        if idx == len(s):
            return 0
        # 判断符号
        if s[idx] == "+":
            idx += 1
            sign = 1
        elif s[idx] == "-":
            idx += 1
            sign = -1
        # 逐个提取数字
        while idx < len(s):
            digit = ord(s[idx]) - ord("0")
            if digit < 0 or digit > 9:
                break
            res = res * 10 + digit
            if res > 2**31-1 and sign == 1:
                return 2**31-1
            elif res > 2**31 and sign == -1:
                return - 2**31
            idx += 1
        return sign * res
```

12. Integer to Roman

```python
class Solution:
    def intToRoman(self, num: int) -> str:
        # 构造映射之后贪心扣数字即可
        digit = [1000,900,500,400,100,90,50,40,10,9,5,4,1]
        sign = ["M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"]
        d = dict(zip(digit, sign))
        res = ""
        n = len(d)
        for i in range(n):
            while num - digit[i] >= 0:
                num -= digit[i]
                res += d[digit[i]]
        return res
```

13. Roman to Integer

```python
class Solution:
    def romanToInt(self, s: str) -> int:
        # 正常来说，前面的数字比后面的大，如果发现前面的小
        # 说明和后面是一个整体，当前位置实际是减去的意思
        res = 0
        n = len(s)
        s_to_n = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}
        for i in range(n):
            if i+1 <= n-1 and s_to_n[s[i]] < s_to_n[s[i+1]]:
                res -= s_to_n[s[i]]
            else:
                res += s_to_n[s[i]]
        return res
```

38. Count and Say

```python
class Solution:
    def countAndSay(self, n: int) -> str:
        # 暴力即可
        s = "1"
        for _ in range(n-1):
            temp = ""
            count = 0 # 数有多少连续的
            for i in range(len(s)):
                if i == 0:
                    count += 1
                elif s[i] == s[i-1]:
                    count += 1
                else:
                    temp += "%d%s"%(count, s[i-1])
                    count = 1
            s = temp + "%d%s"%(count, s[-1])
        return s
```

65. Valid Number

```python
class Solution:
    def isNumber(self, s: str) -> bool:

        # 先看是不是有e或者E
        # 有的话分割，左边浮点/整数，右边整数
        # 没有的话整个浮点
        # 浮点：+和-在最前面，至少有一个数字，至多有一个 "."
        s_list = s.replace("E", "e").split("e")

        def check_float(s):
            if len(s) == 0:
                return False
            idx = 0
            has_point = False
            valid = False
            if s[0] == "+" or s[0] == "-":
                idx += 1
            elif s[0] == ".":
                has_point = True
                idx += 1
            while idx < len(s):
                if not s[idx].isdigit():
                    if s[idx] != "." or has_point:
                        return False
                    else: # == "."
                        has_point += 1
                else:
                    valid = True
                idx += 1
            return valid

        def check_int(s):
            if len(s) == 0:
                return False
            idx = 0
            valid = False
            if s[idx] == "+" or s[idx] == "-":
                idx += 1
            while idx < len(s):
                if not s[idx].isdigit():
                    return False
                else:
                    valid = True
                idx += 1
            return valid > 0
```

```
        if len(s_list) == 1:
            return check_float(s_list[0])
        elif len(s_list) == 2:
            return check_float(s_list[0]) and check_int(s_list[1])
        else:
            return False
```

165. Compare Version Numbers

```
class Solution:
    def compareVersion(self, version1: str, version2: str) -> int:
        # 哪个短，就按照长的补全.0
        # 每个位置比较，处理方法就是把前面的0全部拿掉，然后转为整数
        def process(s):
            s = s.lstrip("0")
            return 0 if len(s) == 0 else int(s)
        dot1 = version1.count(".")
        dot2 = version2.count(".")
        if dot1 > dot2:
            version2 = version2 + ".0" * (dot1 - dot2)
        else:
            version1 = version1 + ".0" * (dot2 - dot1)
        v1 = [process(s) for s in version1.split(".")]
        v2 = [process(s) for s in version2.split(".")]
        for i in range(len(v1)):
            if v1[i] > v2[i]:
                return 1
            elif v1[i] < v2[i]:
                return -1
        return 0
```

273. Integer to English Words

```python
class Solution:

    def numberToWords(self, num: int) -> str:
        self.n_to_s = {
            1: "One",2: "Two",3: "Three",4: "Four",5: "Five",6: "Six",
            7: "Seven",8: "Eight",9: "Nine",10: "Ten",11: "Eleven",
            12: "Twelve",13: "Thirteen",14: "Fourteen",15: "Fifteen",
            16: "Sixteen",17: "Seventeen",18: "Eighteen",19: "Nineteen",
            20: "Twenty",30: "Thirty",40: "Forty",50: "Fifty",60: "Sixty",
            70: "Seventy",80: "Eighty",90: "Ninety",
        }
        self.billion, self.million = 1000000000, 1000000
        self.thousand, self.hundred = 1000, 100
        if num == 0:
            return "Zero"
        s = ""
        if num >= self.billion:
            s += self.numberToWords(num // self.billion)
            num %= self.billion
            s += " Billion"
        if num >= self.million:
            if len(s) != 0:
                s += " "
            s += self.numberToWords(num // self.million)
            num %= self.million
            s += " Million"
        if num >= self.thousand:
            if len(s) != 0:
                s += " "
            s += self.numberToWords(num // self.thousand)
            num %= self.thousand
            s += " Thousand"
        if num >= self.hundred:
            if len(s) != 0:
                s += " "
            s += self.numberToWords(num // self.hundred)
            num %= self.hundred
            s += " Hundred"
        for k in sorted(list(self.n_to_s), reverse=True):
            if num >= k:
                num -= k
                if len(s) != 0:
                    s += " "
                s += self.n_to_s[k]
        return s
```

299. Bulls and Cows

```
class Solution:
    def getHint(self, secret: str, guess: str) -> str:
        s_set, g_set = defaultdict(int), defaultdict(int)
        bull, cow = 0, 0
        for i in range(len(secret)):
            if secret[i] == guess[i]:
                bull += 1
            else:
                s_set[secret[i]] += 1
                g_set[guess[i]] += 1
        ks = set(s_set.keys()).intersection(set(g_set.keys()))
        for k in ks:
            cow += min(s_set[k], g_set[k])
        return "{}A{}B".format(bull, cow)
```

443. String Compression

```
class Solution:
    def compress(self, chars: List[str]) -> int:
        idx = 0 # 要开始改的地方
        ptr = 0 # 往前走的指针
        while ptr < len(chars):
            c = chars[ptr]
            count = 0
            while ptr < len(chars):
                if chars[ptr] == c:
                    count += 1
                    ptr += 1
                else:
                    break
            chars[idx] = c
            idx += 1
            if count == 1:
                continue
            else:
                tmp = str(count)
                length = len(tmp)
                chars[idx:idx+length] = list(tmp)
                idx += length
        return idx
```

539. Minimum Time Difference

```
class Solution:
    def findMinDifference(self, timePoints: List[str]) -> int:
        # 由鸽巢原理可知，如果长度超过1440
        # 那么必然会有两个相同的时间，此时可以直接返回0
        timePoints.sort()
        z = ord("0")
        def get_minutes(s):
            return 60 * ((ord(s[0]) - z) * 10
                  + (ord(s[1]) - z))
                  + ((ord(s[3]) - z) * 10 + (ord(s[4]) - z))
        last = get_minutes(timePoints[0])
        res = 10000
        for s in timePoints[1:]:
            cur = get_minutes(s)
            val = min(cur - last, last + 1440 - cur)
            res = min(res, val)
            last = cur
        # 注意是环形的，兜圈回来
        cur = get_minutes(timePoints[0])
        val = min(last - cur, cur + 1440 - last)
        res = min(res, val)
        return res
```

高精度运算

43. Multiply Strings

```python
class Solution:
    def multiply(self, num1: str, num2: str) -> str:
        digit_list = [0] * (len(num1) + len(num2))
        for i in range(len(num1)-1, -1, -1):
            for j in range(len(num2)-1, -1, -1):
                n1 = ord(num1[i]) - ord("0")
                n2 = ord(num2[j]) - ord("0")
                # i+j+1 可以把多个10^k权重一样的放到一个位置
                digit_list[i + j + 1] += n1 * n2
        carry = 0 # 进位
        for i in range(len(digit_list)-1, -1, -1):
            val = (digit_list[i] + carry) % 10
            carry = (digit_list[i] + carry) // 10
            digit_list[i] = val
        while digit_list and digit_list[0] == 0:
            digit_list.pop(0)

        res = "".join([str(i) for i in digit_list])
        return res if digit_list else "0"
```

67. Add Binary

```python
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        carry = 0
        if len(a) < len(b):
            a, b = b, a
        b = "0" * (len(a) - len(b)) + b
        carry = 0
        res = [0] * len(a)
        for i in range(len(a)-1, -1, -1):
            tmp = int(a[i]) + int(b[i]) + carry
            carry = tmp // 2
            res[i] = str(tmp % 2)
        if carry:
            return "".join(["1"] + res)
        else:
            return "".join(res)
```

415. Add Strings

```python
class Solution:
    def addStrings(self, num1: str, num2: str) -> str:
        add = False # 进位
        zero = ord("0")
        res = ""
        idx1, idx2 = len(num1)-1, len(num2)-1
        while idx1 >= 0 and idx2 >= 0:
            cur = int(add) + ord(num1[idx1]) - zero + ord(num2[idx2]) - zero
            res = str(cur % 10) + res
            add = cur >= 10
            idx1 -= 1
            idx2 -= 1
        while idx1 >= 0:
            cur = int(add) + ord(num1[idx1]) - zero
            res = str(cur % 10) + res
            add = cur >= 10
            idx1 -= 1
        while idx2 >= 0:
            cur = int(add) + ord(num2[idx2]) - zero
            res = str(cur % 10) + res
            add = cur >= 10
            idx2 -= 1
        if add:
            res = "1" + res
        return res
```

变换与匹配

5. Longest Palindromic Substring

```python
class Solution:
    def longestPalindrome(self, s: str) -> str:

        # center expanding method
        # the key point for optimization is we should find
        # all the same chars around i, then get expanding step

        # this is because if a longest string "ab(b)bb" does not
        # contain all the b's, then there exists a nearby char
        # which is different from b, (for example "a",) then it will
        # conflict with palindromic property in one side

        max_count = 0
        max_str = ""
        for i in range(len(s)):
            cur_count = 1
            left, right = i, i
            while left-1 >= 0 and s[left-1] == s[i]:
                left -= 1
                cur_count += 1
            while right+1 <= len(s)-1 and s[right+1] == s[i]:
                right += 1
                cur_count += 1
            while (
                left-1 >= 0 and right+1 <= len(s)-1
            ) and (s[right+1] == s[left-1]):
                left -= 1
                right += 1
                cur_count += 2
            cur_str = s[left:right+1]
            if cur_count > max_count:
                max_str = cur_str
                max_count = cur_count
        return max_str
```

6. Zigzag Conversion

```python
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        if numRows <= 1:
            return s
        # res列表里存放了被一行的结果
        # flag来控制遍历的方向，到首行或者尾行时候换方向
        res = ["" for _ in range(numRows)]
        row, flag = 0, -1
        for c in s:
            res[row] += c
            if row == 0 or row == numRows-1:
                flag *= -1
            row += flag
        return "".join(res)
```

28. Find the Index of the First Occurrence in a String

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:

        s, p = haystack, needle
        def build_next(s):
            # next数组中每一个元素代表当前元素和之前所有元素形成的
            # 子串里，最长的公共前后缀长度，例如"abcdabce"
            # [0, 0, 0, 0, 1, 2, 3, 0]
            # 由于需要线性遍历时间，考虑当前指针i，和之前的指针prev
            # 这个prev实际上是用来监控前缀最后一个字母和i是不是一样的
            # 如果一样的，说明可以向后扩充一个，如果不一样，那么由于
            # abcab[d] ... abcab[c]，这个d和c已经不匹配了，所以要拉到
            # 再前面一个b，也就是prev = next[prev−1]，然后插是不是和c一样了
            # 如果已经到头了（prev=0），说明无法发现公共前后缀，next填0

            # 那么，为什么时间复杂度是O(n)?因为在这个过程中prev最大的
            # 增加次数其实就是n次，所以对一个的减少次数也不会超过n次，
            # 因此均摊下来的复杂度是O(b)
            nxt = [0] # 第一个永远是0
            prev = 0
            i = 1

            while i < len(s):
                if s[i] == s[prev]:
                    prev += 1
                    i += 1
                    nxt.append(prev)
                elif prev:
                    prev = nxt[prev-1]
                else:
                    i += 1
                    nxt.append(0)
            return nxt

        nxt = build_next(p)
        s_ptr, p_ptr = 0, 0
        while s_ptr < len(s):
            if s[s_ptr] == p[p_ptr]:
                s_ptr += 1
                p_ptr += 1
            elif p_ptr:
                # p_ptr游标向前移动到待匹配位置 abc[e]abc[d]
                # 即，假设d匹配失败，那么直接匹配e
                p_ptr = nxt[p_ptr-1]
            else:
```

```
            # 这说明当前的s连一个p的字母都无法匹配
            s_ptr += 1

        if p_ptr == len(p):
            return s_ptr - p_ptr
    return -1
```

30. Substring with Concatenation of All Words

```python
class Solution:
    def findSubstring(self, s: str, words: List[str]) -> List[int]:

        w_num = len(words)
        w_len = len(words[0])
        t = w_num * w_len
        n = len(s)
        res = []
        d = defaultdict(int)
        for w in words:
            d[w] += 1

        for i in range(w_len):
            # i代表出发的地点，长度为3，则
            # 036...; 147...; 258...
            start = i
            cur_d = defaultdict(int)
            # 如果没有超出长度
            while start + t <= n:
                cur = s[start: start+t]
                if start == i:
                    # 第一次跑
                    cur = s[start]
                    for j in range(w_num):
                        subw = s[start+j*w_len:start+(j+1)*w_len]
                        if subw in d:
                            cur_d[subw] += 1
                else:
                    # 后面跑
                    before = s[start-w_len:start]
                    after = s[start+t-w_len:start+t]
                    if before in d:
                        # 把第一个拿掉
                        cur_d[before] -= 1
                    if after in d:
                        # 把最后一个加上
                        cur_d[after] += 1
                if (
                    all(w in cur_d for w in d)
                    and all(cur_d[w] == d[w] for w in cur_d)
                ):
                    res.append(start)
                start += w_len

        return res
```

68. Text Justification

```python
class Solution:
    def fullJustify(self, words: List[str], maxWidth: int) -> List[str]:
        res = []
        n = len(words)

        def get_left_and_right(i):
            left, right = i, i
            cur_len = len(words[right])
            right += 1 # right先加，这总是要加的
            while right < n: # 到n了说明都能放进最后一行
                if cur_len + 1 + len(words[right]) > maxWidth:
                    break
                cur_len += 1 + len(words[right])
                right += 1
            return left, right

        right = 0
        while right < n:
            left, right = get_left_and_right(right)
            sub_words = words[left: right]
            if right == n: # 到达最后一行
                cur_res = " ".join(sub_words).ljust(maxWidth, " ")
                res.append(cur_res)
                break
            n_word = len(sub_words) # 当前行的单词个数
            n_space = maxWidth - sum(len(w) for w in sub_words) # 空格个数
            space_count = (n_word - 1) # 需要填充的空格次数
            if space_count:
                # 不能整除说明有多余的空格
                space_len = n_space // space_count
                space_pad = n_space % space_count
            cur_res, idx = sub_words[0], 1 # idx是接下来要操作的单词
            while space_count:
                if space_pad:
                    # 把多余的空格放进来
                    cur_res += " "
                    space_pad -= 1
                cur_res += " "*space_len + sub_words[idx]
                space_count -= 1
                idx += 1
            cur_res = cur_res.ljust(maxWidth, " ") # 防止1个词独占1行的情况
            res.append(cur_res)
        return res
```

168. Excel Sheet Column Title

```
class Solution:
    def convertToTitle(self, columnNumber: int) -> str:
        res = ""
        columnNumber -= 1
        num_to_char = dict(zip(range(26), list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")))
        while True:
            res = num_to_char[columnNumber % 26] + res
            if columnNumber < 26: # 说明当前对应的是最后一位
                return res
            columnNumber = columnNumber // 26 - 1 # 26->1->A 26*2->2->B
        return res
```

171. Excel Sheet Column Number

```
class Solution:
    def titleToNumber(self, columnTitle: str) -> int:
        res = 0
        k = 0
        char_to_num = dict(
            zip(list("ABCDEFGHIJKLMNOPQRSTUVWXYZ"), range(26)))
        for c in columnTitle[::-1]:
            res += 26 ** k * (char_to_num[c] + 1)
            k += 1
        return res
```

214. Shortest Palindrome

```python
class Solution:
    def shortestPalindrome(self, s: str) -> str:
        # (aba)b # b(aba)
        # 本质上就是求公共前后缀，也就是KMP中的next数组最后一个元素
        def build_next(s):
            nxt = [0]
            prev = 0
            i = 1

            while i < len(s):
                if s[i] == s[prev]:
                    prev += 1
                    i += 1
                    nxt.append(prev)
                elif prev:
                    prev = nxt[prev-1]
                else:
                    i += 1
                    nxt.append(0)
            return nxt[-1]
        n = build_next(s+"#"+s[::-1])
        return s[n:][::-1] + s
```

336. Palindrome Pairs

```python
class Solution:
    def palindromePairs(self, words: List[str]) -> List[List[int]]:
        # 思路:
        # s1和s2加起来是回文，如果s1长度不超过s2
        # 那么
        # 1) s2的某个后缀肯定是s1逆序，而且前面是回文
        # 2) s2的某个前缀肯定是s1逆序，而且后面是回文
        # 因此可先用哈希把所有元素的逆序记下，然后迭代words和对应的分割点
        reverse_map = {w[::-1]: i for i, w in enumerate(words)}
        res = set()
        for idx, w in enumerate(words):
            for i in range(len(w)+1):
                w1 = w[:i]
                w2 = w[i:]
                if w1 in reverse_map and w2 == w2[::-1]:
                    j = reverse_map[w1]
                    if idx != j:
                        res.add((idx, j))
                if w2 in reverse_map and w1 == w1[::-1]:
                    j = reverse_map[w2]
                    if idx != j:
                        res.add((j, idx))
        return [list(i) for i in res]
```

418. Sentence Screen Fitting

```
class Solution:
    def wordsTyping(self, sentence: List[str], rows: int, cols: int) -> int:
        # 为什么需要这个尾巴的" "，因为两段拼接的话仍然需要一个空格
        s = " ".join(sentence) + " "
        n = len(s)
        ptr = 0
        for _ in range(rows):
            ptr += cols
            # 如果if成立，那么说明这里是恰好当且行塞满，空格不能作为下一行开头
            if s[ptr % n] == " ":
                # 下一行开始要是空格，直接补到下一个单词开头
                ptr += 1
            else:
                # 把上一行尾部被截断的单词给补回来
                while ptr > 0 and s[(ptr-1) % n] != " ":
                    ptr -= 1
        # 由于ptr指向的是下一个单词的开头，因为从0开始数的
        # 所以直接除n就是有几份
        return ptr // n
```

777. Swap Adjacent in LR String

```python
class Solution:
    def canTransform(self, start: str, end: str) -> bool:
        # 思路：先看是不是两个X数量相同，如果把X去掉，那么剩下的序列应该是一样的
        # 由于L只会往左边走，R只会往右边走，那么只需要看start的第i个L是不是index大
        # end里第i个L，以及start的第i个R是不是index小于等于end里的第i个R
        if start.count("X") != end.count("X"):
            return False
        i, j, n = 0, 0, len(start)
        while i < n and j < n:
            if start[i] == "X":
                i += 1
                continue
            if end[j] == "X":
                j += 1
                continue
            if start[i] != end[j]:
                return False
            if start[i] == "L" and i < j:
                return False
            if start[i] == "R" and i > j:
                return False
            i += 1
            j += 1
        return True
```

792. Number of Matching Subsequences

```
class Solution:
    def numMatchingSubseq(self, s: str, words: List[str]) -> int:
        # s长度Ls，wi长度为Lwi
        # 暴力解复杂度：Ls * max(Lwi)
        # 优化复杂度：Ls + \sum{Lwi}
        # 策略：
        # 造一个bucket，首字母一样的放到一个bucket，每次匹配一个字符
        # 匹配到了之后把列表依次元素弹出然后删去首字母
        # 再按照剩余首字母放到对应bucket
        # 弹出如果发现字符串长度为1，那么res += 1
        bucket = defaultdict(list)
        for w in words:
            bucket[w[0]].append(w)
        res = 0
        for c in s:
            ws = bucket[c]
            length = len(ws)
            for _ in range(length):
                w = ws.pop(0)
                if len(w) == 1:
                    res += 1
                else:
                    bucket[w[1]].append(w[1:])
        return res
```

833. Find And Replace in String

```
class Solution:
    def findReplaceString(self, s: str, indices: List[int], sources: List[str
        # 把字符串化作字符列表
        # [* * * * *]
        # [* * a b c d e]
        # 比如要从第三个开始要换成abcde
        # 直接把字符放在当前位置，其他用空字符串代替
        # [* * abcde * *]
        chars = list(s)
        for i, sr, t in zip(indices, sources, targets):
            # 也可以写成s.startswith(sr, i)
            if s[i:].startswith(sr):
                chars[i] = t
                for idx in range(i+1, i+len(sr)):
                    chars[idx] = ""
        return "".join(chars)
```

843. Guess the Word

```
# """
# This is Master's API interface.
# You should not implement it, or speculate about its implementation
# """
# class Master:
# def guess(self, word: str) -> int:

class Solution:

    def findSecretWord(self, wordlist, master):
        # O(N^2)
        def pair_matches(a, b):
            return sum(c1 == c2 for c1, c2 in zip(a, b))

        def most_overlap_word():
            counts = [defaultdict(int) for _ in range(6)]
            for word in candidates:
                for i, c in enumerate(word):
                    counts[i][c] += 1
            # 1. we have a guess word, x chosen
            # 2. this guess word x has overlap with other remaining
            # candidates. we use the number of overlapped letters to
            # bucket other candidates properly bucket range
            # 0, 1, 2, 3, 4, 5, 6.
            # 3. use probability knowledge, could easier figure that
            # bucket 0 should be the one with most possibility
            # POWER((25/26), 6) 80%
            # 4. we want to find a word that overlaps
            # most with other words so that we could eliminate all
            # words that might have 0 matches with the guess
            # word, this would help us to reach the secret.
            # 换句话说，让matches数字尽可能大，从而在更新
            # candidates的时候把0的淘汰掉
            return max(
                candidates,
                key=lambda x:sum(counts[i][c] for i, c in enumerate(x))
            )

        candidates = wordlist[:]
        while candidates:
            s = most_overlap_word()
            matches = master.guess(s)
            if matches == 6:
                return
            # 这里的逻辑是s和target差了多少
            # 那么根据matches筛选出来的结果，target一定也在里面
```

```
            candidates = [
                w for w in candidates
                if pair_matches(s, w) == matches
            ]
```

1044. Longest Duplicate Substring

```
class Solution:
    def longestDupSubstring(self, S: str) -> str:

        # 使用rabin-karp算法，进行滚动哈希
        p = 2**63 - 1
        def rabin_karp(mid):
            cur_hash = 0
            for i in range(mid):
                cur_hash = (cur_hash * 26 + nums[i]) % p
            hashes = {cur_hash}
            pos = -1
            max_pow = pow(26, mid, p)
            for i in range(mid, len(S)):
                cur_hash = (
                    26 * cur_hash
                    - nums[i-mid] * max_pow
                    + nums[i]) % p
                if cur_hash in hashes:
                    pos = i + 1 - mid
                hashes.add(cur_hash)
            return pos

        #bst
        nums = [ord(c) - ord("a") + 1 for c in S]
        l, r = 0, len(S) - 1
        start, end = 0, 0

        while (l <= r):
            mid = (l + r) // 2
            pos = rabin_karp(mid)
            if pos == -1:
                r = mid - 1
            else:
                start, end = pos, pos + mid
                l = mid + 1

        return S[start: end]
```

数位

位运算

89. Gray Code

```python
class Solution:
    def grayCode(self, n: int) -> List[int]:
        # 为什么需要镜像反转
        # a1, ..., am, am, ..., a1
        # 这样如果前面加0，后面加1，此时正好内部
        # 差1位，0am和1am差1,0a1和1a1差1
        res = [0]
        head = 1
        while n > 0:
            res += [i + head for i in res[::-1].copy()]
            # 巧妙用位运算
            head <<= 1
            n -= 1
        return res
```

136. Single Number

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        # 交换律: a ^ b ^ c <=> a ^ c ^ b
        # 任何数于0异或为任何数 0 ^ n => n
        # 相同的数异或为0: n ^ n => 0
        res = nums[0]
        for i in range(1, len(nums)):
            res ^= nums[i]
        return res
```

137. Single Number II

```
# 这题用java写
class Solution {
    public int singleNumber(int[] nums) {
        // 一堆数都出现了三次，那么他们每一位二进制加起来
        // 都是能被3整除的，所以每一位加起来的二进制mod3余数
        int res = 0;
        for(int i = 0; i < 32; i++){
            int sum = 0;
            for(int n: nums)
                // 获取第i位是否为1
                if((n >> i & 1) == 1)
                    sum++;
            sum %= 3;
            // 把s（0或1）放到res的相应位置
            res |= sum<<i;
        }
        return res;
    }
}
```

190. Reverse Bits

```python
class Solution:
    def reverseBits(self, n: int) -> int:
        res = 0
        for i in range(32):
            if n & (1 << i):
                res += 1 << (31-i)
        return res

"""
# 逐层交换的骚操作，学习一个
class Solution {
private:
    const uint32_t M1 = 0x55555555; // 01010101010101010101010101010101
    const uint32_t M2 = 0x33333333; // 00110011001100110011001100110011
    const uint32_t M4 = 0x0f0f0f0f; // 00001111000011110000111100001111
    const uint32_t M8 = 0x00ff00ff; // 00000000111111110000000011111111

public:
    uint32_t reverseBits(uint32_t n) {
        n = n >> 1 & M1 | (n & M1) << 1;
        n = n >> 2 & M2 | (n & M2) << 2;
        n = n >> 4 & M4 | (n & M4) << 4;
        n = n >> 8 & M8 | (n & M8) << 8;
        return n >> 16 | n << 16;
    }
};
"""
```

191. Number of 1 Bits

```
class Solution:
    def hammingWeight(self, n: int) -> int:
        return sum(n & (1 << i) != 0 for i in range(32))
"""
# 高级写法：利用n & (n−1)逐个把最低位的1变成0
class Solution:
    def hammingWeight(self, n: int) −> int:
        ret = 0
        while n:
            n &= n − 1
            ret += 1
        return ret
"""
```

201. Bitwise AND of Numbers Range

```
class Solution:
    def rangeBitwiseAnd(self, left: int, right: int) -> int:
        # 给定两个整数，我们要找到它们对应的二进制字符串的公共前缀
        # 如果某一位有差别，那么更底层的位一定已经变化了，&为0
        # 由于一定从某一位开始不一样了，而且一定是大的数字是1
        # 小的数字是0，因此while的条件是m < n
        shift = 0
        while left < right:
            left >>= 1
            right >>= 1
            shift += 1
        return left << shift
```

231. Power of Two

```
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        return n > 0 and (n & (n-1)) == 0
```

260. Single Number III

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> List[int]:
        # 利用除答案以外的元素均出现两次
        # 我们可以先对 nums 中的所有元素执行异或操作
        # 得到 n，n 为两答案的异或值（n 必然不为 0）。
        # 然后取 n 二进制表示中为 1 的任意一位 k
        # n 中的第 k 位为 1 意味着两答案的第 k 位二进制表示不同。
        # 对 nums 进行遍历，对第 k 位分别为 0 和 1 的元素分别
        # 求异或和（两答案必然会被分到不同的组），即为答案。

        # 有一位二进制不同即可
        n = 0
        k = -1
        for i in nums:
            n ^= i
        for i in range(32):
            if (n >> i) & 1:
                k = i
        res = [0, 0]
        for i in nums:
            if (i >> k) & 1:
                res[0] ^= i
            else:
                res[1] ^= i
        return res
```

318. Maximum Product of Word Lengths

```python
class Solution:
    def maxProduct(self, words: List[str]) -> int:
        # 把单词映射到26位整数表示
        # 每个单词用二进制表示，然后看&是不是0
        res = 0
        # 从唯一字母到由这些字母构成的最长单词的映射
        set_to_ws = defaultdict(str)
        for w in words:
            s = "".join(sorted(set(w)))
            if len(set_to_ws[s]) < len(w):
                # 去除重复字母
                set_to_ws[s] = w
        words = list(set_to_ws.values())

        word2 = words.copy()
        def get_bin(ws):
            return sum(1 << (ord(c)-97) for c in set(ws)), len(ws)

        for i, ws in enumerate(words):
            words[i] = get_bin(ws)

        for n, n_len in words:
            for m, m_len in words:
                if (n != m) and (n & m == 0):
                    res = max(res, n_len * m_len)
        return res
```

342. Power of Four

```python
class Solution:
    def isPowerOfFour(self, n: int) -> bool:
        return n > 0 and (n & (n - 1)) == 0 and n % 3 == 1
```

371. Sum of Two Integers

```
# 这道题用java
class Solution {
    public int getSum(int a, int b) {
        if (a == 0) return b;
        if (b == 0) return a;

        while (b != 0) {
            int carry = a & b; //将存在进位的位置置1
            a = a ^ b; // 计算无进位的结果
            b = carry << 1;
        }

        return a;
    }
}
```

1386. Cinema Seat Allocation

```python
class Solution:
    def maxNumberOfFamilies(self, n: int, reservedSeats) -> int:
        left, mid, right = 0b00001111, 0b11000011, 0b11110000
        row_to_bit = defaultdict(int)
        for s in reservedSeats:
            # 左右两端是否预约对结果无影响
            if 2 <= s[1] <= 9:
                row_to_bit[s[0]] |= 1 << (s[1] - 2)
        res = 2 * (n - len(row_to_bit)) # 没有预约，一排两个
        for row in row_to_bit:
            bit = row_to_bit[row]
            if (
                (left | bit) == left
                or (mid | bit) == mid
                or (right | bit) == right
            ):
                res += 1
        return res
```

数字

7. Reverse Integer

```python
class Solution:
    def reverse(self, x: int) -> int:
        MIN, MAX = -2**31, 2**31-1
        res = 0
        while x%10 == 0 and x != 0:
            x //= 10
        while x != 0:
            digit = x % 10 if x > 0 else -(-x % 10)
            res = res * 10 + digit
            if res > MAX or res < MIN:
                return 0
            # Python3 的整数除法在 x 为负数时会向下（更小的负数）取整
            # 因此不能写成 x //= 10
            x = x // 10 if x > 0 else -(-x // 10)
        return res
```

```java
class Solution {
    public int reverse(int x) {
        int res = 0, rem = 0;
        while(x != 0) {
            rem = x % 10;
            x = x/10;
            if(Math.abs(res) > (Integer.MAX_VALUE- Math.abs(rem))/10) return 0
            res = (res * 10) + rem;
        }
        return res;
    }
}
```

9. Palindrome Number

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        # 负数或者最低位为0的非0数，返回false
        if x < 0 or (x % 10 == 0 and x != 0):
            return False
        rev_x = 0
        # 直接把x的逆序数构造出来
        while x > rev_x:
            rev_x = rev_x * 10 + x % 10
            x //= 10
        # handle the case for both odd and even
        return x == rev_x or x == rev_x // 10
```

29. Divide Two Integers

```
class Solution:
    def divide(self, dividend: int, divisor: int) -> int:
        # x // y 的结果一定在[0, x]上，因此可以二分搜索
        # 快速乘法的原理：
        # 20 * 14    =    20 * (1110)2 = 20 * (2^3) * 1
        # + 20 * (2^2) * 1+20 * (2^1) * 1 +20 * (2^0) * 0 = 160+80+40=280.
        x, y = dividend, divisor
        if (x > 0) and (y < 0) or (x < 0) and (y > 0):
            neg = True
        else:
            neg = False
        x, y = abs(x), abs(y)

        def quick_mul(a, b):
            ans = 0
            while a != 0:
                if a & 1 == 1:
                    ans += b
                a >>= 1 # 一位一位取
                b += b # 其实就是倍乘
            return ans

        l, r = 0, x
        while l < r:
            mid = r - (r - l) // 2
            if quick_mul(mid, y) > x:
                r = mid - 1
            else:
                l = mid
        res = -l if neg else l
        if res > 2**31-1:
            return 2**31-1
        elif res < -2**31:
            return -2**31
        else:
            return res
```

50. Pow(x, n)

```
class Solution:
    def myPow(self, x: float, n: int) -> float:

        def helper(x, n):
            if x == 0:
                # 0 ^ 0 is not valid
                return 0
            if n == 0:
                return 1
            res = helper(x, n//2)
            res = res * res
            # if n is odd, then x^5 = x * x^2 * x^2
            return x * res if n%2==1 else res

        res = helper(x, abs(n))
        return res if n > 0 else 1/res
```

66. Plus One

```
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:
        carry = 0
        for i in range(len(digits)-1, -1, -1):
            tmp = 1 + digits[i]
            carry = tmp // 10
            digits[i] = tmp % 10
            if not carry:
                break
        if carry:
            return [1] + digits
        else:
            return digits
```

166. Fraction to Recurring Decimal

```python
class Solution:
    def fractionToDecimal(self, numerator: int, denominator: int):
        # 先搞负号，然后算整数
        # 用哈希表把算小数时候的余数和余数位置（方便插入括号）都记录下来
        # 如果余数在哈希表里说明有循环节了
        res = ""
        if numerator * denominator < 0:
            res += "-"
        # 记得弄成正数
        numerator = abs(numerator)
        denominator = abs(denominator)
        res += str(numerator // denominator)
        remainder = numerator % denominator
        if remainder == 0:
            return res
        res += "."

        r_to_pos = dict()
        while remainder and remainder not in r_to_pos:
            r_to_pos[remainder] = len(res)
            remainder *= 10
            res += str(remainder // denominator)
            remainder %= denominator

        if remainder in r_to_pos:
            pos = r_to_pos[remainder]
            res = res[:pos] + "(" + res[pos:] + ")"

        return res
```

172. Factorial Trailing Zeroes

```
class Solution:
    def trailingZeroes(self, n: int) -> int:
        five, two = 0, 0
        while n > 0:
            temp = n
            while temp != 0 and temp%5 == 0:
                temp //= 5
                five += 1
            while temp != 0 and temp%2 == 0:
                temp //= 2
                two += 1
            n -= 1
        return min(five, two)
"""
# 含有 2 的因子每两个出现一次，含有 5 的因子每 5 个出现一次，
# 所有 2 出现的个数远远多于 5，换言之找到一个 5，一定能找到一
# 个 2 与之配对。所以我们只需要找有多少个 5。

# n! = 1 * 2 * 3 * 4 * (1 * 5) * ... * (2 * 5) * ... * (3 * 5) *... * n
# 因为每隔 5 个数出现一个 5，所以计算出现了多少个 5，我们只需要
# 用 n/5 就可以算出来。
# 每隔 25 个数字，出现的是两个 5，所以除了每隔 5 个数算作一个 5
# 每隔 25 个数，还需要多算一个 5。
# 也就是我们需要再加上 n / 25 个 5。
public int trailingZeroes(int n) {
    int count = 0;
    while (n > 0) {
        count += n / 5;
        n = n / 5;
    }
    return count;
}
"""
```

204. Count Primes

```
import math
class Solution:
    def countPrimes(self, n: int) -> int:
        # 注意i遍历到最大\sqrt{n}即可，为n如果不是素数，那么至少有一个因子<=\sqrt
        ans = [True] * n
        max_i = int(math.sqrt(n))
        for i in range(2, max_i+1):
            if ans[i]:
                # 为什么从i*i开始，因为i*2~i*(i-1)都已经被右边的因子判断过了
                for j in range(i*i, n, i):
                    ans[j] = False
        return sum(i for i in ans[2:])
```

233. Number of Digit One

```
class Solution:
    def countDigitOne(self, n: int) -> int:
        """
        The idea is to calculate occurrence of 1 on every digit.
        There are 3 scenarios, for example
        if n = xyzdabc
        and we are considering the occurrence of one on thousand, it should be:

        (1) xyz * 1000 + 0
        if d == 0, means there is no 1 because of this digit(none)
        (2) xyz * 1000 + abc + 1
        if d == 1, means there is abc of 1 because of this digit(patrial)
        (3) xyz * 1000 + 1000
        if d > 1, means there is fully 1000 of 1 because of this digit(fully)
        """
        if n == 0:
            return 0
        res, d = 0, len(str(n))
        for i in range(d):
            d = n % (10 ** (i+1)) // (10 ** (i))
            xyz, abc = n // (10 ** (i+1)), n % (10 ** (i))
            if d == 0:
                res += xyz * 10 ** (i)
            elif d == 1:
                res += xyz * 10 ** (i) + abc + 1
            else:
                res += xyz * 10 ** (i) + 10 ** (i)
        return res
```

258. Add Digits

```
class Solution:
    def addDigits(self, num: int) -> int:
        while num // 10 != 0:
            temp = num
            res = 0
            while temp:
                res += temp % 10
                temp //= 10
            num = res
        return num
```

263. Ugly Number

```
class Solution:
    def isUgly(self, n: int) -> bool:
        # 注意负数和0
        if n <= 0:
            return False
        for i in [2, 3, 5]:
            while n >= 1 and n % i == 0:
                n //= i
        return n == 1
```

319. Bulb Switcher

```
from math import sqrt
class Solution:
    # 在[1,n]内有多少个数，其约数的个数为奇数。
    # 由于约数成对出现，如果奇数，肯定这个约束和另一个值一样
    def bulbSwitch(self, n: int) -> int:
        return int(sqrt(n))
```

326. Power of Three

```
class Solution:
    def isPowerOfThree(self, n: int) -> bool:
        # 在2**31内，最大的是3**19
        return n > 0 and 1162261467 % n == 0
```

343. Integer Break

```
class Solution:
    def integerBreak(self, n: int) -> int:
        # n=ax (分成a个)  -> y = (x^{1/x})^n
        # y = x^{1/x} 在e取最值
        if n == 2:
            return 1
        if n == 3:
            return 2
        num_3 = n // 3
        res = 3 ** (num_3 - 1)
        if n%3 == 0:
            return res * 3
        if n%3 == 1:
            return res * 4
        if n%3 == 2:
            return res * 3 * 2
```

357. Count Numbers with Unique Digits

```python
class Solution:
    def countNumbersWithUniqueDigits(self, n: int) -> int:
        res = 0
        while n:
            # 每次计算k位，最高位9选1，剩下的9选一，8选一，...
            tmp = 1
            factor = 9
            for _ in range(n-1):
                tmp *= factor
                factor -= 1
            res += 9 * tmp
            n -= 1
        return res + 1 # 最后把0算上
```

372. Super Pow

```
class Solution:
    def superPow(self, a: int, b: List[int]) -> int:
        n = 1337
        # 模的性质:
        # (a * b) % k = (a%k * b%k) % k
        # 5^1234 = 5^1230 * 5^4 = (5^123)^10 * 5^4 = ((5^12)^10 * 5^3)^ 10 * 5^4
        # 所以从右到左逐步展开

        def fast_pow(a, b):
            # 快速幂，b分奇数偶数讨论
            if b == 0:
                return 1
            if b % 2 == 0:
                res = fast_pow(a, b // 2) % n
                return (res * res) % n
            else:
                return (a % n * fast_pow(a, b - 1)) % n

        def compute(a, L):
            if L == []:
                return 1
            cur = L.pop()
            res1 = fast_pow(a, cur) # 最后一维开始
            res2 = fast_pow(compute(a, L), 10) # 这步是关键
            return (res1 * res2) % n

        return compute(a, b)
```

386. Lexicographical Numbers

```python
class Solution:
    def lexicalOrder(self, n: int) -> List[int]:
        # 如果我们将h位的所有整数看做如下所示h层的k叉树
        # [1,n] 范围内的所有整数的字典序实际上就是这棵k叉树的先序遍历顺序
        res = []
        num = 1
        while len(res) < n:
            # 因为所有<=n的元素一定都是res的元素，故按照顺序遍历的，放心加
            while num <= n:
                res.append(num)
                num *= 10
            while num % 10 == 9 or num > n:
                # 第一个条件是叶子节点最右侧是9，如果num是9了，那么
                # 一定表示要返回上层；如果num大了，那么也是要返回上层
                num //= 10
            num += 1
        return res
```

670. Maximum Swap

```
class Solution:
    def maximumSwap(self, num: int) -> int:
        # 从右到左找到每个位置的元素之后元素的最大值
        # 从左到右找到第一个最大值不是自己的，和最大值交换位置
        s = str(num)
        index = list(range(len(s)))
        max_val = -1
        max_index = -1
        for i in range(len(s)-1, -1, -1):
            cur_val = ord(s[i]) - ord("0")
            # 为什么不用判断相等的情况?
            # 我们希望如果max_index尽可能远离 (1993 98368)
            if cur_val < max_val:
                index[i] = max_index
            elif cur_val > max_val:
                max_index = i
                max_val = cur_val
        for i in range(len(s)):
            if index[i] != i:
                return s[:i] + s[index[i]] + s[i+1:index[i]] + s[i] + s[index[
        return s
```

1291. Sequential Digits

```
class Solution:
    def sequentialDigits(self, low: int, high: int) -> List[int]:
        res = []
        q = list(range(1, 10)) # 初始1,2,3,4,...,8,9
        while q:
            cur = q.pop(0)
            if low <= cur <= high:
                res.append(cur)
            # 看看还能不能向后扩张
            last = cur % 10
            if last != 9:
                new = cur * 10 + last + 1
                if new <= high:
                    q.append(new)
        return res
```

1492. The kth Factor of n

```
class Solution:
    def kthFactor(self, n: int, k: int) -> int:
        # 考虑一对一对的因子
        L1, L2 = [], []
        for i in range(1, int(sqrt(n))+1):
            if n % i == 0:
                L1.append(i)
                L2.append(n // i)
        if L1[-1] == L2[-1]:
            L2.pop()
        L = L1 + L2[::-1]
        return -1 if k > len(L) else L[k-1]
```

贪心算法

数组

31. Next Permutation

```
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        # Difficult Question!
        # 改变右边的比改变左边的造成的变化小，所以从
        # 右往左找到第一个nums[i-1]<nums[i]的pair
        i = len(nums)-1
        while i >= 1 and nums[i-1] >= nums[i]:
            i -= 1
        if i == 0:
            # 完全逆序
            nums[:] = nums[::-1]
            return
        # 从后向前在i到最后里找一个j大于i-1，然后和i-1交换
        j = len(nums)-1
        while nums[j] <= nums[i-1]:
            # 为什么是有=号，因此如果相等了
            # 换过去是同一个排列（[1,5,1]）
            j -= 1
        nums[i-1], nums[j] = nums[j], nums[i-1]
        # 交换后i之后的元素为降序，把它们倒过来使得尽可能小
        nums[i:] = nums[i:][::-1]
```

134. Gas Station

```python
class Solution:
    def canCompleteCircuit(self, gas: List[int], cost: List[int]) -> int:
        # 使用链接里的折线图分析
        # 从0出发的累计油量到最后应该是要不小于0的
        # 否则兜一圈油没了

        # 那么为了让中途不出现油量不足的情况
        # 就要找到累计油量出现最少的那个点
        # 然后从它的后一个点出发
        min_idx, min_remain = 0, gas[0] - cost[0]
        total = min_remain
        n = len(gas)
        for i in range(1, n):
            total += gas[i] - cost[i]
            if total < min_remain:
                min_remain = total
                min_idx = i
        if total < 0:
            return -1
        else:
            return (min_idx + 1) % n
```

135. Candy

```
class Solution:
    def candy(self, ratings: List[int]) -> int:

        # 左边遍历，如果左边比右边大则加1，右边遍历同理
        # 此时满足了必要条件，充分条件可以由x1<=y1且
        # x2<=y2得到max(x1,x2)<=max(y1,y2)知
        left = [1] * len(ratings)
        right = [1] * len(ratings)
        for i in range(1, len(ratings)):
            if ratings[i] > ratings[i-1]:
                left[i] = left[i-1] + 1
        for i in range(len(ratings)-2,-1,-1):
            if ratings[i] > ratings[i+1]:
                right[i] = right[i+1] + 1
        return sum(max(left[i], right[i]) for i in range(len(ratings)))
```

152. Maximum Product Subarray

```
class Solution:
    def maxProduct(self, nums: List[int]) -> int:

        # method 1:
        # since the negative * negative might be a large
        # value, we should track the max and min at the
        # same time
        """
        max_val, min_val = 1, 1
        res = float("-inf")
        for i in range(len(nums)):
            cur = nums[i]
            temp_max, temp_min = max_val, min_val
            # 以上一个结尾的最大和最小
            max_val = max(temp_max * cur, temp_min * cur, cur)
            min_val = min(temp_max * cur, temp_min * cur, cur)
            res = max(res, max_val)
        return res
        """

        # method 2:
        # this is a very tricky method
        # when negative count is even, product all
        # when ... is odd, assumed n, get the left n-1
        # and right n-1 negative val to product
        # thus, we can use arr and its reverse to record
        # this, and reset val after 0 to itself
        reverse_nums = nums[::-1]
        for i in range(1, len(nums)):
            nums[i] *= nums[i - 1] or 1
            reverse_nums[i] *= reverse_nums[i - 1] or 1
        return max(nums + reverse_nums)
```

169. Majority Element

```python
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        counter = 0
        for x in nums:
            if counter == 0:
                # 说明前面的都被抵消了，或者说后面的众数一定是总体的众数
                candidate = x
                counter += 1
            elif x == candidate:
                counter += 1
            else:
                counter -= 1
        return candidate
```

229. Majority Element II

```python
class Solution:
    def majorityElement(self, nums: List[int]) -> List[int]:
        # 答案最多只会有两个数字
        # 那么先钦定两个，然后后面和它们pk，如果不一样，则抵消
        # 怎么保证答案只有一个时候，不会被踢出candidate？原因在于
        # 剩下来不到2/3元素是一对一对抵消的，对数不会超过1/3，正好
        # 答案有充分的元素和它们抵消
        if len(nums) <= 2:
            return list(set(nums))
        candidate1, candidate2, vote1, vote2 = nums[0], nums[0], 0, 0
        for n in nums:
            # 在这两个candidates里
            if candidate1 == n:
                vote1 += 1
            elif candidate2 == n:
                vote2 += 1
            # 新来元素了，且前面已经有candidates可以被替换
            elif vote1 == 0:
                candidate1 = n
                vote1 += 1
            elif vote2 == 0:
                candidate2 = n
                vote2 += 1
            # 还是高频元素
            else:
                vote1 -= 1
                vote2 -= 1
        res = set()
        if sum(n==candidate1 for n in nums) > int(len(nums) // 3):
            res.add(candidate1)
        if sum(n==candidate2 for n in nums) > int(len(nums) // 3):
            res.add(candidate2)
        return list(res)
```

300. Longest Increasing Subsequence

```
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        # 考虑一个简单的贪心，如果我们要使上升子序列尽可能的长
        # 则我们需要让序列上升得尽可能慢，因此我们希望每次在上
        # 升子序列最后加上的那个数尽可能的小。基于上面的贪心思
        # 路，我们维护一个数组d[i]，表示长度为i的最长上升子序
        # 列的末尾元素的最小值。

        # it's obvious that dp is ordered
        dp = [nums[0]]
        for n in nums:
            if n > dp[-1]:
                dp.append(n)
                continue
            # find the pos of smallest num not less than n
            left, right = 0, len(dp)-1
            while left < right:
                mid = left + (right - left) // 2
                if dp[mid] < n:
                    # since this condition, we will
                    # go forward until meet a number
                    # whose value is not less than n
                    left = mid + 1
                else:
                    right = mid
            dp[left] = n
        return len(dp)
"""
# naive O(N^2) solution

class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        dp = [1] * len(nums)
        for i in range(len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
                    dp[i] = max(dp[i], dp[j] + 1)
        return max(dp)
"""
```

334. Increasing Triplet Subsequence

```python
class Solution:
    def increasingTriplet(self, nums: List[int]) -> bool:
        # 新元素，比small小则换small，比mid小则换mid，比mid大则true
        # 虽然small换掉了，和mid索引顺序不对，但是这样更好地更新mid，而且
        # 如果比mid大，也不会错过结果
        if len(nums) < 3:
            return False
        s, m = float("inf"), float("inf")
        for n in nums:
            # 一定要是取等号，bad case [1,1,-2,6]
            # 相当于把原本应该赋给mid的给截获了
            if n <= s:
                s = n
            elif n <= m:
                m = n
            elif n > m:
                return True
        return False
```

455. Assign Cookies

```
class Solution:
    def findContentChildren(self, g: List[int], s: List[int]) -> int:
        g.sort()
        s.sort()
        # 大的饼干优先分配
        g_ptr, s_ptr, res = len(g)-1, len(s)-1, 0
        while g_ptr >= 0 and s_ptr >= 0:
            while g_ptr >= 0 and g[g_ptr] > s[s_ptr]:
                g_ptr -= 1
            if g_ptr >= 0:
                res += 1
                g_ptr -= 1
                s_ptr -= 1
        return res
```

561. Array Partition

```python
class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        # 左右两两一组
        nums.sort()
        return sum(nums[::2])
```

575. Distribute Candies

```
"""
class Solution:
    def distributeCandies(self, candyType: List[int]) -> int:
        res = 0
        max_res = len(candyType) // 2
        last_type = -999999
        candyType.sort()
        for i in candyType:
            if res == max_res:
                return res
            if i != last_type:
                res += 1
                last_type = i
        return res
"""

class Solution:
    def distributeCandies(self, candyType: List[int]) -> int:
        # better solution
        # 考虑每种糖最多支持一个
        return min(len(set(candyType)), len(candyType) // 2)
```

605. Can Place Flowers

```python
class Solution:
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
        for i in range(len(flowerbed)):
            if flowerbed[i] == 0:
                if (
                        # 左边没花
                        i==0 or flowerbed[i-1]==0
                    ) and (
                        # 右边没花
                        i==len(flowerbed)-1 or flowerbed[i+1]==0
                    ):
                    flowerbed[i] = 1
                    n -= 1
        return n <= 0
```

767. Reorganize String

```python
class Solution:
    def reorganizeString(self, s: str) -> str:
        # 桶的数量等于频数最高字符的频数
        # 然后剩下的从头开始一直平铺过去
        # 最后检查有没有超过1个的只有1个元素的（只可能在末尾），有则返回""
        cnt = Counter(s)
        # 返回了一个列表，只有一个元素，是字符和频数的元组
        bucket_num = cnt.most_common(1)[0][1]
        idx = 0 # 平铺对应的索引
        bucket = [[] for i in range(bucket_num)]
        # ["abc", "ab", "ab", "a"]
        for c, n in cnt.most_common():
            for _ in range(n):
                bucket[idx].append(c)
                idx = (idx + 1) % bucket_num
        if sum(len(b)==1 for b in bucket) >= 2:
            # ["abc", "a", "a"]
            return ""
        else:
            return "".join(["".join(b) for b in bucket])
```

1304. Find N Unique Integers Sum up to Zero

```
class Solution:
    def sumZero(self, n: int) -> List[int]:
        if n % 2 == 0:
            res = []
        else:
            res = [0]
        i = 1
        while n > 1:
            res.append(i)
            res.append(-i)
            i += 1
            n -= 2
        return res
```

1578. Minimum Time to Make Rope Colorful

```python
class Solution:
    def minCost(self, colors: str, neededTime: List[int]) -> int:
        # 贪心，保留连续组里数值最大的
        if len(colors) == 1:
            return 0
        total_val = neededTime[0]
        last = colors[0]
        max_val = neededTime[0]
        res = 0
        for i in range(1, len(colors)):
            if colors[i] == last:
                max_val = max(neededTime[i], max_val)
            else:
                res += max_val # 需要扣掉的那个
                last = colors[i]
                max_val = neededTime[i]
            total_val += neededTime[i]
        return total_val - (res + max_val) # 这里max_val是处理最后一个
```

1647. Minimum Deletions to Make Character Frequencies Unique

```python
class Solution:
    def minDeletions(self, s: str) -> int:
        # 没填的坑，填的位置尽可能大
        d = defaultdict(int)
        for c in s:
            d[c] += 1
        res, used = 0, set()
        for c in d:
            freq = d[c]
            # 用过了的坑位不能再用
            while freq in used and freq > 0:
                freq -= 1
                res += 1
            used.add(freq)
        return res
```

1710. Maximum Units on a Truck

```python
class Solution:
    def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
        # 贪心，直接选最大unit的box即可
        boxTypes.sort(key=lambda x: -x[1])
        boxes = 0
        for box, units in boxTypes:
            if truckSize > box:
                truckSize -= box
                boxes += box * units
            else:
                boxes += truckSize * units
                return boxes
        return boxes
```

2178. Maximum Split of Positive Even Integers

```
class Solution:
    def maximumEvenSplit(self, finalSum: int) -> List[int]:
        # 贪心：2+4+6+...+2*n <= finalSum
        # 然后把加不下的最后一个数加到2*n上去
        if finalSum % 2 == 1:
            return []
        res = []
        cur = 2
        while finalSum - cur >= 0:
            res.append(cur)
            finalSum -= cur
            cur += 2
        res[-1] += finalSum
        return res
```

2214. Minimum Health to Beat Game

```
class Solution:
    def minimumHealth(self, damage: List[int], armor: int) -> int:
        # 最多能够抵御的血量
        return 1 + sum(damage) - min(max(damage), armor)
```

区间

56. Merge Intervals

```
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        intervals = sorted(intervals, key=lambda x: x[0])
        start = intervals[0][0]
        max_right = intervals[0][1]
        res = []
        for itv in intervals:
            if itv[0] > max_right:
                # the start and max_right processed here is the former
                # result, thus we need to add the last interval process
                # after the for loop
                res.append([start, max_right])
                start = itv[0]
                max_right = itv[1]
            max_right = max(max_right, itv[1])
        res.append([start, max_right])
        return res
```

57. Insert Interval

```python
class Solution:
    def insert(self, intervals: List[List[int]], newInterval: List[int]):
        res = []
        for itv in intervals:
            # newInterval代表可能要处理的下一个区间
            if newInterval[1] < itv[0]:
                res.append(newInterval)
                newInterval = itv
            elif itv[1] < newInterval[0]:
                # [itv] [new_itv]
                res.append(itv)
            else:
                # 有交集
                newInterval = [
                    min(newInterval[0], itv[0]),
                    max(newInterval[1], itv[1]),
                ]
        res.append(newInterval)
        return res
```

253. Meeting Rooms II

```python
class Solution:
    def minMeetingRooms(self, intervals: List[List[int]]) -> int:
        # 把开始点和结束点都排好序，然后模拟开始和结束的过程
        # 0.....30
        # 5....10
        # 7......13
        # [0, 5, 7]
        # [10, 13, 30]
        # 如果开始的时间指针的值小于结束指针的值，那么会议房数+1
        # 如果...大于等于...，那么房间数量-1
        # 直到开始指针越界，这个时候房间数量只会逐步减少
        res, count = 0, 0
        n = len(intervals)
        start = sorted(i[0] for i in intervals)
        end = sorted(i[1] for i in intervals)
        s_ptr, e_ptr = 0, 0
        # 注意这里e_ptr不越界的原因在于它不会超过s_ptr
        while s_ptr < n:
            if start[s_ptr] < end[e_ptr]:
                count += 1
                s_ptr += 1
                res = max(res, count)
            else:
                count -= 1
                e_ptr += 1
        return res
```

435. Non-overlapping Intervals

```python
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        # 把问题转化为最多能保留多少个区间，使他们互不重复，则按照终点排序
        # 每个区间的结尾很重要，结尾越小，则后面越有可能容纳更多的区间。
        intervals.sort(key=lambda x: x[1])
        res = 0
        last = intervals[0]
        for i in range(1, len(intervals)):
            cur = intervals[i]
            if cur[0] < last[1]:
                res += 1
            else:
                last = cur
        return res
```

452. Minimum Number of Arrows to Burst Balloons

```python
class Solution:
    def findMinArrowShots(self, points: List[List[int]]) -> int:
        # 右端点排序后，尽可能地往右边射箭
        points.sort(key=lambda x: x[1])
        res = 1
        last_end = points[0][1]
        for i in range(1, len(points)):
            cur_start = points[i][0]
            if cur_start > last_end:
                res += 1
                last_end = points[i][1]
        return res
```

759. Employee Free Time

```
"""
# Definition for an Interval.
class Interval:
    def __init__(self, start: int = None, end: int = None):
        self.start = start
        self.end = end
"""

class Solution:
    def employeeFreeTime(self, schedule: '[[Interval]]'):
        # 对所有区间排序，然后看空了哪个即可，O(Nlog(N))
        itvs = sorted(
            [i for itv in schedule for i in itv],
            key=lambda x: (x.start, x.end)
        )
        if len(itvs) == 0:
            return []
        res = []
        cur_end = itvs[0].end
        for itv in itvs[1:]:
            if itv.start > cur_end:
                res.append(Interval(cur_end, itv.start))
                cur_end = itv.end
            elif itv.end > cur_end:
                cur_end = itv.end
        return res
"""
# T: O(n*log(k)), n is the number of intervals across all employees
# k is the number of employees
# S: O(k)
class Solution:
    def employeeFreeTime(self, schedule: '[[Interval]]'):
        # 堆优化
        iterator = heapq.merge(
            *schedule, key=operator.attrgetter('start')
        )
        res, prev_end = [], next(iterator).end
        for event in iterator:
            if event.start > prev_end:
                res.append(Interval(prev_end, event.start))
            prev_end = max(prev_end, event.end)
        return res
"""
```

2158. Amount of New Area Painted Each Day

```python
class Solution:
    def amountPainted(self, paint: List[List[int]]) -> List[int]:
        seen = {}
        work_log = []

        for start, end in paint:
            work = 0
            while start < end:
                if start in seen:
                    # 通过已有的元素进行跳跃
                    tmp = seen[start]
                    seen[start] = max(seen[start], end) # more compress
                    start = tmp
                else:
                    # 一步一步走
                    seen[start] = end
                    work += 1
                    start += 1
            work_log.append(work)

        return work_log
```

字符串

179. Largest Number

```
"""
if s2 + s1 > s1 + s2 and s3 + s2 > s2 + s3, then we have s3 + s1 > s1 + s3

Proof:
s3 + s2 + s1 > s3 + s1 + s2 > s2 + s1 + s3 > s1 + s2 + s3.
so we have s3 + s2 + s1 > s1 + s2 + s3, which equals to s3 + s1 > s1 + s3.

Q.E.D
"""
class Solution:
    def largestNumber(self, nums: List[int]) -> str:
        nums = [str(i) for i in nums]
        nums.sort(
            reverse=True,
            key=cmp_to_key(lambda x, y: 1 if x+y > y+x else -1),
        ) # functools
        res = "".join(nums)
        if res[0] == "0":
            return "0"
        return res
```

409. Longest Palindrome

```
class Solution:
    def longestPalindrome(self, s: str) -> int:
        d = defaultdict(int)
        for i in s:
            d[i] += 1
        odd, even = 0, 0
        have_odd = False
        for i in d:
            if d[i] % 2 == 0:
                even += d[i]
            else:
                have_odd = True
                odd += d[i] - 1
        return even + odd + (have_odd)
```

1405. Longest Happy String

```python
class Solution:
    def longestDiverseString(self, a: int, b: int, c: int) -> str:
        res = []
        L = [[a, "a"], [b, "b"], [c, "c"]]
        # 贪心，每次总是取最大个数的元素拼接
        while True:
            L.sort(key=lambda x: -x[0])
            if L[0][0] <= 0:
                break
            has_valid = False
            for i, (n, c) in enumerate(L):
                if len(res) >= 2 and c == res[-1] and c == res[-2]:
                    continue
                if n > 0:
                    # 找到第一个可以拼接的元素
                    has_valid = True
                    res.append(c)
                    L[i][0] -= 1
                    break
            if not has_valid:
                # 为什么要判断valid，[10,1,1],
                # 没有可以拼的了
                break
        return "".join(res)
```

2193. Minimum Number of Moves to Make Palindrome

```python
class Solution:
    def minMovesToMakePalindrome(self, s: str) -> int:
        # 贪心算法是：每次固定字符串最左边的字母 a 不变
        # 找出距离字符串右侧最近的 a，把它交换到字符串最右边。
        # 这样字符串的头尾字母就相等了。把字符串的头尾去掉，
        # 就变成了子问题。把所有子问题的答案加起来就是最少交换次数。
        res = 0
        while len(s) > 1:
            c = s[0]
            idx = len(s)-1
            count = 0
            while s[idx] != c:
                idx -= 1
                count += 1
            if idx == 0:
                # 注意这里如果是右边找过去找到头上了
                # 说明只剩这一个，直接移动到当中，而不是两端
                res += (len(s) - 1) // 2
                s = s[1:]
            else:
                res += count
                s = s[1:idx] + s[idx+1:] # 去掉对应的两个字母
        return res
```

构造回文串的过程，实际上是每次选择一对字母并把它们交换到字符串头尾的过程。考虑字母 x 和字母 y 哪个先选，分以下情况讨论：

- 字母 x 和 y 的位置满足 $\underbrace{\cdots}_{a\text{ 个字母}}\ x\ \underbrace{\cdots}_{b\text{ 个字母}}\ y\ \underbrace{\cdots}_{c\text{ 个字母}}\ y\ \underbrace{\cdots}_{d\text{ 个字母}}\ x\ \underbrace{\cdots}_{e\text{ 个字母}}$。如果先把 x 换到头尾，再把 y 换到头尾，那么需要 $(a+e)+(b+d)$ 次交换；如果先换 y 再换 x，那么需要 $(a+b+1+d+e+1)+(a+e)$ 次交换。显然先换 x 更优。
- 字母 x 和 y 的位置满足 $\underbrace{\cdots}_{a\text{ 个字母}}\ x\ \underbrace{\cdots}_{b\text{ 个字母}}\ y\ \underbrace{\cdots}_{c\text{ 个字母}}\ x\ \underbrace{\cdots}_{d\text{ 个字母}}\ y\ \underbrace{\cdots}_{e\text{ 个字母}}$。如果先换 x 再换 y，那么需要 $(a+d+e+1)+(a+b+e)$ 次交换；如果先换 y 再换 x，那么需要 $(a+b+1+e)+(a+d+e)$ 次交换。先换哪个都一样。
- 字母 x 和 y 的位置满足 $\underbrace{\cdots}_{a\text{ 个字母}}\ x\ \underbrace{\cdots}_{b\text{ 个字母}}\ x\ \underbrace{\cdots}_{c\text{ 个字母}}\ y\ \underbrace{\cdots}_{d\text{ 个字母}}\ y\ \underbrace{\cdots}_{e\text{ 个字母}}$。如果先换 x 再换 y，那么需要 $(a+c+d+e+2)+(a+b+c+e)$ 次交换；如果先换 y 再换 x，那么需要 $(a+b+c+2+e)+(a+c+d+e)$ 次交换。先换哪个都一样。

上述讨论可以得到结论：每次交换最外边出现的字母不劣。因此贪心解法成立。

单调栈

84. Largest Rectangle in Histogram

```python
class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:
        # 找到左右两侧比该柱子矮的两个柱子的坐标（最近的）
        # 面积为两侧矮柱子之间的宽度乘以该柱子的高度
        left = [-1]
        right = [len(heights)]

        left_stack = [(0, heights[0])]
        for i in range(1, len(heights)):
            cur = heights[i]
            # 为什么是<=而不是<，比如3 0 1 1 1 1 (1)
            # 此时是小于的话，前面这一串1就加不进去了
            while left_stack != [] and cur <= left_stack[-1][1]:
                left_stack.pop(-1)
            if left_stack == []:
                left.append(-1)
            else:
                left.append(left_stack[-1][0])
            left_stack.append((i, cur))

        right_stack = [(len(heights)-1, heights[-1])]
        for i in range(len(heights)-2, -1, -1):
            cur = heights[i]
            while right_stack != [] and cur <= right_stack[-1][1]:
                right_stack.pop(-1)
            if right_stack == []:
                right.append(len(heights))
            else:
                right.append(right_stack[-1][0])
            right_stack.append((i, cur))

        res = 0
        right = right[::-1]
        for i in range(len(left)):
            res = max(res, heights[i] * (right[i]-left[i]-1))
        return res
```

85. Maximal Rectangle

```
class Solution:
    def maximalRectangle(self, matrix: List[List[str]]) -> int:
        m, n = len(matrix), len(matrix[0])
        for j in range(n):
            for i in range(m):
                if matrix[i][j] == "0":
                    matrix[i][j] = 0
                else:
                    matrix[i][j] = (int(matrix[i][j]) + (matrix[i-1][j]
                        if i-1 >= 0 else 0)
        return max(self.largestRectangleArea(matrix[i])
            for i in range(m))
    def largestRectangleArea(self, heights: List[int]) -> int:
        # 找到左右两侧比该柱子矮的两个柱子的坐标（最近的）
        # 面积为两侧矮柱子之间的宽度乘以该柱子的高度
        left = [-1]
        right = [len(heights)]
        left_stack = [(0, heights[0])]
        for i in range(1, len(heights)):
            cur = heights[i]
            # 为什么是<=而不是<，比如3 0 1 1 1 1 (1)
            # 此时是小于的话，前面这一串1就加不进去了
            while left_stack != [] and cur <= left_stack[-1][1]:
                left_stack.pop(-1)
            if left_stack == []:
                left.append(-1)
            else:
                left.append(left_stack[-1][0])
            left_stack.append((i, cur))

        right_stack = [(len(heights)-1, heights[-1])]
        for i in range(len(heights)-2, -1, -1):
            cur = heights[i]
            while right_stack != [] and cur <= right_stack[-1][1]:
                right_stack.pop(-1)
            if right_stack == []:
                right.append(len(heights))
            else:
                right.append(right_stack[-1][0])
            right_stack.append((i, cur))
        res = 0
        right = right[::-1]
        for i in range(len(left)):
            res = max(res, heights[i] * (right[i]-left[i]-1))
        return res
```

316. Remove Duplicate Letters

```python
class Solution:
    def removeDuplicateLetters(self, s: str) -> str:
        # 如果 k 是需要移除的次数
        # 对于每一个在字符串 s 中出现的字母 c 都有一个 k 值
        # 这个 k 是 c 出现次数 − 1
        letter_to_count = defaultdict(int)
        stack = []
        my_set = set()
        for c in s:
            # 统计词频
            letter_to_count[c] += 1
        for c in s:
            if c in my_set:
                letter_to_count[c] -= 1
                continue
            count = letter_to_count[c]
            while (
                stack
                and letter_to_count[stack[-1]] > 1
                and stack[-1] > c
            ):
                # stack[−1] 后面还有机会再放，不必放到前面
                # 这个时候注意my_set的remove
                letter_to_count[stack[-1]] -= 1
                my_set.remove(stack.pop(-1))
            stack.append(c)
            my_set.add(c)
        return "".join(stack)
```

321. Create Maximum Number

```python
class Solution:
    def maxNumber(self, nums1: List[int], nums2: List[int], k: int):
        # for遍历nums1选i，nums2选k−i，每个都选出最大子序列（单调栈见402）
        # 每个迭代时候合并，然后记录最大值
        m, n = len(nums1), len(nums2)
        def get_max(arr, i):
            remain = len(arr) - i
            stack = []
            ptr = 0
            for idx in range(len(arr)):
                cur = arr[idx]
                # i>0表示还能再remove
                while i > 0 and stack and stack[-1] < cur:
                    stack.pop(-1)
                    i -= 1
                stack.append(cur)
            return stack[:remain]
        def get_merged(A, B):
            # 从第一个开始比，直到有一个不一样或者到末尾
            ans = [0] * (len(A) + len(B))
            idx = 0
            while A or B:
                bigger = A if A > B else B
                ans[idx] = bigger[0]
                bigger.pop(0)
                idx += 1
            return ans
        res = [0] * k
        for i in range(k+1):
            arr1 = get_max(nums1, m-i)
            arr2 = get_max(nums2, n-(k-i))
            tmp = get_merged(arr1, arr2)
            # 这里是防止有地方要拿的比总数量还多
            if len(tmp) != k:
                continue
            if tmp > res:
                res = tmp
        return res
```

402. Remove K Digits

```python
class Solution:
    def removeKdigits(self, num: str, k: int) -> str:
        stack = []
        n = len(num)
        # 保留n-k个
        remain = n - k
        for i in range(n):
            cur = num[i]
            # 逐个遍历，如果栈顶元素大则弹出
            while k > 0 and stack and stack[-1] > cur:
                stack.pop(-1)
                k -= 1
            stack.append(cur)
        if len(stack) == 0:
            return "0"
        # 123456 -> [1,2,3,4,5,6]
        res = "".join([i for i in stack])[:remain].lstrip("0")
        if res == "":
            return "0"
        else:
            return res
```

496. Next Greater Element I

```
class Solution:
    def nextGreaterElement(self, nums1: List[int], nums2: List[int]):
        stack = []
        temp = [0] * len(nums2)
        res = []
        # 倒序遍历，如果新的数字比栈顶大就一直出栈
        for i in range(len(nums2)-1, -1, -1):
            cur = nums2[i]
            while stack != [] and stack[-1] <= cur:
                stack.pop(-1)
            # 本来是要取栈顶的，但是没元素了，只能取自己
            if stack == []:
                temp[i] = cur
            else:
                temp[i] = stack[-1]
            stack.append(cur)
        d = dict()
        for i in range(len(nums2)):
            d[nums2[i]] = temp[i]
        return [d[i] if i!=d[i] else -1 for i in nums1]
```

503. Next Greater Element II

```python
# 环形
class Solution:
    def nextGreaterElements(self, nums: List[int]) -> List[int]:
        return self._nextGreaterElement(nums, nums + nums)

    def _nextGreaterElement(self, nums1, nums2):
        stack = []
        temp = [0] * len(nums2)
        res = []
        # 倒序遍历，如果新的数字比栈顶大就一直出栈
        for i in range(len(nums2)-1, -1, -1):
            cur = nums2[i]
            while stack != [] and stack[-1] <= cur:
                stack.pop(-1)
            # 本来是要取栈顶的，但是没元素了，只能取自己
            if stack == []:
                temp[i] = -1
            else:
                temp[i] = stack[-1]
            stack.append(cur)
        return temp[:len(nums2) // 2]
```

556. Next Greater Element III

```python
# 使用相同数字，排列出next permutation
class Solution:
    def nextGreaterElement(self, n: int) -> int:
        # 本质上就是31-next-permutation
        nums = list([int(i) for i in str(n)])
        if len(nums) == 1:
            return -1
        i = len(nums) - 1
        # 改变右边的比改变左边的造成的变化小，所以从
        # 右往左找到第一个nums[i-1]<nums[i]的pair
        while i >= 1 and nums[i-1] >= nums[i]:
            i -= 1
        # 完全逆序
        if i == 0:
            return -1
        j = len(nums) - 1
        # 从后向前在i到最后里找一个j大于i-1，然后和i-1交换
        while nums[i-1] >= nums[j]:
            # 为什么是有=号，因此如果相等了
            # 换过去是同一个排列（[1,5,1]）
            j -= 1
        nums[j], nums[i-1] = nums[i-1], nums[j]
        nums[i:] = nums[i:][::-1]
        digit = 0
        res = 0
        for d in range(len(nums)):
            res += nums[len(nums)-d-1] * 10 ** digit
            if res >= 2 ** 31:
                return -1
            digit += 1
        return res
```

907. Sum of Subarray Minimums

```python
class Solution:
    def sumSubarrayMins(self, arr: List[int]) -> int:
        # 计算当前元素能够最为最小值的得分
        left = []
        right = []
        stack = []
        for i in range(len(arr)):
            # 避免重复计算，右边不允许有相等元素，但是左边可以有
            while stack and arr[stack[-1]] >= arr[i]:
                stack.pop(-1)
            if stack:
                left.append(stack[-1])
            else:
                left.append(-1)
            stack.append(i)
        stack = []
        for i in range(len(arr)-1, -1, -1):
            while stack and arr[stack[-1]] > arr[i]:
                stack.pop(-1)
            if stack:
                right.append(stack[-1])
            else:
                right.append(len(arr))
            stack.append(i)
        # right的是从右往左的，需要反过来
        right[:] = right[::-1]
        res = 0
        for i in range(len(arr)):
            res += ((i - left[i]) * (right[i] - i) * arr[i])
        return res % int(1e9 + 7)
```

1541. Minimum Insertions to Balance a Parentheses String

```
class Solution:
    def minInsertions(self, s: str) -> int:
        # 右括号不可能单独出现。遇到单独出现的右括号
        # 就必须补齐一个使其成对，然后尝试去抵消一个左括号
        stack, res = 0, 0 # 这里直接用数字的stack表示压了多少个左括号
        i = 0
        while i < len(s):
            c = s[i]
            if c == "(":
                stack += 1
                i += 1
            else:
                if i+1 < len(s):
                    if s[i+1] == "(":
                        if stack:
                            stack -= 1
                            res += 1
                        else:
                            res += 2 # 前面一个左，后面一个右
                        i += 1
                    else:
                        if stack:
                            stack -= 1
                        else:
                            res += 1
                        # 匹配到"))"了之后往后移两位
                        i += 2
                else:
                    if stack:
                        stack -= 1
                        res += 1
                    else:
                        res += 2
                    i += 1
        return res + stack * 2
```

2104. Sum of Subarray Ranges

```
class Solution:
    def subArrayRanges(self, nums: List[int]) -> int:
        # 本质上就是907
        # 需要思考，如果当前元素为子区间最小值，这样的子区间有多少个
        # 所以要求previous smaller和next smaller，也就是单调栈

        # 一个区间只能给一个最小值，我们规定最右边的是最小值
        # 如果相同元素，8448(4)，我们假设左侧可以延伸，右侧不行
        stack_pre = []
        left_bound = []
        # 为了方便处理，stack存下标
        for i in range(len(nums)):
            while stack_pre and nums[stack_pre[-1]] >= nums[i]:
                stack_pre.pop()
            if len(stack_pre) == 0:
                left_bound.append(-1)
            else:
                left_bound.append(stack_pre[-1])
            stack_pre.append(i)
        stack_nxt = []
        right_bound = []
        for i in range(len(nums)-1, -1, -1):
            while stack_nxt and nums[stack_nxt[-1]] > nums[i]:
                stack_nxt.pop()
            if len(stack_nxt) == 0:
                right_bound.append(len(nums))
            else:
                right_bound.append(stack_nxt[-1])
            stack_nxt.append(i)
        right_bound = right_bound[::-1]

        # min区间计数
        min_res = 0
        for i in range(len(nums)):
            min_res += (right_bound[i] - i)*(i - left_bound[i])*nums[i]

        ############################
        # 最大值完全相同的来一遍 #
        ############################
        stack_pre = []
        left_bound = []
        # 为了方便处理，stack存下标
        for i in range(len(nums)):
            while stack_pre and nums[stack_pre[-1]] <= nums[i]:
                stack_pre.pop()
            if len(stack_pre) == 0:
```

```
            left_bound.append(-1)
        else:
            left_bound.append(stack_pre[-1])
        stack_pre.append(i)
    stack_nxt = []
    right_bound = []
    for i in range(len(nums)-1, -1, -1):
        while stack_nxt and nums[stack_nxt[-1]] < nums[i]:
            stack_nxt.pop()
        if len(stack_nxt) == 0:
            right_bound.append(len(nums))
        else:
            right_bound.append(stack_nxt[-1])
        stack_nxt.append(i)
    right_bound = right_bound[::-1]

    # min区间计数
    max_res = 0
    for i in range(len(nums)):
        max_res += (right_bound[i] - i)*(i - left_bound[i])*nums[i]

    return max_res - min_res
```

2281. Sum of Total Strength of Wizards

```python
class Solution:
    def totalStrength(self, a: List[int]) -> int:
        # 单调栈 + 前缀和的前缀和
        n = len(a)

        left = [-1] * n
        stack = []
        for i in range(n):
            while stack and a[stack[-1]] >= a[i]:
                stack.pop()
            if stack:
                left[i] = stack[-1]
            stack.append(i)

        right = [n] * n
        stack = []
        for i in range(n-1, -1, -1):
            while stack and a[stack[-1]] > a[i]:
                # 左边可以延伸，但右边不行，否则重复算
                stack.pop()
            if stack:
                right[i] = stack[-1]
            stack.append(i)

        s = accumulate(a, initial=0)
        ss = list(accumulate(s, initial=0)) # init相当于前面插0

        ans = 0
        for i, v in enumerate(a):
            l, r = left[i] + 1, right[i] - 1 # 左闭右闭
            # total是lr区间里面所有子区间和的和
            total = (i - l + 1) * (ss[r + 2] - ss[i + 1])
            total -= (r - i + 1) * (ss[i + 1] - ss[l])
            ans += v * total

        return ans % (10 ** 9 + 7)
```

设子数组左端点为 l，右端点为 r，当前枚举的元素下标为 i，那么有 $L \le l \le i \le r \le R$。

设 $strength$ 数组的前缀和为 s，其中 $s[i] = \sum_{j=0}^{i-1} strength[j]$，因此子数组 $[l, r]$ 的元素和可以表示为

$$s[r+1] - s[l]$$

在范围 $[L, R]$ 内的所有子数组的元素和的和可以表示为

$$\begin{aligned}
&\sum_{r=i+1}^{R+1} \sum_{l=L}^{i} (s[r] - s[l]) \\
=&\left(\sum_{r=i+1}^{R+1} \sum_{l=L}^{i} s[r] \right) - \left(\sum_{r=i+1}^{R+1} \sum_{l=L}^{i} s[l] \right) \\
=&(i - L + 1) \cdot \sum_{r=i+1}^{R+1} s[r] - (R - i + 1) \cdot \sum_{l=L}^{i} s[l]
\end{aligned}$$

因此我们还需要计算出前缀和 s 的前缀和 ss，其中 $ss[i] = \sum_{j=0}^{i-1} s[j]$，上式即为

$$(i - L + 1) \cdot (ss[R+2] - ss[i+1]) - (R - i + 1) \cdot (ss[i+1] - ss[L])$$

再乘上 v 即为当前巫师的贡献，累加所有贡献即为答案。

2355. Maximum Number of Books You Can Take

```python
class Solution:
    def maximumBooks(self, books: List[int]) -> int:
        n = len(books)
        def guass_sum(right, count):
            count = min(count, right) # 因此减1，最左边不能是负的
            left = right - count + 1
            return (left + right) * (count) // 2

        res = 0
        stack = []
        cur = 0 # 当前书架全拿的最大值

        for i, c in enumerate(books):
            while stack and c - books[stack[-1]] < i - stack[-1]:
                # 先把前面加的给扣掉
                # *
                # * *
                # * * *
                # * * * *
                # * * * *
                # (*)* *(*)<-right
                right = stack.pop()
                tmp_count = right+1 if not stack else right-stack[-1]
                cur -= guass_sum(books[right], tmp_count)
            tmp_count = i+1 if not stack else i-stack[-1]
            cur += guass_sum(books[i], tmp_count)
            stack.append(i)
            res = max(res, cur)
        return res
```

哈希表

序列

1. Two Sum

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        # use hash table to store
        hash_table = {nums[0]: 0}
        for i in range(1, len(nums)):
            x = nums[i]
            query = target - x
            if query in hash_table:
                return [i, hash_table[query]]
            hash_table[x] = i
```

128. Longest Consecutive Sequence

```python
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        s = set(nums)
        res = 0

        for i in s:
            # 每次只看i作为开头第一个数的情况，也就是i-1不再s里
            if i-1 not in s:
                count, val = 1, i
                while val+1 in s:
                    val += 1
                    count += 1
                res = max(res, count)
        return res
```

187. Repeated DNA Sequences

```
class Solution:
    def findRepeatedDnaSequences(self, s: str) -> List[str]:
        add = set()
        seen = set()
        # 注意是len(s)−9不是−10
        for i in range(len(s)-9):
            cur = s[i:i+10]
            if cur in seen:
                add.add(cur)
            else:
                seen.add(cur)
        return list(add)
```

454. 4Sum II

```
class Solution:
    def fourSumCount(self,
        nums1: List[int], nums2: List[int],
        nums3: List[int], nums4: List[int]
    ):
        # 把i j k l分为i j和k l两组，先把i+j所有出现的情况用哈希表存
        # 然后k l查看−(k + l)在哈希表里的个数就行
        d = defaultdict(int)
        for i in nums1:
            for j in nums2:
                d[i+j] += 1
        res = 0
        for k in nums3:
            for l in nums4:
                res += d[-(k+l)]
        return res
```

532. K-diff Pairs in an Array

```
class Solution:
    def findPairs(self, nums: List[int], k: int) -> int:
        # 用两个hash或set（当哈希用）存已访问的数和已发现的k−diff中的较小值
        if k < 0:
            return 0
        seen, diff = set(), set()
        for i in nums:
            if i + k in seen:
                # 只记录前一个
                diff.add(i)
            if i - k in seen:
                # 只记录前一个
                diff.add(i - k)
            seen.add(i)
        return len(diff)
```

重复

202. Happy Number

```
"""
方法二:
若三位数，下一个比自己大的数字一定低于243
若4位或4位以上的数字在每一步都会丢失一位，直到降到3位为止
【证明】Suppose we have largest possible N−digit number:
        x1 = 999...999 = 10^N − 1
        Next number in the sequence will be: x2 = N ∗ 81 <= N ∗ 10^2
        and x2 will have less than N digits
        for any N > 3 (lg x2 + 1 < lg x1)
事实上它一定是一个从4开始的链
可以验证除了这个链上的数字都会归1
"""
class Solution:
    def isHappy(self, n: int) -> bool:
        # 方法一：用哈希检测是否出现环了
        s = set()
        while n != 1 and n not in s:
            s.add(n)
            n = self.get_n(n)
        return n == 1
    def get_n(self, n):
        res = 0
        while n:
            res += (n%10)**2
            n //= 10
        return res
```

205. Isomorphic Strings

```python
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        d = dict()
        for a, b in zip(s, t):
            if a not in d:
                d[a] = b
            else:
                # 如果出现过了，只能映射到同一个字符
                if d[a] != b:
                    return False
        d = dict()
        for a, b in zip(t, s):
            if a not in d:
                d[a] = b
            else:
                if d[a] != b:
                    return False
        return True
```

217. Contains Duplicate

```
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        return len(set(nums)) != len(nums)
```

219. Contains Duplicate II

```python
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int):
        # 这里remove巧妙使用数组索引进行元素定位
        s = set()
        for i in range(len(nums)):
            if i > k:
                s.remove(nums[i-k-1])
            if nums[i] in s:
                return True
            s.add(nums[i])
        return False
```

220. Contains Duplicate III

```python
class Solution:
    def containsNearbyAlmostDuplicate(
        self, nums: List[int], k: int, t: int
    ):
        # 先人为地搞出一堆区间，如果t=4，那么-8~-5 -4~-1 0~3 4~7 ...
        # 新来一个数字，什么情况会返回True?
        # 假设先不考虑k的索引范围限制，那么就是
        # 1) 当前映射的区间里有数字
        # 2) 前后的区间检查，发现确实两个差值小于等于t
        # 如果考虑k，那么只需要当第i个数字处理的时候，把
        # 第i-k个数字对应的桶删掉就ok了
        itv_to_item = dict()
        for i, n in enumerate(nums):
            itv = n // (t+1)
            if itv in itv_to_item:
                return True
            if itv-1 in itv_to_item:
                item = itv_to_item[itv-1]
                if n - item <= t:
                    return True
            if itv+1 in itv_to_item:
                item = itv_to_item[itv+1]
                if item - n <= t:
                    return True
            itv_to_item[itv] = n # 放在删桶前面是为了防止k=0
            if i-k >= 0:
                _itv = nums[i-k] // (t+1)
                if _itv in itv_to_item:
                    itv_to_item.pop(_itv)
        return False
```

244. Shortest Word Distance II

```
class WordDistance:

    def __init__(self, wordsDict: List[str]):
        self.d = defaultdict(list)
        # 存放的有序索引
        for i, item in enumerate(wordsDict):
            self.d[item].append(i)

    def shortest(self, word1: str, word2: str) -> int:
        # 把问题转化为从两个有序列表中找到最小的元素差
        L1, L2 = self.d[word1], self.d[word2]
        i, j = 0, 0
        res = 100000
        while i < len(L1) and j < len(L2):
            a, b = L1[i], L2[j]
            res = min(res, abs(a-b))
            if a > b:
                j += 1
            else:
                i += 1
        return res
```

290. Word Pattern

```
class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:
        s = s.split()
        if len(pattern) != len(s):
            return False

        d = dict()
        for a, b in zip(pattern, s):
            if a not in d:
                d[a] = b
            else:
                if d[a] != b:
                    return False

        d = dict()
        for a, b in zip(s, pattern):
            if a not in d:
                d[a] = b
            else:
                if d[a] != b:
                    return False

        return True
```

349. Intersection of Two Arrays

```
class Solution:
    def intersection(self, nums1, nums2):
        return list(set(nums1).intersection(set(nums2)))
```

350. Intersection of Two Arrays II

```
class Solution:
    def intersect(self, nums1: List[int], nums2: List[int]):
        d = dict()
        res = []
        for i in nums1:
            if i in d:
                d[i] += 1
            else:
                d[i] = 1
        for i in nums2:
            if i in d and d[i] >= 1:
                res.append(i)
                d[i] -= 1
        return res
```

355. Design Twitter

```
import collections
import heapq
import itertools


class Twitter(object):
    def __init__(self):
        self.timer = itertools.count(step=-1)
        self.tweets = collections.defaultdict(collections.deque)
        self.followees = collections.defaultdict(set)

    def postTweet(self, userId, tweetId):
        self.tweets[userId].appendleft((next(self.timer), tweetId))

    def getNewsFeed(self, userId):
        # heapq.merge就是把多个有序的合并成一个有序的
        # 注意| {userId} 需要并上自己的推文
        tweets = heapq.merge(*(self.tweets[u]
            for u in self.followees[userId] | {userId}))
        # 至多取前10项
        return [t for _, t in itertools.islice(tweets, 10)]

    def follow(self, followerId, followeeId):
        self.followees[followerId].add(followeeId)

    def unfollow(self, followerId, followeeId):
        self.followees[followerId].discard(followeeId)
```

359. Logger Rate Limiter

```
class Logger:

    def __init__(self):
        self.d = defaultdict(int)

    def shouldPrintMessage(self, timestamp: int, message: str) -> bool:
        if timestamp >= self.d[message]:
            self.d[message] = timestamp + 10
            return True
        else:
            return False
```

380. Insert Delete GetRandom O(1)

```python
class RandomizedSet:

    def __init__(self):
        # 这道题关键在于，python的list的append和pop最后一个元素是O(1)的
        # 因此我们只需要每次修改的时候保证删除的那个元素移到最后就行了
        self.nums = []
        self.d = dict()

    def insert(self, val: int) -> bool:
        if val in self.d:
            return False
        self.nums.append(val)
        self.d[val] = len(self.nums)-1
        return True

    def remove(self, val: int) -> bool:
        if val not in self.d:
            return False
        idx = self.d[val]
        val_last = self.nums[-1]
        self.nums[idx] = val_last
        self.d[val_last] = idx
        self.d.pop(val)
        self.nums.pop(-1)
        return True

    def getRandom(self) -> int:
        return random.choice(self.nums)
```

599. Minimum Index Sum of Two Lists

```python
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]):
        # 哈希表
        s = dict()
        val = float("inf")
        for i in range(len(list1)):
            if list1[i] not in s:
                s[list1[i]] = i
        for i in range(len(list2)):
            # 先把这个最小索引和拿到
            if list2[i] in s:
                if s[list2[i]] + i < val:
                    val = s[list2[i]] + i
        res = []
        for i in range(len(list2)):
            if list2[i] in s:
                if s[list2[i]] + i == val:
                    res.append(list2[i])
        return res
```

1817. Finding the Users Active Minutes

```python
class Solution:
    def findingUsersActiveMinutes(
        self,
        logs: List[List[int]],
        k: int
    ) -> List[int]:
        # 这道题统计的是活跃k分钟的用户有多少
        user_to_min = defaultdict(set)
        for i in logs:
            user_to_min[i[0]].add(i[1])
        res = [0] * k
        for key in user_to_min:
            res[len(user_to_min[key])-1] += 1
        return res
```

几何

939. Minimum Area Rectangle

```python
class Solution:
    def minAreaRect(self, points: List[List[int]]) -> int:
        # 一对一对遍历，但是利用哈希表记录
        # 然后查找相应的两个对称点是不是在哈希表里
        seen = set()
        res = float("inf")
        for x1, y1 in points:
            for x2, y2 in seen:
                if (x1, y2) in seen and (x2, y1) in seen:
                    val = abs(x1-x2) * abs(y1-y2)
                    if val < res:
                        res = val
            seen.add((x1, y1))
        return res if res != float("inf") else 0
```

2013. Detect Squares

```
class DetectSquares:

    def __init__(self):
        # 一个记录每个点上的数量
        # 一个记录同一条竖线(x)上有多少个点(y)
        self.xy_to_count = defaultdict(int)
        self.x_to_y = defaultdict(set)

    def add(self, point: List[int]) -> None:
        self.xy_to_count[tuple(point)] += 1
        self.x_to_y[point[0]].add(point[1])

    def count(self, point: List[int]) -> int:
        px, py = point
        x, ys = px, self.x_to_y[px]
        res = 0
        for y in ys:
            if y != py:
                k = py - y
                # 竖线两侧的正方形都要判断
                if (
                    (px+k, py) in self.xy_to_count
                    and (x+k, y) in self.xy_to_count
                ):
                    res += (
                        self.xy_to_count[(x,y)]
                        * self.xy_to_count[(px+k, py)]
                        * self.xy_to_count[(x+k, y)]
                    )
                if (
                    (px-k, py) in self.xy_to_count
                    and (x-k, y) in self.xy_to_count
                ):
                    res += (
                        self.xy_to_count[(x,y)]
                        * self.xy_to_count[(px-k, py)]
                        * self.xy_to_count[(x-k, y)]
                    )
        return res
```

链表

删除

19. Remove Nth Node From End of List

```
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int):

        # 先统计一共有多少
        count = 0
        cur = head
        while cur is not None:
            count += 1
            cur = cur.next

        # 倒数第n个，就是把头拿掉
        if n == count:
            return head.next
        cur = head
        # 跑到要删的前一个节点
        while count > n+1:
            cur = cur.next
            count -= 1
        cur.next = cur.next.next
        return head
```

82. Remove Duplicates from Sorted List II

```
# 全部扔掉
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]):
        dummy = ListNode(0, head)
        cur = dummy
        node = dummy.next
        while node:
            while node.next and node.val == node.next.val:
                node = node.next
            # 判断是否cur和node之间是否有重复的，只有当cur
            # 后面是node才说明node没因为重复而向前
            if cur.next == node:
                cur = cur.next
            else:
                # 这个时候cur.next，如果node后面还有重复，那么是可能变化的
                # 这里为什么要cur.next？cur.next表示的是下一步可能的连接节点
                cur.next = node.next
            # 每次后面那个节点总是要移动的
            node = node.next
        return dummy.next
```

83. Remove Duplicates from Sorted List

```
# 只留一个
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]):
        if head == None:
            return None
        cur = head
        val = cur.val
        node = head.next
        while node != None:
            if node.val != val:
                cur.next = node
                cur = node
                val = cur.val
            node = node.next
        cur.next = None
        return head
```

203. Remove Linked List Elements

```
class Solution:
    def removeElements(self, head: Optional[ListNode], val: int):
        dummy = ListNode(val=51)
        dummy.next = head
        last, cur = dummy, head
        while cur:
            if cur.val == val:
                last.next = cur.next
                cur = cur.next
            else:
                last = last.next
                cur = cur.next
        return dummy.next
```

237. Delete Node in a Linked List

```
class Solution:
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-place instead.
        """
        # 脑筋急转弯：把下一个节点的值拷贝过来，然后把下一个节点删掉
        node.val = node.next.val
        node.next = node.next.next
```

遍历

2. Add Two Numbers

```
# iteration
class Solution:
    def addTwoNumbers(self, l1, l2):
        # addition means 进位
        addition = False
        head = ListNode()
        temp = head
        while l1 != None or l2 != None:
            cur1 = l1.val if l1 else 0
            cur2 = l2.val if l2 else 0
            cur = addition + cur1 + cur2
            addition = (cur >= 10)
            cur %= 10
            temp.next = ListNode(cur)
            temp = temp.next
            l1 = l1.next if l1 else None
            l2 = l2.next if l2 else None
        if addition:
            temp.next = ListNode(1)
        return head.next
# recursion
class Solution:
    def addTwoNumbers(self, l1, l2):
        return self.helper(l1, l2, 0)
    def helper(self, l1, l2, addition):
        if l1 == None and l2 == None:
            return ListNode(1) if addition else None
        cur = (l1.val if l1 else 0) + (l2.val if l2 else 0) + addition
        cur_node = ListNode(cur % 10)
        if cur >= 10:
            addition = True
        cur_node.next = self.helper(
            l1.next if l1 else None,
            l2.next if l2 else None,
            cur >= 10
        )
        return cur_node
```

86. Partition List

```
class Solution:
    def partition(self, head: Optional[ListNode], x: int):
        # 虚拟头结点，注意放空节点避免产生环
        small, big = ListNode(), ListNode()
        small_head, big_head = small, big
        while head:
            if head.val >= x:
                big.next = head
                big = head
                # 这个时候把当前节点往前移，然后把big的tail放空
                head = head.next
                big.next = None
            else:
                small.next = head
                small = head
                head = head.next
                small.next = None
        small.next = big_head.next
        return small_head.next
```

138. Copy List with Random Pointer

```python
class Solution:
    def copyRandomList(self, head: 'Optional[Node]'):
        if head is None:
            return head
        # 下面复制一条单链
        cur = head
        while cur:
            node = Node(cur.val)
            node.next = cur.next # 新节点指向原节点的下一个
            cur.next = node # 原节点指向新节点（原链表断开）
            cur = node.next
        # 连接随机指针
        old, new = head, head.next
        while old:
            if old.random:
                old.next.random = old.random.next
            old = old.next.next # 原节点通过新节点移动到下一个节点
        # 分离两条链
        p, q = head, head.next
        res = q
        while p.next.next:
            p_ = q.next
            q_ = p_.next
            p.next = p_ # 原链表合并
            q.next = q_
            p = p.next
            q = q.next
        return res
```

160. Intersection of Two Linked Lists

```
class Solution:
    def getIntersectionNode(self, headA, headB):
        a, b = headA, headB
        # 自动处理None
        while a != b:
            a = a.next if a is not None else headB
            b = b.next if b is not None else headA
        return a
```

328. Odd Even Linked List

```
class Solution:
    def oddEvenList(self, head: Optional[ListNode]):
        if head is None or head.next is None:
            return head
        odd, even = head, head.next
        even_head = head.next
        # 分两类可能
        # 1) * # * # (*)odd (#)even
        # 2) * # * # * # (*)odd ()even
        # 所以最后只要连上odd即可
        while even and even.next:
            odd.next = even.next
            odd = odd.next
            even.next = odd.next
            even = odd.next
        odd.next = even_head
        return head
```

445. Add Two Numbers II

```
class Solution:
    def addTwoNumbers(self, l1, l2):
        # 用两个栈来反向存储
        stack1, stack2 = [], []
        head = l1
        while head:
            stack1.append(head.val)
            head = head.next
        head = l2
        while head:
            stack2.append(head.val)
            head = head.next

        res_stack = []
        carry = 0
        while stack1 or stack2:
            num1 = stack1.pop(-1) if stack1 else 0
            num2 = stack2.pop(-1) if stack2 else 0
            res = num1 + num2 + carry
            res_stack.append(ListNode(res % 10))
            carry = res // 10
        if carry:
            res_stack.append(ListNode(1))

        if res_stack:
            res = res_stack.pop(-1)
            head = res
            while res_stack:
                cur = res_stack.pop(-1)
                head.next = cur
                head = cur
            return res
        else:
            return None
```

旋转与反转

24. Swap Nodes in Pairs

```
class Solution:
    def swapPairs(self, head: Optional[ListNode]):
        dummy = ListNode(next=head)
        cur = dummy
        while cur and cur.next:
            # 先看好前面是不是保证有两个节点，否则啥也不干跳出来
            if cur.next and cur.next.next:
                node1 = cur.next
                node2 = cur.next.next
                node3 = cur.next.next.next
                cur.next = node2
                node2.next = node1
                node1.next = node3
                cur = node1
            else:
                break
        return dummy.next
```

25. Reverse Nodes in k-Group

```
class Solution:
    def reverseKGroup(self, head, k):
        if head is None or head.next is None:
            return head
        dummy = ListNode(0, head)
        pre, end = dummy, dummy
        while end.next is not None:
            k_ = k
            while end is not None and k_ > 0:
                k_ -= 1
                end = end.next
            if end is None:
                break
            # 记录下一组的头
            nxt = end.next
            end.next = None
            start = pre.next
            pre.next = self.reverse(start, end)
            start.next = nxt
            pre = start
            end = start
        return dummy.next
    def reverse(self, start, end):
        if start is None or start.next is None:
            return start
        pre = None
        cur = start
        while cur is not None:
            nxt = cur.next
            cur.next = pre
            pre = cur
            cur = nxt
        return pre
```

61. Rotate List

```
class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListN
        # 首先先把一共有几个点拿到
        temp = head
        n = 0
        while temp:
            temp = temp.next
            n += 1
        # 处理特殊情况
        if n == 0 or n == 1:
            return head
        k %= n
        if k == 0:
            return head
        # 把最后一个点拿到，然后连起来
        # （因为现在已经排除了k==n了，所以一定会连起来）
        last = head
        while last and last.next:
            last = last.next
        last.next = head
        # 拿到需要断开的那个节点
        drop = head
        step = n - k - 1
        while step:
            drop = drop.next
            step -= 1
        # 把它下一个节点，也就是结果返回的头保存一下
        res = drop.next
        drop.next = None
        return res
```

92. Reverse Linked List II

```
class Solution:
    def reverseBetween(self, head, left, right):
        if head is None:
            return head
        if left == right:
            return head
        dummy = ListNode(next=head)
        p = dummy
        for _ in range(left-1):
            p = p.next
        tail = p.next
        # 1 - (2) - (3) - 4
        # 1 ----------|
        ######(2)---(3)
        ######|------------4
        # 一个个节点转过来
        for _ in range(right-left):
            temp = p.next # p为1，next为上图中的3
            p.next = tail.next # 1连到4
            tail.next = tail.next.next # 2连到5
            p.next.next = temp # 4连到3
        return dummy.next
```

206. Reverse Linked List

```
# Recursion (please also see 156)
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head == None or head.next == None:
            return head
        new_head = self.reverseList(head.next)
        head.next.next = head
        head.next = None
        return new_head

# iteration
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if head == None or head.next == None:
            return head
        front, back = head, None
        # (* back) -> (* front) -> *
        while front:
            temp = front # 赋
            front = temp.next # 增
            temp.next = back # 回
            back = temp # 跳
        return back
```

2130. Maximum Twin Sum of a Linked List

```python
class Solution:
    def pairSum(self, head: Optional[ListNode]) -> int:
        # 0 (1) (2) 3
        # 前面那个走到n/2位置反转链表
        n = 0
        node = head
        while node:
            n += 1
            node = node.next
        step = n // 2
        node = head
        while step > 1:
            node = node.next
            step -= 1
        second_head = node.next # 第二段的头
        node.next = None

        pre, node = None, head
        while node:
            tmp = node.next
            node.next = pre
            pre = node
            node = tmp
        first_head = pre # 反转前的尾巴，现在的头

        res = 0
        while first_head:
            res = max(res, first_head.val + second_head.val)
            first_head = first_head.next
            second_head = second_head.next
        return res
```

排序与合并

21. Merge Two Sorted Lists

```
class Solution:
    def mergeTwoLists(self, list1, list2):
        if list1 == None:
            return list2
        if list2 == None:
            return list1
        # this is very exciting
        # at current state, we capture the smaller node and search
        # for the next node by recursion from its next and the node
        # on the other linked list
        if list1.val < list2.val:
            list1.next = self.mergeTwoLists(list1.next, list2)
            return list1
        else:
            list2.next = self.mergeTwoLists(list1, list2.next)
            return list2
```

23. Merge k Sorted Lists

```python
import heapq

class Solution:
    def mergeKLists(self, lists):
        # although we need not compare ListNode,
        # since we use heapq.heappop, so we have
        # to define a virtual order
        ListNode.__lt__ = lambda a, b: 0

        # first we add all first node from the LinkedList
        # to the heapq (L)
        L = []
        if len(lists) == 0:
            return None
        for node in lists:
            if node:
                heapq.heappush(L, (node.val, node))

        dummy = ListNode()
        cur = dummy
        while L:
            # we can ensure that the popped element
            # is not None
            val, node = heapq.heappop(L)
            # we connect the current popped val to the cur node
            cur.next = ListNode(val)
            cur = cur.next
            # if node is None, we will not add it
            if node.next:
                heapq.heappush(L, (node.next.val, node.next))

        return dummy.next
```

147. Insertion Sort List

```
class Solution:
    def insertionSortList(self, head):
        # 准备一个dummy，先不连，然后把新的节点一个个拿过去
        dummy = ListNode()
        pre = dummy # 要准备连的节点
        cur = head
        while cur is not None:
            nxt = cur.next
            while pre.next is not None and pre.next.val < cur.val:
                # 找到当前cur要插入的pre位置
                pre = pre.next
            cur.next = pre.next
            pre.next = cur
            pre = dummy # 重置节点
            cur = nxt # 把之前拿到的下一个节点给cur，准备下一轮插入
        return dummy.next
```

148. Sort List

```
class Solution:
    def sortList(self, head):
        # base case
        if head is None or head.next is None:
            return head

        left = head
        right = self.get_right(head) # 获得分割指针的前一个节点
        tmp = right.next
        right.next = None
        right = tmp # 指向后一个节点，也就是真正的right头

        left = self.sortList(left)
        right = self.sortList(right)
        return self.merge(left, right)

    def get_right(self, head):
        slow, fast = head, head.next
        while fast is not None and fast.next is not None:
            slow = slow.next
            fast = fast.next.next
        return slow

    def merge(self, l, r):
        dummy = ListNode(-1)
        cur = dummy
        while l is not None and r is not None:
            if l.val < r.val:
                tmp = l.next
                cur.next = l
                l.next = None
                l = tmp
            else:
                tmp = r.next
                cur.next = r
                r.next = None
                r = tmp
            cur = cur.next
        if l is not None:
            cur.next = l
        if r is not None:
            cur.next = r
        return dummy.next
```

快慢指针

141. Linked List Cycle

```
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        # 快慢指针3步套路：
        # 首先判断fast和fast.next是否空
        # 然后步进，最后判断是否相等
        fast = head
        slow = head
        while True:
            if fast == None or fast.next == None:
                return False
            slow = slow.next
            fast = fast.next.next
            if slow == fast:
                return True
        # 为什么保证可以碰到？因为每次相当于快的和慢的扩大1，
        # 碰到只需间隔增大到环的整数倍即可

        # 假设环外部分长度为a，slow指针进入环后
        # 又走了b的距离与fast相遇。此时，fast指针已经走完了环的n圈
        # 因此它走过的总距离为： a+n(b+c)+b=a+(n+1)b+nc
        # 这又等于2(a + b)，因此a=c+(n−1)(b−c)
        # 因此，只需要再添加一个指针指向头结点，当它走完环外距离a的时候
        # 则会与在绕圈等它的slow相遇。而相遇点恰好是入环点
```

142. Linked List Cycle II

```
class Solution:
    def detectCycle(self, head: Optional[ListNode]) -> Optional[ListNode]:
        slow, fast = head, head
        while True:
            if fast == None or fast.next == None:
                return None
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                break
        fast = head
        while fast != slow:
            fast = fast.next
            slow = slow.next
        return slow
        # 假设环外部分长度为a，slow指针进入环后
        # 又走了b的距离与fast相遇。此时，fast指针已经走完了环的n圈
        # 因此它走过的总距离为： a+n(b+c)+b=a+(n+1)b+nc
        # 这又等于2(a + b)，因此a=c+(n−1)(b−c)
        # 因此，只需要再添加一个指针指向头结点，当它走完环外距离a的时候
        # 则会与在绕圈等它的slow相遇。而相遇点恰好是入环点
```

143. Reorder List

```
class Solution:
    def reorderList(self, head: Optional[ListNode]) -> None:
        """
        Do not return anything, modify head in-place instead.
        """
        # 先把中点找出来，然后断开
        node1, mid, last = head, head, head
        while last != None and last.next != None:
            last = last.next.next
            mid = mid.next
        node2 = mid.next
        mid.next = None

        # 反转后面那条链表
        pre = None
        cur = node2
        while cur != None:
            temp = cur.next
            cur.next = pre
            pre = cur
            cur = temp
        node2 = pre

        # 合并链表
        while node1 != None and node2 != None:
            temp1 = node1.next
            temp2 = node2.next
            node1.next = node2
            node1 = temp1
            node2.next = node1
            node2 = temp2

        return node1
```

234. Palindrome Linked List

```
        # 反转和遍历同时进行
        pre, cur, fast = None, head, head
        while fast != None and fast.next != None:
            fast = fast.next.next
            temp = cur.next
            cur.next = pre
            pre = cur
            cur = temp
        if fast != None:
            # 1 (2)_pre (3)_cur 2 1 的情况
            # 跳过中心节点
            cur = cur.next
        # 这里pre和cur分别是左右出发的节点
        while cur != None:
            if cur.val != pre.val:
                return False
            cur = cur.next
            pre = pre.next
        return True
```

287. Find the Duplicate Number

```
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        # 如果有重复数字，则下标n->f(n)的映射必定会形成环
        # 找到数组中的重复整数 <==> 找到链表的环入口
        slow, fast = 0, 0
        while True:
            slow = nums[slow]
            fast = nums[nums[fast]]
            if slow == fast:
                break
        fast = 0
        # 注意快慢指针指的就是元素（node）而不是索引
        while fast != slow:
            fast = nums[fast]
            slow = nums[slow]
        # 假设环外部分长度为a，slow指针进入环后
        # 又走了b的距离与fast相遇。此时，fast指针已经走完了环的n圈
        # 因此它走过的总距离为： a+n(b+c)+b=a+(n+1)b+nc
        # 这又等于2(a + b)，因此a=c+(n-1)(b-c)
        # 因此，只需要再添加一个指针指向头结点，当它走完环外距离a的时候
        # 则会与在绕圈等它的slow相遇。而相遇点恰好是入环点
        return slow
```

缓存应用

146. LRU Cache

```
class Node:

    def __init__(self, key=0, value=0):
        self.key = key
        self.value = value
        self.prev = None
        self.next = None

class LRUCache:

    def __init__(self, capacity: int):
        # 缓存数据的键映射到其在双向链表中的位置
        self.cache = {}
        # 虚拟头部
        self.head = Node()
        # 虚拟尾部
        self.tail = Node()
        # head -> tail (next方向向右)
        # <- (prev方向向左)
        self.head.next = self.tail
        self.tail.prev = self.head
        self.c = capacity
        # 实际节点个数
        self.count = 0

    def get(self, key: int) -> int:
        if key not in self.cache:
            return -1
        node = self.cache[key]
        # 断开节点
        self.delete(node)
        # 插入到头上
        self.add_node_to_head(node)
        return node.value

    def put(self, key: int, value: int) -> None:
        if key not in self.cache:
            node = Node(key, value)
            self.cache[key] = node
            self.add_node_to_head(node)
            self.count += 1
            # 容量已经满了
            if self.count > self.c:
                node = self.delete_last_node()
                self.count -= 1
        else:
```

```
            node = self.cache[key]
            node.value = value
            self.delete(node)
            self.add_node_to_head(node)

    def add_node_to_head(self, node):
        node.prev = self.head
        node.next = self.head.next
        self.head.next.prev = node
        self.head.next = node

    def delete(self, node):
        node.prev.next = node.next
        node.next.prev = node.prev

    def delete_last_node(self):
        node = self.tail.prev
        self.delete(node)
        self.cache.pop(node.key)


# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

460. LFU Cache

```
#DEBUG
#print("get", key, self.hash_key_to_node.keys(), self.hash_freq_to_dll.keys())
#print("put", key, value, self.hash_key_to_node.keys(), self.hash_freq_to_dll.keys())

class Node:

    def __init__(self, key=0, value=0, freq=1):
        self.key = key
        self.value = value
        self.freq = freq
        self.prev = None
        self.next = None

class DoubleLinkedList:

    def __init__(self, freq):
        self.head = Node(freq=freq)
        self.tail = Node(freq=freq)
        self.head.next = self.tail
        self.tail.prev = self.head

class LFUCache:

    # 当容量到了，先看freq最小的，然后看时间最远的
    def __init__(self, capacity: int):
        # 从key到节点的映射
        self.hash_key_to_node = {}
        # 从freq到链表的映射
        self.hash_freq_to_dll = {}
        self.c = capacity
        self.count = 0
        # 用于O(1)找到当前最小freq的
        self.minfreq = 0

    def get(self, key: int) -> int:
        if key not in self.hash_key_to_node:
            return -1
        else:
            node = self.hash_key_to_node[key]
            # 节点位置变动，检查当前DLL是不是只有这个节点
            # 如果是，且当前DLL的freq就是minfreq，那么需要+1
            self.check_freq(node)
            self.move(node)
            node.freq += 1
            self.add(node)
            return node.value
```

```
    def put(self, key: int, value: int) -> None:
        if key not in self.hash_key_to_node:
            # 这里的容量为0时要单独讨论
            if self.c != 0:
                self.count += 1
                if self.count > self.c:
                    self.pop_last()
                    self.count -= 1
                self.minfreq = 1
                node = Node(key, value)
                self.add(node)
                self.hash_key_to_node[node.key] = node
        else:
            # 拿到节点->更新值->检查minfreq->换位置和freq
            node = self.hash_key_to_node[key]
            node.value = value
            self.check_freq(node)
            self.move(node)
            node.freq += 1
            self.add(node)

    def check_freq(self, node):
        if node.freq == self.minfreq:
            cur_dll = self.hash_freq_to_dll[node.freq]
            if cur_dll.head.next == node and cur_dll.tail.prev == node:
                # 表示当前minfreq的其实只有一个节点，因此
                # 移走了之后minfreq增加1
                self.minfreq += 1

    def pop_last(self):
        dll = self.hash_freq_to_dll[self.minfreq]
        node = dll.tail.prev
        self.move(node)
        # pop操作只会在这个函数内存在
        self.hash_key_to_node.pop(node.key)

    def move(self, node):
        node.prev.next = node.next
        node.next.prev = node.prev

    def add(self, node):
        # 如果当前freq的DLL不存在，那么需要新建
        if node.freq not in self.hash_freq_to_dll:
            self.hash_freq_to_dll[node.freq] = DoubleLinkedList(node.freq)
        dll = self.hash_freq_to_dll[node.freq]
        # 拿到DLL后，把当前node塞到头的后面
        node.prev = dll.head
```

```
        node.next = dll.head.next
        dll.head.next.prev = node
        dll.head.next = node


# Your LFUCache object will be instantiated and called as such:
# obj = LFUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

双指针

头尾指针

11. Container With Most Water

```python
class Solution:
    def maxArea(self, height: List[int]) -> int:
        # 关键点在于，总是向内收缩端短的
        # 因为能够装的水取决于短的，长的向内移动不会使水增加
        water = lambda x, y: (y-x) * min(height[x], height[y])
        res = float("-inf")
        left, right = 0, len(height)-1
        while left < right:
            cur = water(left, right)
            res = max(res, cur)
            if height[left] > height[right]:
                right -= 1
            else:
                left += 1
        return res
```

15. 3Sum

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        if len(nums) < 3:
            return []
        res = []
        nums.sort()
        for i in range(len(nums)-2):
            cur = nums[i]
            if cur > 0:
                return res
            # 这里是为了去重，如果前面已经有同样的数作为第一个值
            # 那么这里不用再考虑了，前面的三元组肯定已经包含
            if i > 0 and nums[i] == nums[i-1]:
                continue
            left, right = i+1, len(nums)-1
            while left < right:
                target = -cur
                if nums[left] + nums[right] > target:
                    right -= 1
                elif nums[left] + nums[right] < target:
                    left += 1
                else:
                    res.append([cur, nums[left], nums[right]])
                    # we should tight the bound to deduplicate
                    while left < right and nums[left] == nums[left+1]:
                        left += 1
                    while left < right and nums[right] == nums[right-1]:
                        right -= 1
                    # we should also adjust left and right
                    left += 1
                    right -= 1
        return res
```

16. 3Sum Closest

```
class Solution:
    def threeSumClosest(self, nums: List[int], target: int) -> int:
        nums.sort()
        diff = float("inf")
        while len(nums) >= 3:
            cur = nums.pop(0)
            i, j = 0, len(nums)-1
            while i < j:
                val = cur + nums[i] + nums[j]
                val_diff = abs(target - val)
                if diff > val_diff:
                    diff = val_diff
                    res = val
                if val > target:
                    j -= 1
                elif val < target:
                    i += 1
                else:
                    return target
        return res
```

18. 4Sum

```
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        # 方法一：三重for循环，hash去重
        # 方法二：固定i和j，剩下一个数转为two sum
        n = len(nums)
        nums.sort()
        res = set()
        for i in range(n):
            for j in range(i+1, n):
                s = target - nums[i] - nums[j]
                left, right = j+1, n-1
                while left < right:
                    if nums[left] + nums[right] == s:
                        res.add((nums[i], nums[j], nums[left], nums[right]))
                        left += 1
                        right -= 1
                    elif nums[left] + nums[right] > s:
                        right -= 1
                    else:
                        left += 1
        return res
```

42. Trapping Rain Water

```
class Solution:
    def trap(self, height: List[int]) -> int:
        left, right = 0, len(height)-1
        max_left, max_right = height[0], height[-1]
        res = 0
        while left <= right:
            # after each loop, we will process the current
            # position and go on
            if height[left] <= height[right]:
                if height[left] >= max_left:
                    max_left = height[left]
                else:
                    res += max_left - height[left]
                left += 1
            else:
                if height[right] >= max_right:
                    max_right = height[right]
                else:
                    res += max_right - height[right]
                right -= 1
        return res
```

75. Sort Colors

```python
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)
        left, right = 0, n-1
        cur = 0
        # 双指针，把0和2分别弄到左边和右边去，cur为当前处理的位置
        while cur <= right:
            val = nums[cur]
            if val == 0:
                nums[cur], nums[left] = nums[left], nums[cur]
                cur += 1
                left += 1
            elif val == 2:
                nums[cur], nums[right] = nums[right], nums[cur]
                right -= 1
            elif val == 1:
                cur += 1
```

167. Two Sum II - Input Array Is Sorted

```
class Solution:
    def twoSum(self, numbers: List[int], target: int):
        head, tail = 0, len(numbers)-1
        while head < tail:
            val = numbers[head] + numbers[tail]
            if val == target:
                return [head+1, tail+1]
            elif val > target:
                tail -= 1
            elif val < target:
                head += 1
```

345. Reverse Vowels of a String

```python
class Solution:
    def reverseVowels(self, s: str) -> str:
        v = list("aeiouAEIOU")
        left, right = 0, len(s)-1
        while left < right:
            while left < right and s[left] not in v:
                left += 1
            while left < right and s[right] not in v:
                right -= 1
            # 为什么要left!=right，bad case: "a."，字符串变长了
            if left != right and s[left] in v and s[right] in v:
                s = (
                    s[:left]
                    + s[right]
                    + s[left+1:right]
                    + s[left]
                    + s[right+1:]
                )
            left += 1
            right -= 1
        return s
```

680. Valid Palindrome II

```
class Solution:
    def validPalindrome(self, s: str) -> bool:
        left, right = 0, len(s)-1
        while left < right:
            if s[left] != s[right]:
                return (
                    self.check(s[left+1:right+1])
                    or self.check(s[left:right])
                )
            left += 1
            right -= 1
        return True
    def check(self, s):
        left, right = 0, len(s)-1
        while left < right:
            if s[left] != s[right]:
                return False
            left += 1
            right -= 1
        return True
```

977. Squares of a Sorted Array

```
class Solution:
    def sortedSquares(self, nums: List[int]) -> List[int]:
        start, end = 0, len(nums)-1
        res = []
        while start < end:
            a, b = nums[start] ** 2, nums[end] ** 2
            if a < b:
                res.insert(0, b)
                end -= 1
            else:
                res.insert(0, a)
                start += 1
        res.insert(0, nums[start] ** 2)
        return res
```

2330. Valid Palindrome IV

```python
class Solution:
    def makePalindrome(self, s: str) -> bool:
        l, r = 0, len(s) - 1
        cnt = 0
        while l < r:
            if s[l] != s[r]:
                cnt += 1
                if cnt > 2:
                    return False
            l += 1
            r -= 1
        return True
```

滑动窗口

3. Longest Substring Without Repeating Characters

```python
class Solution:
    # 其实就是一个队列,比如例题中的 abcabcbb，进入这个队列（窗口）为 abc
    # 满足题目要求，当再进入 a，队列变成了 abca，这时候不满足要求。
    # 所以，我们要移动这个队列!
    def lengthOfLongestSubstring(self, s: str) -> int:
        left, res, count = 0, 0, 0
        nums = set()
        for i in s:
            count += 1
            while i in nums:
                nums.remove(s[left])
                left += 1
                count -= 1
            res = max(res, count)
            # no matter what happens, we always add new char to set
            nums.add(i)
        return res
```

76. Minimum Window Substring

```python
class Solution:
    def minWindow(self, s: str, t: str) -> str:

        target_hash = defaultdict(int)
        roll_hash = defaultdict(int)
        left, right = 0, 0
        cnt = 0 # 记录需要的总长度
        res = ""

        # 把t中的字符加入到target中
        for c in t:
            target_hash[c] += 1

        while right < len(s):
            # 更新roll_hash
            roll_hash[s[right]] += 1
            if roll_hash[s[right]] <= target_hash[s[right]]:
                # 说明这个字符需要被算到t的某一个字母
                cnt += 1

            while (
                left <= right
                and roll_hash[s[left]] > target_hash[s[left]]
            ):
                # left <= right is for bad case: "b" "a"
                roll_hash[s[left]] -= 1
                left += 1

            if cnt == len(t):
                if res == "" or right-left+1 < len(res):
                    res = s[left:right+1]

            right += 1
        return res
```

209. Minimum Size Subarray Sum

```python
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        left, right = 0, 0
        res = float("inf")
        val = 0
        while right < len(nums):
            val += nums[right]
            if val >= target:
                while left < right and val - nums[left] >= target:
                    val -= nums[left]
                    left += 1
                res = min(res, right - left + 1)
            right += 1
        return 0 if res == float("inf") else res
```

340. Longest Substring with At Most K Distinct Characters

```
class Solution:
    def lengthOfLongestSubstringKDistinct(self, s: str, k: int) -> int:
        if k == 0:
            return 0
        res = 0
        d = defaultdict(int)
        # 记录当前非零的数量
        cnt_none_zero = 0

        left, right = 0, 0
        while right < len(s):
            c = s[right]
            if d[c] == 0:
                cnt_none_zero += 1
                if cnt_none_zero > k:
                    while left < right:
                        c_left = s[left]
                        d[c_left] -= 1
                        # left一定修改放在break前面
                        # 否则left指针少动一次
                        left += 1
                        # c_left != c是因为减少的这个数量一定要
                        # 是其他字母的，自己的当前肯定至少有right位置的1个
                        if d[c_left] == 0 and c_left != c:
                            break
            d[c] += 1
            length = right - left + 1
            if length > res:
                res = length
            right += 1
        return res
```

438. Find All Anagrams in a String

```
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:

        p_dict = defaultdict(int)
        for c in p:
            p_dict[c] += 1
        if len(s) < len(p):
            return []
        s_dict = defaultdict(int)
        for c in s[:len(p)]:
            s_dict[c] += 1

        def is_anagram(s_dict, p_dict):
            non_zero = 0
            for c in s_dict:
                if s_dict[c] == 0:
                    # 由于drop，会出现零次的，不能考虑
                    continue
                else:
                    if c not in p_dict or s_dict[c] != p_dict[c]:
                        return False
                    non_zero += 1
            return non_zero == len(p_dict)

        res = []
        if is_anagram(s_dict, p_dict):
            res.append(0)

        for i in range(len(p), len(s)):
            new = s[i]
            drop = s[i-len(p)]
            s_dict[new] += 1
            s_dict[drop] -= 1
            if is_anagram(s_dict, p_dict):
                res.append(i-len(p)+1)

        return res
```

487. Max Consecutive Ones II

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        n_zeros = 0
        left, right = 0, 0
        res = 0
        while right < len(nums):
            # 每次right都往前一个，尽可能不缩小左边
            cur = nums[right]
            if cur == 0:
                n_zeros += 1
            while n_zeros > 1 and left < right:
                if nums[left] == 0:
                    n_zeros -= 1
                left += 1
            res = max(res, right-left+1)
            right += 1
        return res
```

1004. Max Consecutive Ones III

```python
class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:
        # 问题转化为当把0看成1时，最长连续1的个数
        # 可以用滑动窗口，右边每次移动一个（记录是不是0）
        # 左边如果不满足（k个0用完了）则缩进（如果是0则记得更新zeros）
        n = len(nums)
        left, right = 0, 0
        res = 0
        zeros = 0
        while right < n:
            if nums[right] == 0:
                zeros += 1
            while zeros > k:
                if nums[left] == 0:
                    zeros -= 1
                left += 1
            res = max(res, right-left+1)
            right += 1
        return res
```

二分查找

数组查找

4. Median of Two Sorted Arrays

```
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> f
        # really tough question

        if len(nums2) == 0:
            len1 = len(nums1)
            if len1 % 2 == 1:
                return nums1[len1//2]
            else:
                return (nums1[len1//2] + nums1[len1//2 - 1]) / 2
        n = len(nums1) + len(nums2)
        k = n // 2

        # for O(log(min(len(nums1), len(nums2))))
        if len(nums1) > len(nums2):
            self.findMedianSortedArrays(nums2, nums1)

        l, r = 0, len(nums1)-1
        while True:
            # i for nums1's pos, j for nums2's pos
            i = (l + r) // 2
            j = k - i - 2 # minus 2 since i, j are index from 0

            left1 = nums1[i] if i >= 0 else float("-inf")
            right1 = nums1[i+1] if i+1 < len(nums1) else float("inf")
            left2 = nums2[j] if j >= 0 else float("-inf")
            right2 = nums2[j+1] if j+1 < len(nums2) else float("inf")

            if left1 <= right2 and left2 <= right1:
                if n % 2 == 1:
                    return min(right1, right2)
                else:
                    return (max(left1, left2) + min(right1, right2)) / 2
            # the left part of nums1 has too much element and we
            # need narrow the right bound
            elif left1 > right2:
                r -= 1
            elif left2 > right1:
                l += 1
```

33. Search in Rotated Sorted Array

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1
        while left < right:
            mid = left + (right - left) // 2
            if nums[mid] >= nums[left]: # in the left part
                if target >= nums[left] and nums[mid] >= target: # left is ordered
                    right = mid
                else: # not ordered
                    left = mid + 1 # target at least after mid
            else: # in the right part
                if target <= nums[right] and nums[mid] <= target: # right is order
                    left = mid # search
                else:
                    right = mid - 1 # target at least before mid
        return -1 if nums[left] != target else left
```

34. Find First and Last Position of Element in Sorted Array

```
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        if len(nums) == 0:
            return [-1, -1]
        left, right = 0, len(nums)-1
        while left < right:
            mid = left + (right - left) // 2
            # only move left when necessary
            if nums[mid] < target:
                left = mid + 1
            else:
                right = mid
        first = -1 if nums[left] != target else left
        left, right = 0, len(nums)-1
        while left < right:
            mid = right - (right - left) // 2
            if nums[mid] > target:
                right = mid - 1
            else:
                left = mid
        last = -1 if nums[left] != target else left
        return [first, last]
```

35. Search Insert Position

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        if target > nums[-1]:
            return len(nums)
        left, right = 0, len(nums)-1
        while left < right:
            mid = left + (right - left) // 2
            if nums[mid] >= target:
                right = mid
            else:
                left = mid + 1
        return left
```

81. Search in Rotated Sorted Array II

```python
class Solution:
    def search(self, nums: List[int], target: int) -> bool:
        left, right = 0, len(nums)-1
        while left < right:

            # if judgement is omitted, [1,0,1,1,1] will fail
            # because we don't know the ordered side given left=right=mid=1
            while left < right and nums[left] == nums[left+1]:
                left += 1
            while left < right and nums[right] == nums[right-1]:
                right -= 1
            # since left and right are moved since duplicates
            # so we should judge the result first here
            # othersize, find 0 in [1,1] will fail
            if left == right:
                return nums[left] == target

            mid = left + (right - left) // 2
            if nums[mid] == target:
                return True
            if nums[mid] >= nums[left]:
                if target >= nums[left] and nums[mid] >= target:
                    right = mid
                else:
                    left = mid + 1
            else:
                if target >= nums[mid] and nums[right] >= target:
                    left = mid
                else:
                    right = mid - 1
        return nums[left] == target
```

153. Find Minimum in Rotated Sorted Array

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        left, right = 0, len(nums)-1
        while left < right:
            mid = left + (right - left) // 2
            if nums[mid] > nums[right]:
                left = mid + 1
            else:
                right = mid
        return nums[left]
```

154. Find Minimum in Rotated Sorted Array II

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        left, right = 0, len(nums)-1
        while left < right:
            while left < right and nums[left] == nums[left+1]:
                left += 1
            while left < right and nums[right] == nums[right-1]:
                right -= 1
            if left == right:
                return nums[left]
            mid = left + (right - left) // 2
            # if we don't contain this, than
            # [1,3,5] will fail
            if nums[left] < nums[right]:
                return nums[left]
            # when we can to this step, it means
            # array is not strictly ascending
            if nums[mid] >= nums[left]:
                left = mid + 1
            else:
                right = mid
        return nums[left]

"""
# a better solution from china site
int findMini(vector<int>& nums){
    /*
    -思路-
    旋转数组肯定将nums分为两个有序的数组，分为别nums1和nums2
    并且假设nums1中的元素均大于等于nums2中的元素
    那么我们要求的元素就是nums2的首元素

    --步骤--
    如果nums[mid] > nums[right],说明此时的mid严格的在nums1当中
    那么nums2的首元素设为i的话，就应当是：mid < i <= right。
    取left = mid + 1;
    如果nums[mid] < nums[right],说明此时的mid严格的在nums2当中
    也就是：mid <= i < right
    取right = mid
    如果nums[mid] == nums[right],细分为三种情况。
        情况一：[1,1,1,1,1,1,1,1]
        情况二：[4,5,6,7,1,1,1,1,1,1]
        情况二：[4,5,6,7,0,1,1,1,1,1]
    取right--便可
```

```
    */

    int left = 0,right = nums.size() - 1,mid;
    while (left <= right)
    {
        mid = left + (right - left) / 2;
        if(nums[mid] > nums[right]) left = mid + 1;
        else if(nums[mid] < nums[right]) right = mid;
        else --right;
    }
    //最终退出的时候事left = right + 1;
    return nums[left];
}
"""
```

162. Find Peak Element

```python
class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        # 如果nums[k] >= nums[k+1] 说明它或它的左边一定有解
        left, right = 0, len(nums)-1
        while left < right:
            mid = left + (right - left) // 2
            # mid不可能是right，因为是向下取整的
            # 如果没有取整，根本进不了循环
            if nums[mid] >= nums[mid+1]:
                right = mid
            else:
                left = mid + 1
        return left
```

540. Single Element in a Sorted Array

```python
class Solution:
    def singleNonDuplicate(self, nums: List[int]) -> int:

        if len(nums) == 1:
            return nums[0]

        def check(i):
            if i == 0:
                if nums[1] != nums[0]:
                    return 0
                else:
                    return -1
            elif i == len(nums)-1:
                if nums[-1] != nums[-2]:
                    return 0
                else:
                    return 1
            elif nums[i] == nums[i+1]:
                if i%2==0:
                    return -1
                else:
                    return 1
            elif nums[i] == nums[i-1]:
                if i%2==1:
                    return -1
                else:
                    return 1

        left, right = 0, len(nums) - 1
        while left < right:
            mid = left + (right - left) // 2
            val = check(mid)
            if val == 1:
                # 相当于 nums[mid] > x
                right = mid
            elif val == -1:
                # 相当于 nums[mid] < x
                left = mid + 1
            else:
                return nums[mid]
        return nums[left]
```

矩阵查找

74. Search a 2D Matrix

```python
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:

        # map the matrix idx to a array idx

        r, c = len(matrix), len(matrix[0])
        left, right = 0, r*c-1
        while left < right:
            mid = left + (right - left) // 2
            mid_row, mid_col = self.map_val(mid, r, c)
            if matrix[mid_row][mid_col] < target:
                left = mid + 1
            else:
                right = mid
        left_row, left_col = self.map_val(left, r, c)
        return matrix[left_row][left_col] == target

    def map_val(self, n, r, c):
        row = n // c
        col = n % c
        return row, col
```

240. Search a 2D Matrix II

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        # search from the right corner
        x, y = 0, len(matrix[0])-1
        while x <= len(matrix)-1 and y >= 0:
            if matrix[x][y] < target:
                x += 1
            elif matrix[x][y] > target:
                y -= 1
            else:
                return True
        return False
```

378. Kth Smallest Element in a Sorted Matrix

```python
class Solution:
    def kthSmallest(self, matrix: List[List[int]], k: int) -> int:
        # 这个方法的精髓在于随着左边右边的调整，自动二分出来就是最后的答案
        self.matrix = matrix
        self.m, self.n = len(matrix), len(matrix[0])
        left, right = matrix[0][0], matrix[-1][-1]
        while left < right:
            mid = left + (right - left) // 2
            count = self.count_less_than_mid(mid)
            if count < k:
                left = mid + 1
            else:
                right = mid
        return left
    def count_less_than_mid(self, mid):
        i, j = self.m-1, 0
        count = 0
        while i >= 0 and j <= self.n-1:
            if self.matrix[i][j] <= mid:
                # 只在小于等于时候统计count，然后往右走
                count += i+1
                j += 1
            else:
                # 这个时候说明要往上走
                i -= 1
        return count
```

668. Kth Smallest Number in Multiplication Table

```python
class Solution:
    def findKthNumber(self, m: int, n: int, k: int) -> int:
        left, right = 1, m*n
        while left < right:
            mid = left + (right - left) // 2
            val = self.count(m, n, mid)
            if val >= k:
                right = mid
            else:
                left = mid + 1
        return left
    def count(self, m, n, s):
        # 数一数一共有几个元素小于等于s
        return sum(min(s//i, n) for i in range(1, m+1))
```

抽象查找

69. Sqrt(x)

```
class Solution:
    def mySqrt(self, x: int) -> int:
        left, right = 0, x
        while left < right:
            mid = right - (right - left) // 2
            if mid * mid <= x:
                left = mid
            else:
                right = mid - 1
        return left
```

275. H-Index II

```python
class Solution:
    def hIndex(self, citations: List[int]) -> int:
        # 背模板!
        """
        [0,1,3,5,6]
        [1,2,100]
        [0]
        [100]
        [0,0,4,4]
        """
        # [----*####] 找左边界
        left, right = 0, len(citations) - 1
        while left - right:
            mid = left + (right - left) // 2
            val = citations[mid]
            count = len(citations) - mid
            if val < count:
                left = mid + 1
            else:
                right = mid
        # bad case: [0] ([100])
        if citations[left] >= len(citations) - left:
            return len(citations) - left
        else:
            return 0
```

278. First Bad Version

```
# The isBadVersion API is already defined for you.
# def isBadVersion(version: int) -> bool:

class Solution:
    def firstBadVersion(self, n: int) -> int:
        left, right = 1, n
        while left < right:
            mid = left + (right - left) // 2
            if isBadVersion(mid):
                right = mid
            else:
                left = mid + 1
        return left
```

367. Valid Perfect Square

```
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        left, right = 1, num
        while left < right:
            mid = right - (right - left) // 2
            val = mid * mid
            if val == num:
                return True
            elif val < num:
                left = mid
            else:
                right = mid - 1
        return left * left == num
```

374. Guess Number Higher or Lower

```
# The guess API is already defined for you.
# @param num, your guess
# @return -1 if num is higher than the picked number
# 1 if num is lower than the picked number
# otherwise return 0
# def guess(num: int) -> int:

class Solution:
    def guessNumber(self, n: int) -> int:
        left, right = 1, n
        while left < right:
            mid = left + (right - left) // 2
            val = guess(mid)
            if val == -1:
                right = mid
            elif val == 1:
                left = mid + 1
            else:
                return mid
        return left
```

410. Split Array Largest Sum

```python
class Solution:
    def splitArray(self, nums: List[int], m: int) -> int:
        self.m = m
        self.nums = nums
        left, right = max(nums), sum(nums)
        while left < right:
            mid = left + (right - left) // 2
            if self.check(self.nums, mid):
                right = mid
            else:
                left = mid + 1
        return left
    def check(self, arr, n):
        count, s = 1, 0
        for val in arr:
            if s + val > n:
                count += 1
                s = val
            else:
                s += val
        return count <= self.m
"""
class Solution:
    def splitArray(self, nums: List[int], m: int) -> int:
        pass
        # 动态规划：O(n^2)
        # dp[i][j]表示前i+1个元素分成j组对一个的最小可能值
        # dp[i][j] = min(dp[k][j-1] + sum(nums[k+1:i]) for k in range(0, i-1))
        acc = list(accumulate(nums)) # 前缀和
        dp = [[float("inf") for j in range(m+1)] for i in range(len(nums))]
        for i in range(len(nums)):
            for j in range(1, min(i+2, m+1)):
                if j == 1:
                    dp[i][j] = acc[i]
                else:
                    dp[i][j] = min(max(dp[k][j-1], acc[i] - acc[k]) for k in range(i))
        return dp[-1][-1]
"""
```

441. Arranging Coins

```
class Solution:
    def arrangeCoins(self, n: int) -> int:
        left, right = 1, n
        while left < right:
            mid = right - (right - left) // 2
            val = (mid * (mid + 1)) / 2
            if val <= n:
                left = mid
            else:
                right = mid - 1
        return left
```

540. Single Element in a Sorted Array

```
class Solution:
    def findClosestElements(self, arr: List[int], k: int, x: int):
        # a和b是左边界的最左可能和最右可能
        a, b = 0, len(arr)-k
        while a < b:
            mid = a + (b-a) // 2
            if x - arr[mid] > arr[mid+k] - x:
                a = mid + 1
            else:
                b = mid
        return arr[a: a+k]
```

719. Find K-th Smallest Pair Distance

```python
class Solution:
    def smallestDistancePair(self, nums: List[int], k: int) -> int:
        def count(mid: int) -> int:
            # 计算所有距离小于等于mid的数对数目cnt
            cnt = 0
            left = 0
            for right, num in enumerate(nums):
                while num - nums[left] > mid:
                    left += 1
                cnt += right - left
            return cnt

        nums.sort()
        # 距离最小为0，最大为max−min，使用二分
        # 每个二分内部，用双指针来计算小于等于当前距离的有多少对
        left, right = 0, nums[-1] - nums[0]
        while left < right:
            mid = left + (right - left) // 2
            if count(mid) < k:
                left = mid + 1
            else:
                right = mid
        return left
```

1146. Snapshot Array

```python
class SnapshotArray:

    def __init__(self, length: int):
        # 每一个元素都是一个字典，是快照号码到值的映射
        self.d = [{0: 0} for i in range(length)]
        self.s_id = 0

    def set(self, index: int, val: int) -> None:
        self.d[index][self.s_id] = val

    def snap(self) -> int:
        self.s_id += 1
        return self.s_id - 1 # 实际上记录的比快照次数多1

    def get(self, index: int, snap_id: int) -> int:
        if snap_id in self.d[index]:
            return self.d[index][snap_id]
        arr = list(self.d[index].keys())
        # 找最大小于目标值的元素，找左用右
        left, right = 0, len(arr)-1
        while left < right:
            mid = right - (right - left) // 2
            if arr[mid] >= snap_id:
                right = mid - 1
            else:
                left = mid
        return self.d[index][arr[left]]
```

1891. Cutting Ribbons

```python
class Solution:
    def maxLength(self, ribbons: List[int], k: int) -> int:
        # 二分搜索可能的分段值
        left, right = 0, max(ribbons)
        while left < right:
            # 因为左侧是合法的，所以要配向上取整
            mid = right - (right - left) // 2
            count = 0
            for r in ribbons:
                count += r // mid
            if count >= k:
                left = mid
            else:
                right = mid - 1
        return left
```

树

递归

100. Same Tree

```
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]):
        if p == None and q != None:
            return False
        if p != None and q == None:
            return False
        if p == None and q == None:
            return True
        return p.val == q.val
            and self.isSameTree(p.left, q.left)
            and self.isSameTree(p.right, q.right)
```

101. Symmetric Tree

```
class Solution:
    def isSymmetric(self, root: Optional[TreeNode]) -> bool:

        def helper(ra, rb):
            if ra == None and rb == None:
                return True
            elif ra != None and rb == None:
                return False
            elif ra == None and rb != None:
                return False
            elif ra.val != rb.val:
                return False
            else:
                return helper(ra.left, rb.right) and helper(ra.right, rb.left)

        return helper(root.left, root.right)
```

104. Maximum Depth of Binary Tree

```
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        return self.dfs(root, 0)
    def dfs(self, root, depth):
        if root == None:
            return depth
        return max(self.dfs(root.left, depth+1), self.dfs(root.right, depth+1)
```

110. Balanced Binary Tree

```
class Solution:
    def isBalanced(self, root: Optional[TreeNode]) -> bool:
        def helper(root):
            # this function return current height and current status
            if root == None:
                return 0, True
            left_h, left_isbalanced = helper(root.left)
            right_h, right_isbalanced = helper(root.right)
            # update current height
            h = max(left_h, right_h) + 1
            # 1) left or right not balanced
            # 2) both balanced but height diff >= 2
            if (not left_isbalanced) or (not right_isbalanced):
                return h, False
            if abs(left_h - right_h) >= 2:
                return h, False
            else:
                return h, True
        return helper(root)[1]
```

111. Minimum Depth of Binary Tree

```
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        if root == None:
            return 0
        if root.left == None:
            return self.minDepth(root.right) + 1
        if root.right == None:
            return self.minDepth(root.left) + 1
        return min(self.minDepth(root.left), self.minDepth(root.right)) + 1
```

112. Path Sum

```python
# 给定targetSum，返回是不是有路径
class Solution:
    def hasPathSum(self, root: Optional[TreeNode], targetSum: int):

        def helper(root, val):
            if root == None:
                return False
            # 由于叶子节点可以是负的，因此不能用这个判断False
            # elif root.val > val:
            # return False
            elif (
                root.val == val
                and root.left == None
                and root.right == None
            ):
                return True
            else:
                return (
                    helper(root.left, val-root.val)
                    or helper(root.right, val-root.val
                )

        return helper(root, targetSum)
```

113. Path Sum II

```
# 给定和，找到所有路径
class Solution:
    def pathSum(self, root: Optional[TreeNode], targetSum: int):

        self.res = []

        def helper(root, val, res):
            if root == None:
                return
            elif (
                root.val == val
                and root.left == None
                and root.right == None
            ):
                # 一定要注意，哪个分支加入的，哪个分支就需要归还（pop）
                res.append(root.val)
                self.res.append(res.copy())
                res.pop(-1)
            else:
                res.append(root.val)
                helper(root.left, val-root.val, res)
                helper(root.right, val-root.val, res)
                res.pop(-1)

        helper(root, targetSum, [])
        return self.res
```

124. Binary Tree Maximum Path Sum

```python
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
    def maxPathSum(self, root: Optional[TreeNode]) -> int:
        if root == None:
            return 0
        self.res = float("-inf")
        def dfs(node):
            if node == None:
                return 0
            # 左边的可能的最大值
            left = max(dfs(node.left), 0)
            # 右边的可能的最大值
            right = max(dfs(node.right), 0)
            # 记录当前节点（事实上，答案等价于遍历所有点指标的最大值
            # ，指标是经过这一点的最大值）
            self.res = max(self.res, node.val+left+right)
            # 左边大传左边，右边大传右边
            return max(node.val+left, node.val+right)
        dfs(root)
        return self.res
```

129. Sum Root to Leaf Numbers

```
class Solution:
    def sumNumbers(self, root: Optional[TreeNode]) -> int:
        self.res = 0
        def helper(root, n):
            if root.left == None and root.right == None:
                self.res += n
                return
            elif root.left == None and root.right != None:
                helper(root.right, n*10+root.right.val)
                return
            elif root.left != None and root.right == None:
                helper(root.left, n*10+root.left.val)
                return
            else:
                helper(root.left, n*10+root.left.val)
                helper(root.right, n*10+root.right.val)
        helper(root, root.val)
        return self.res
```

144. Binary Tree Preorder Traversal

```
class Solution:
    def preorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        self.res = []
        def dfs(root):
            if root is None:
                return
            self.res.append(root.val)
            dfs(root.left)
            dfs(root.right)
        dfs(root)
        return self.res
```

156. Binary Tree Upside Down

```
class Solution:
    def upsideDownBinaryTree(self, root: Optional[TreeNode]):
        # why the right node should have no children?
        # if so, the right's left node and its sibling will all be the head
        # and if right have no sibling, then the tree will have no head
        """
        the only possible structure of the original tree is like:
             ....
            * *
           * *
          * *
         * *
        * *
        """
        # root == None to handle the null input
        if root == None or (root.left == None and root.right == None):
            return root
        left = self.upsideDownBinaryTree(root.left)
        """
        # why these 2 lines are wrong?
        # the left is only for passing the final root
        # we are still handling the current level
        # namely, we should use root.left to get the actual level
        left.left = root.right
        left.right = root
        """
        root.left.left = root.right
        root.left.right = root
        # if we don't set them to null
        # then will generate a circle in tree
        root.left = None
        root.right = None
        return left
```

222. Count Complete Tree Nodes

```
# Integer.toBinaryString
class Solution:
    def countNodes(self, root: Optional[TreeNode]) -> int:
        # 先把高度拿到，然后最后一层用二进制代表每一个节点
        # 二进制数字除了最高维外，代表向左还是向右，此时
        # 根据结果是None还是不是None来进行二分搜索，找到最后一个不是None的

        def get_h(root, h):
            if root is None:
                return h
            return get_h(root.left, h+1)

        def search(root, b):
            if len(b) == 1:
                if b == "0":
                    return root.left
                else:
                    return root.right
            else:
                if b[0] == "0":
                    return search(root.left, b[1:]) # 不要忘记return
                else:
                    return search(root.right, b[1:])

        h = get_h(root, 0)
        if h == 0:
            return 0
        full_nodes = 2 ** h - 1
        left, right = 2 ** (h-1), full_nodes
        while left < right:
            mid = right - (right - left) // 2
            b_mid = bin(mid)[3:]
            if search(root, b_mid) is None:
                right = mid - 1
            else:
                left = mid
        return left # 注意不是返回mid，是left or right
```

226. Invert Binary Tree

```
class Solution:
    def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
        if root is None:
            return None
        temp = root.left
        root.left = root.right
        root.right = temp
        self.invertTree(root.left)
        self.invertTree(root.right)
        return root
```

257. Binary Tree Paths

```
class Solution:
    def binaryTreePaths(self, root: Optional[TreeNode]) -> List[str]:
        self.res = []
        self.dfs(root, [])
        return self.res
    def dfs(self, root, arr):
        if root == None:
            return
        arr.append(str(root.val))
        if root.left == None and root.right == None:
            self.res.append("->".join(arr))
            arr.pop(-1)
            # why we need pop here?
            # actually the pop of last line is the last operation
            # for a node, but for the leaf, we return here, thus
            # it's impossible to get arr.pop(−1) at last line, so
            # we need pop the val before return
            return
        self.dfs(root.left, arr)
        self.dfs(root.right, arr)
        arr.pop(-1)
```

366. Find Leaves of Binary Tree

```
class Solution:
    def findLeaves(self, root: Optional[TreeNode]) -> List[List[int]]:
        # 如果把最外层的叶子level看成0
        # 那么实际上最上面的root level就是树的最大深度
        res = []
        def dfs(root):
            if root is None:
                return -1
            level = max(dfs(root.left), dfs(root.right)) + 1
            if level >= len(res):
                res.append([])
            res[level].append(root.val)
            return level
        dfs(root)
        return res
```

437. Path Sum III

```
class Solution:
    def pathSum(self, root: Optional[TreeNode], targetSum: int) -> int:

        # 另外还有前缀和做法，每个节点修正为之前所有点之和
        # 最后重新遍历一次把targetSum的点找出来

        self.t = 0
        self.targetSum = targetSum
        self.count = 0

        def search(root):
            if root is None:
                return
            self.t += root.val
            if self.t == self.targetSum:
                self.count += 1
            search(root.left)
            search(root.right)
            self.t -= root.val

        def dfs(root):
            if root is None:
                return
            search(root)
            dfs(root.left)
            dfs(root.right)

        dfs(root)
        return self.count
```

508. Most Frequent Subtree Sum

```
class Solution:
    def findFrequentTreeSum(self, root: Optional[TreeNode]) -> List[int]:
        # 后序遍历
        self.val_to_freq = defaultdict(int)
        def dfs(root):
            if root is None:
                return 0
            left = dfs(root.left)
            right = dfs(root.right)
            val = left + right + root.val
            self.val_to_freq[val] += 1
            return val
        if root is None:
            return []
        dfs(root)
        max_val = max(self.val_to_freq.values())
        return list(filter(
            lambda x: self.val_to_freq[x]==max_val, self.val_to_freq))
```

543. Diameter of Binary Tree

```
class Solution:
    def diameterOfBinaryTree(self, root: Optional[TreeNode]) -> int:
        self.res = 0
        def dfs(root):
            if root is None:
                return 0
            if root.left is None:
                if root.right is None:
                    return 0
                else:
                    val = dfs(root.right)+1
                    self.res = max(self.res, val)
                    return val
            if root.right is None:
                if root.left is None:
                    return 0
                else:
                    val = dfs(root.left)+1
                    self.res = max(self.res, val)
                    return val
            left_val = dfs(root.left)+1
            right_val = dfs(root.right)+1
            self.res = max(self.res, left_val+right_val)
            return max(left_val, right_val)
        dfs(root)
        return self.res
```

545. Boundary of Binary Tree

```
class Solution:
    def boundaryOfBinaryTree(self, root: Optional[TreeNode]) -> List[int]:
        # 先装左边界，然后叶子，最后有边界，拆三个函数，但注意append元素的顺序
        def get_left_bound(node):
            if (not node.left) and (not node.right):
                # 叶子
                return
            res.append(node.val)
            if node.left:
                get_left_bound(node.left)
            else:
                # 注意这里是if else的关系，而不是if node.right
                # If a node is in the left boundary, has no left child,
                # but has a right child, then the right child is in the left boundary.
                get_left_bound(node.right)

        def get_right_bound(node):
            if (not node.left) and (not node.right):
                return
            if node.right:
                get_right_bound(node.right)
            else:
                get_right_bound(node.left)
            res.append(node.val)

        def get_leaves(node):
            if (not node.left) and (not node.right):
                res.append(node.val)
            if node.left:
                get_leaves(node.left)
            if node.right:
                get_leaves(node.right)

        res = [root.val]
        if root.left:
            get_left_bound(root.left)
            get_leaves(root.left)
        if root.right:
            get_leaves(root.right)
            get_right_bound(root.right)
        return res
```

563. Binary Tree Tilt

```
class Solution:
    def findTilt(self, root: Optional[TreeNode]) -> int:
        self.s = 0
        def dfs(root):
            if root is None:
                return 0
            left = dfs(root.left)
            right = dfs(root.right)
            self.s += abs(left - right)
            return left + right + root.val
        dfs(root)
        return self.s
```

572. Subtree of Another Tree

```python
class Solution:
    def isSubtree(self, root, subRoot) -> bool:
        def compare(r, s):
            if r is None and s is None:
                return True
            elif r is None and s is not None:
                return False
            elif r is not None and s is None:
                return False
            if r.val != s.val:
                return False
            return compare(r.left, s.left) & compare(r.right, s.right)
        self.flag = False
        def dfs(r):
            # we should check subRoot is None later
            if r is None:
                return
            if self.flag:
                return
            self.flag = compare(r, subRoot)
            dfs(r.left)
            dfs(r.right)
        dfs(root)
        return self.flag or (root is None and subRoot is None)
```

617. Merge Two Binary Trees

```
class Solution:
    def mergeTrees(self, root1, root2):

        def dfs(r1, r2):
            if r1 is None and r2 is None:
                return None
            elif r1 is None and r2 is not None:
                cur = TreeNode(r2.val)
                cur.left = dfs(r1, r2.left)
                cur.right = dfs(r1, r2.right)
            elif r1 is not None and r2 is None:
                cur = TreeNode(r1.val)
                cur.left = dfs(r1.left, r2)
                cur.right = dfs(r1.right, r2)
            else:
                cur = TreeNode(r1.val + r2.val)
                cur.left = dfs(r1.left, r2.left)
                cur.right = dfs(r1.right, r2.right)
            return cur
        return dfs(root1, root2)
```

654. Maximum Binary Tree

```
# java不用传切片，直接nums+两个索引
class Solution:
    def constructMaximumBinaryTree(self, nums: List[int]):
        if not nums:
            return None
        val = max(nums)
        node = TreeNode(val)
        idx = nums.index(val)
        node.left = self.constructMaximumBinaryTree(nums[0:idx])
        node.right = self.constructMaximumBinaryTree(nums[idx+1:])
        return node
```

663. Equal Tree Partition

```python
class Solution:
    def checkEqualTree(self, root: Optional[TreeNode]) -> bool:
        # 把总和拿到，看一半是不是某个节点的子树和
        s = set()
        def dfs(r):
            if r is None:
                return 0
            left = dfs(r.left)
            right = dfs(r.right)
            val = left + right + r.val
            # 如果断掉的话，相当于当前节点是一个孩子，头结点不可能是孩子
            # bad case: [0,-1,1]
            if r != root:
                s.add(val)
            return val
        return dfs(root) / 2 in s
```

687. Longest Univalue Path

```
class Solution:
    def longestUnivaluePath(self, root: Optional[TreeNode]) -> int:
        self.res = 0
        self.dfs(root)
        return self.res
    def dfs(self, root):
        if root is None:
            return 0
        val = root.val
        left_res = self.dfs(root.left)
        right_res = self.dfs(root.right)
        if root.left and root.left.val == val:
            left_res += 1
        else:
            left_res = 0
        if root.right and root.right.val == val:
            right_res += 1
        else:
            right_res = 0
        self.res = max(self.res, left_res + right_res)
        return max(left_res, right_res)
```

938. Range Sum of BST

```
class Solution:
    def rangeSumBST(self, root: Optional[TreeNode], low: int, high: int):
        self.res = 0
        def dfs(node):
            if node is None:
                return
            if low <= node.val <= high:
                self.res += node.val
            dfs(node.left)
            dfs(node.right)
        dfs(root)
        return self.res
```

1376-time-needed-to-inform-all-employees.tex

```
class Node:
    def __init__(self):
        self.child = []
        self.time = 0
class Solution:
    def numOfMinutes(self, n: int, headID: int, manager: List[int], informTim
        # 多叉树的最长路径
        nodes = [Node() for _ in range(n)]
        for i, father in enumerate(manager):
            node = nodes[i]
            node.time = informTime[i]
            if father != -1:
                nodes[father].child.append(node)
        def dfs(root):
            child = root.child
            if child:
                return root.time + max(dfs(i) for i in child)
            else:
                return root.time
        return dfs(nodes[headID])
```

1448. Count Good Nodes in Binary Tree

```python
class Solution:
    def goodNodes(self, root: TreeNode) -> int:
        # 把上面的最大值信息随着路径放下去即可

        def dfs(root, val):
            if root is None:
                return
            if root.val >= val:
                self.res += 1
            dfs(root.left, max(val, root.val))
            dfs(root.right, max(val, root.val))

        self.res = 0
        dfs(root, float("-inf"))
        return self.res
```

1628. Design an Expression Tree With Evaluate Function

```
import abc
from abc import ABC, abstractmethod
"""
Your TreeBuilder object will be instantiated and called as such:
obj = TreeBuilder();
expTree = obj.buildTree(postfix);
ans = expTree.evaluate();
"""

class Node(ABC):
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
    def evaluate(self) -> int:
        if isinstance(self.val, int):
            return self.val
        left = self.left.evaluate()
        right = self.right.evaluate()
        if self.val == "+":
            return left + right
        if self.val == "-":
            return left - right
        if self.val == "*":
            return left * right
        if self.val == "/":
            return left // right

class TreeBuilder(object):
    def buildTree(self, postfix: List[str]) -> 'Node':
        stack = []
        for s in postfix:
            if s in "+-*/":
                node = Node(s)
                node.right = stack.pop()
                node.left = stack.pop()
                stack.append(node)
            else:
                node = Node(int(s))
                stack.append(node)
        return stack[0]
```

层序遍历

102. Binary Tree Level Order Traversal

```
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        queue = [root] if root != None else []
        res = []
        while queue:
            temp_list = []
            # 直接使用队列长度当做计数器
            for _ in range(len(queue)):
                node = queue.pop(0)
                temp_list.append(node.val)
                if node.left != None:
                    queue.append(node.left)
                if node.right != None:
                    queue.append(node.right)
            res.append(temp_list)
        return res
```

103. Binary Tree Zigzag Level Order Traversal

```
class Solution:
    def zigzagLevelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        queue = [root] if root != None else []
        res = []
        ordered = True
        while queue:
            temp_list = []
            q_len = len(queue)
            if ordered:
                for _ in range(q_len):
                    node = queue.pop(0)
                    temp_list.append(node.val)
                    if node.left != None:
                        queue.append(node.left)
                    if node.right != None:
                        queue.append(node.right)
            else:
                for i in range(q_len):
                    node = queue.pop(-1) # 注意这句
                    temp_list.append(node.val)
                    if node.right != None:
                        queue.insert(0, node.right)
                    if node.left != None:
                        queue.insert(0, node.left)
            ordered = bool(1-ordered)
            res.append(temp_list)
        return res
```

107. Binary Tree Level Order Traversal II

```
class Solution:
    def levelOrderBottom(self, root: Optional[TreeNode]) -> List[List[int]]:
        queue = [root] if root != None else []
        res = []
        while queue:
            temp_list = []
            # 直接使用队列长度当做计数器
            for _ in range(len(queue)):
                node = queue.pop(0)
                temp_list.append(node.val)
                if node.left != None:
                    queue.append(node.left)
                if node.right != None:
                    queue.append(node.right)
            res.append(temp_list)
        return res[::-1]
```

116. Populating Next Right Pointers in Each Node

```
class Solution:
    def connect(self, root: 'Optional[Node]') -> 'Optional[Node]':
        if root == None:
            return None
        queue = [root]
        while queue:
            for i in range(len(queue)):
                cur = queue.pop(0)
                if cur.left != None:
                    queue.append(cur.left)
                if cur.right != None:
                    queue.append(cur.right)
                if i == 0:
                    last = cur
                else:
                    last.next = cur
                    last = cur # 不要忘了更新last
            cur.next = None
        return root
```

117. Populating Next Right Pointers in Each Node II

```
class Solution:
    def connect(self, root: 'Node') -> 'Node':
        if root == None:
            return None
        queue = [root]
        while queue:
            for i in range(len(queue)):
                cur = queue.pop(0)
                if cur.left != None:
                    queue.append(cur.left)
                if cur.right != None:
                    queue.append(cur.right)
                if i == 0:
                    last = cur
                else:
                    last.next = cur
                    last = cur # 不要忘了更新last
            cur.next = None
        return root
```

199. Binary Tree Right Side View

```
class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:
        queue = [root] if root != None else []
        res = []
        while queue:
            q_len = len(queue)
            for i in range(q_len):
                node = queue.pop(0)
                if i == q_len-1:
                    res.append(node.val)
                if node.left != None:
                    queue.append(node.left)
                if node.right != None:
                    queue.append(node.right)
        return res
```

314. Binary Tree Vertical Order Traversal

```
class Solution:
    def verticalOrder(self, root: Optional[TreeNode]) -> List[List[int]]:

        # 这一题先用dfs把每个节点搞一个level属性
        # 同时把最左边和最右边能到多少给记一下
        # 然后用bfs层次遍历，根据level的值，放到结果的列表中

        self.left, self.right = 0, 0
        def dfs(root, level):
            if root is None:
                return
            self.left = min(self.left, level)
            self.right = max(self.right, level)
            root.level = level
            dfs(root.left, level-1)
            dfs(root.right, level+1)
        dfs(root, 0)
        res = [[] for _ in range(self.right-self.left+1)]
        if root is None:
            return []
        q = [root]
        while q:
            cur = q.pop(0)
            res[cur.level-self.left].append(cur.val)
            if cur.left is not None:
                q.append(cur.left)
            if cur.right is not None:
                q.append(cur.right)
        return res
```

987. Vertical Order Traversal of a Binary Tree

```
# 同一列排序
class Solution:
    def verticalTraversal(self, root: Optional[TreeNode]) -> List[List[int]]:
        # 使用h和v分别记录水平方向的和竖直方向的坐标情况
        min_h, max_h = 10000, -10000
        node_for_each_h = defaultdict(list)
        q = [(root.val, 0, 0, root)]
        while q:
            val, h, v, node = q.pop(0)
            min_h = min(min_h, h)
            max_h = max(max_h, h)
            node_for_each_h[h].append((v, val))
            if node.left:
                q.append((node.left.val, h-1, v+1, node.left))
            if node.right:
                q.append((node.right.val, h+1, v+1, node.right))
        res = []
        for h in range(min_h, max_h+1):
            res.append([item[1] for item in sorted(node_for_each_h[h])])
        return res
```

中序遍历与搜索树

94. Binary Tree Inorder Traversal

```python
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        # 中序遍历就是等价于左边界分解
        stack, res = [], []
        node = root
        while node is not None:
            stack.append(node)
            node = node.left
        while stack:
            cur = stack.pop(-1)
            res.append(cur.val)
            if cur.right is not None:
                node = cur.right
                # 看到一个节点就把所有左边界压进去
                while node is not None:
                    stack.append(node)
                    node = node.left
        return res
```

95. Unique Binary Search Trees II

```
class Solution:
    def generateTrees(self, n: int) -> List[Optional[TreeNode]]:
        # 获得所有可行的左子树和可行的右子树
        # 那么最后一步我们只要从可行左子树集合中选一棵
        # 再从可行右子树集合中选一棵拼接到根节点上
        # 并将生成的二叉搜索树放入答案数组即可
        def generate(left, right):
            if left > right:
                return [None]
            res = []
            for i in range(left, right+1):
                left_candidates = generate(left, i-1)
                right_candidates = generate(i+1, right)
                for l in left_candidates:
                    for r in right_candidates:
                        node = TreeNode(i)
                        node.left = l
                        node.right = r
                        res.append(node)
            return res
        return generate(1, n)
```

96. Unique Binary Search Trees

```
class Solution:
    def numTrees(self, n: int) -> int:
        # f(i) means the number of structurally unique BST
        # with root i, then G(n) = f(1) + ... + f(n)
        # since f(i) = G(i−1) ∗ G(n−i), then we get the
        # answer is G(n)=G(0)G(n1)+G(1)G(n2)+...+G(n1)G(0)
        G = [0] * (n+1)
        G[0] = 1
        for i in range(1, n+1):
            for j in range(i):
                G[i] += G[j]*G[i-1-j]
        return G[-1]
```

98. Validate Binary Search Tree

```
class Solution:
    def isValidBST(self, root: Optional[TreeNode]) -> bool:
        self.val = float("-inf")

        def dfs(node):
            if node == None:
                return True
            left = dfs(node.left)
            if self.val < node.val:
                self.val = node.val
                flag = True
            else:
                flag = False
            right = dfs(node.right)
            return left & flag & right
        return dfs(root)
```

99. Recover Binary Search Tree

```
class Solution:
    def recoverTree(self, root: Optional[TreeNode]) -> None:
        """
        Do not return anything, modify root in-place instead.
        """
        # 错误1:
        #  出现了两对不满足前小后大
        # 需要交换第一对的第一个元素与第二对的第二个元素
        # 错误2:
        # 只出现一对不满足前小后大，交换这一对元素即可
        self.prev = TreeNode(float("-inf"))
        self.err1, self.err2 = None, None
        def dfs(node):
            if node is None:
                return
            dfs(node.left)
            if self.err1 is None and self.prev.val >= node.val:
                self.err1 = self.prev
            # 非常巧妙，把错误1和错误2共用代码，一次更改
            if self.err1 is not None and self.prev.val >= node.val:
                self.err2 = node
            self.prev = node
            dfs(node.right)
        dfs(root)
        self.err1.val, self.err2.val = self.err2.val, self.err1.val
```

108. Convert Sorted Array to Binary Search Tree

```
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:
        if len(nums) == 0:
            return None
        pos = len(nums) // 2
        val = nums[pos]
        node = TreeNode(val)
        node.left = self.sortedArrayToBST(nums[:pos])
        node.right = self.sortedArrayToBST(nums[pos+1:])
        return node
```

109. Convert Sorted List to Binary Search Tree

```
class Solution:
    def sortedListToBST(self, head: Optional[ListNode]) -> Optional[TreeNode]
        if head == None:
            return None
        elif head.next == None:
            return TreeNode(head.val)
        slow, fast = head, head
        # why we need pre, because we should
        # cancel the next of last node before slow node
        # * * (pre) (slow) * * * (fast)
        pre = None
        while fast != None and fast.next != None:
            pre = slow
            slow = slow.next
            fast = fast.next.next
        # if the "elif" missing, then here pre is still None
        pre.next = None
        root = TreeNode(slow.val)
        root.left = self.sortedListToBST(head)
        root.right = self.sortedListToBST(slow.next)
        return root
```

173. Binary Search Tree Iterator

```
class BSTIterator:

    # 中序遍历的迭代方法：不由分说，压入左边界
    def __init__(self, root: Optional[TreeNode]):
        self.stack = []
        self.pop_all_left(root)

    def pop_all_left(self, root):
        while root != None:
            self.stack.append(root)
            root = root.left

    def next(self) -> int:
        if self.hasNext():
            node = self.stack.pop(-1)
            # 弹出后检查右边是否有，否则再也没有机会弹出
            self.pop_all_left(node.right)
            return node.val

    def hasNext(self) -> bool:
        return len(self.stack) != 0


# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

230. Kth Smallest Element in a BST

```
class Solution:
    def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:

        temp = root
        stack = []
        while temp:
            stack.append(temp)
            temp = temp.left
        while stack:
            k -= 1
            node = stack.pop(-1)
            if k == 0:
                return node.val
            if node.right:
                temp = node.right
                while temp:
                    stack.append(temp)
                    temp = temp.left
```

285. Inorder Successor in BST

```
class Solution:
    def inorderSuccessor(self, root: TreeNode, p: TreeNode):
        # 1) p没有右孩子，且p是父节点的左节点，或者某个节点左子树的最右元素
        # 那么这个节点或者父节点就是返回的
        # 2) p有右孩子，此时返回右子树最左边的元素
        node = p
        if node.right is not None:
            # 第二种情况
            node = node.right
            while node.left is not None:
                node = node.left
            return node
        else:
            # 第一种情况
            res = None
            node = root # 注意这里是从根节点出发
            while node != p:
                if node.val > p.val:
                    res = node
                    node = node.left
                elif node.val < p.val:
                    node = node.right
            return res
```

333. Largest BST Subtree

```
class Solution(object):
    def largestBSTSubtree(self, root):
        self.res = 0
        def find(node):
            if not node: return float('inf'), float('-inf'), 0
            lmin, lmax, lnum = find(node.left)
            rmin, rmax, rnum = find(node.right)
            n = float('-inf')
            if node.val > lmax and node.val < rmin:
                n = lnum + rnum + 1
                self.res = max(n, self.res)
            return min(node.val, lmin), max(node.val, rmax), n
        find(root)
        return self.res
```

426. Convert Binary Search Tree to Sorted Doubly Linked List

```
class Solution:
    def treeToDoublyList(self, root: 'Optional[Node]') -> 'Optional[Node]':
        # 关键在于用一个全局变量保存prev node，然后在遍历过程中更新
        self.pre = None
        self.head = None
        if root is None:
            return root
        def dfs(root):
            if root is None:
                return
            dfs(root.left)
            if self.pre is None:
                self.pre = root
                self.head = root
            else:
                root.left = self.pre
                self.pre.right = root
                self.pre = root
            dfs(root.right)
        dfs(root)
        self.pre.right = self.head
        self.head.left = self.pre
        return self.head
```

450. Delete Node in a BST

```
class Solution:
    def deleteNode(self, root: Optional[TreeNode], key: int):
        # 找到后:
        # 1) 不存在右子树，左子树提上来
        # 2) 不存在左子树，右子树提上来
        # 3) 两棵子树都在，把左子树最大的或者右子树最小的拿上来
        if root == None:
            return root
        if key < root.val:
            root.left = self.deleteNode(root.left, key)
            return root
        if key > root.val:
            root.right = self.deleteNode(root.right, key)
            return root
        if root.left == None:
            return root.right
        if root.right == None:
            return root.left

        # 此处选择拿左子树最大的
        temp = root.left
        while temp != None:
            val = temp.val
            temp = temp.right
        root.val = val
        root.left = self._remove_max(root.left)
        return root

    # 这个写法很nb，如果当前的右边是空的，则把左边的拿上来
    # 此时左边可以是空的，换头并返回头
    def _remove_max(self, node):
        if node.right is None:
            new_root = node.left
            node.left = None
            return new_root
        node.right = self._remove_max(node.right)
        return node
```

530. Minimum Absolute Difference in BST

```
class Solution:
    def getMinimumDifference(self, root: Optional[TreeNode]) -> int:

        self.res = float("inf")
        self.last = float("inf")
        def helper(root):
            if root == None:
                return
            helper(root.left)
            self.res = min(abs(root.val-self.last), self.res)
            self.last = root.val
            helper(root.right)

        helper(root)
        return self.res
```

538. Convert BST to Greater Tree

```
class Solution:
    def convertBST(self, root: Optional[TreeNode]) -> Optional[TreeNode]:

        self.cur = 0
        def helper(root):
            if root == None:
                return
            helper(root.right)
            root.val += self.cur
            self.cur = root.val
            helper(root.left)
        helper(root)
        return root
```

669. Trim a Binary Search Tree

```
class Solution:
    def trimBST(self, root: Optional[TreeNode], low: int, high: int):

        self.low, self.high = low, high
        root = self.cut_low(root)
        root = self.cut_high(root)
        return root

    # 搞清楚return的是什么，应当是能够作为当前节点的节点
    def cut_low(self, root):
        if root == None:
            return root
        if self.low < root.val:
            root.left = self.cut_low(root.left)
            # 当前节点是要保留的
            return root
        if self.low > root.val:
            # 当前节点不会保留
            return self.cut_low(root.right)
        else:
            # 由于是闭区间，因此当前节点是要保留的
            root.left = None
            return root

    def cut_high(self, root):
        if root == None:
            return root
        if self.high < root.val:
            return self.cut_high(root.left)
        if self.high > root.val:
            root.right = self.cut_high(root.right)
            return root
        else:
            root.right = None
            return root
```

700. Search in a Binary Search Tree

```
class Solution:
    def searchBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeN
        if root is None:
            return None
        if root.val > val:
            return self.searchBST(root.left, val)
        elif root.val < val:
            return self.searchBST(root.right, val)
        else:
            return root
```

构造与序列化

105. Construct Binary Tree from Preorder and Inorder Traversal

```
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]):
        pre_iter = iter(preorder)
        inorder_hash = dict(zip(inorder, range(len(inorder))))

        def helper(start, end):
            if start > end:
                return None
            root_val = next(pre_iter)
            root = TreeNode(root_val)
            inorder_root_pos = inorder_hash[root_val]
            root.left = helper(start, inorder_root_pos-1)
            root.right = helper(inorder_root_pos+1, end)
            return root

        return helper(0, len(preorder)-1)
```

106. Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution:
    def buildTree(self, inorder: List[int], postorder: List[int]):
        postorder = postorder[::-1] # left right root => root right left
        post_iter = iter(postorder)
        inorder_hash = dict(zip(inorder, range(len(inorder))))

        def helper(start, end):
            if start > end:
                return None
            root_val = next(post_iter)
            node = TreeNode(root_val)
            inorder_root_pos = inorder_hash[root_val]
            node.right = helper(inorder_root_pos+1, end)
            node.left = helper(start, inorder_root_pos-1)
            return node

        return helper(0, len(inorder)-1)
```

114. Flatten Binary Tree to Linked List

```
class Solution:
    def flatten(self, root: Optional[TreeNode]) -> None:
        """
        Do not return anything, modify root in-place instead.
        """
        """
        Thoughts: for each node
        1) save right and put left to right
        2) put right to rightest node of original left subtree
        3) delete left subtree
        4) continue to move to the next right
        """
        if root == None:
            return None
        if root.left != None:
            temp = root.right
            root.right = root.left
            rightest = root
            while rightest.right != None:
                rightest = rightest.right
            rightest.right = temp
            root.left = None
        self.flatten(root.right)
        return root
```

297. Serialize and Deserialize Binary Tree

```
class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """
        if root == None:
            return "*"
        return str(root.val)
            + "," + self.serialize(root.left)
            + "," + self.serialize(root.right)



    def deserialize(self, data):
        """Decodes your encoded data to tree.

        :type data: str
        :rtype: TreeNode
        """
        data = data.split(",")
        def build():
            cur = data.pop(0)
            if cur == "*":
                return None
            cur = TreeNode(int(cur))
            cur.left = build()
            cur.right = build()
            return cur
        return build()


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

331. Verify Preorder Serialization of a Binary Tree

```python
class Solution:
    def isValidSerialization(self, preorder: str) -> bool:
        # 消消乐
        # 用栈保存，如果它的元素个数>=3，那么看看最后是不是数字+#+#的组合
        # 如果是，那么用一个#替换掉，那这样遍历一遍下来，应该最后留下的
        # 就是只有一个，因为preorder的顺序是 头->左->右，栈里的顺序一样
        stack = []
        for n in preorder.split(","):
            stack.append(n)
            # 这里是while，消消乐，一直消到不符合条件
            while (
                len(stack) >= 3
                and stack[-3] != "#"
                and stack[-2] == "#"
                and stack[-1] == "#"
            ):
                stack.pop()
                stack.pop()
                stack.pop()
                stack.append("#")
        return len(stack) == 1 and stack[0] == "#"
```

589. N-ary Tree Preorder Traversal

```
class Solution:
    def preorder(self, root: 'Node') -> List[int]:
        if root is None:
            return root
        res = []
        stack = [root]
        while stack:
            cur = stack.pop(-1)
            res.append(cur.val)
            stack += cur.children[::-1]
        return res
```

652. Find Duplicate Subtrees

```
class Solution:
    def findDuplicateSubtrees(self, root: Optional[TreeNode]):
        count = defaultdict(int) # 记录每个节点的序列化结果
        res = []
        def dfs(root):
            if root is None:
                return "#"
            serialization = "{},{},{}".format(
                root.val, dfs(root.left), dfs(root.right))
            count[serialization] += 1
            if count[serialization] == 2:
                res.append(root)
            return serialization
        dfs(root)
        return res
```

最近祖先

235. Lowest Common Ancestor of a Binary Search Tree

```
# BST 树
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeN
        # binary search tree can decide the direction to go if not find target
        if root == None:
            return
        if root.val > p.val and root.val > q.val:
            return self.lowestCommonAncestor(root.left, p, q)
        elif root.val < p.val and root.val < q.val:
            return self.lowestCommonAncestor(root.right, p, q)
        else:
            return root # None or root or p or q
```

236. Lowest Common Ancestor of a Binary Tree

```python
# 一般的树
class Solution:
    def lowestCommonAncestor(
        self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode'):
        if root == p:
            return p
        elif root == q:
            return q
        elif root == None:
            return None
        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)
        if left == None and right == None:
            return None
        elif left != None and right != None:
            return root
        elif left != None and right == None:
            return left
        elif right != None and left == None:
            return right
```

1650. Lowest Common Ancestor of a Binary Tree III

```
# 没有root的普通树，但有parent
class Solution:
    def lowestCommonAncestor(self, p: 'Node', q: 'Node') -> 'Node':
        # 很妙的做法：参见160，两个链表分叉节点向前走
        # 然后走到头了换初始位置，然后接着走就能在分叉口相遇
        a, b = p, q
        while a != b:
            a = a.parent if a is not None else q
            b = b.parent if b is not None else p
        return a
"""
# 找到root之后，直接照抄236的helper function
class Solution:
    def lowestCommonAncestor(self, p: 'Node', q: 'Node') -> 'Node':
        def dfs(node):
            if node.parent == None:
                return node
            return dfs(node.parent)
        root = dfs(p)
        return self.helper(root, p, q)

    def helper(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        if root == p:
            return p
        elif root == q:
            return q
        elif root == None:
            return None
        left = self.helper(root.left, p, q)
        right = self.helper(root.right, p, q)
        if left == None and right == None:
            return None
        elif left != None and right != None:
            return root
        elif left != None and right == None:
            return left
        elif right != None and left == None:
            return right
"""
```

2096. Step-By-Step Directions From a Binary Tree Node to Another

```python
class Solution:
    def getDirections(self,
        root: Optional[TreeNode], startValue: int, destValue: int) -> str:
        # 思路：从根节点开始到目标的两个路径先拿到
        # 然后去重复的前缀，在把start出发的路径全部替换为向上路径
        self.start_s, self.dest_s = "", ""
        self.lst = []
        def dfs(root):
            if root is None:
                return
            if root.val == startValue:
                self.start_s = "".join(self.lst)
            if root.val == destValue:
                self.dest_s = "".join(self.lst)
            self.lst.append("L")
            dfs(root.left)
            self.lst.pop()
            self.lst.append("R")
            dfs(root.right)
            self.lst.pop()
        dfs(root)
        prefix_idx, len_s, len_d = 0, len(self.start_s), len(self.dest_s)
        for _ in range(min(len_s, len_d)):
            if self.start_s[prefix_idx] == self.dest_s[prefix_idx]:
                prefix_idx += 1
            else:
                break
        return "U" * len(self.start_s[prefix_idx:])
            + self.dest_s[prefix_idx:]
```

回溯法

17. Letter Combinations of a Phone Number

```python
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        letters = {"2": "abc", "3": "def", "4": "ghi",
                   "5": "jkl", "6": "mno", "7": "pqrs",
                   "8": "tuv", "9": "wxyz"}
        if len(digits) == 0:
            return []
        res = []
        # 记录一共几位数字
        n = len(digits)
        def dfs(i, s):
            # 终止条件为数字已经到了最后一位的后面
            if i == n:
                res.append(s)
                return
            for c in list(letters[digits[i]]):
                dfs(i+1, s+c)
        dfs(0, "")
        return res
```

22. Generate Parentheses

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        res = []
        def dfs(left, right, s):
            if left == n and right == n:
                res.append(s)
            # 右括号至少匹配一个左括号
            if left < right:
                return
            if left < n:
                dfs(left+1, right, s+"(")
            if right < n:
                dfs(left, right+1, s+")")
        dfs(0, 0, "")
        return res
```

37. Sudoku Solver

```
class Solution:
    def solveSudoku(self, board: List[List[str]]) -> None:
        def valid(i, j):
            row, col, block = set(), set(), set()
            for y in range(9):
                if board[i][y] != ".":
                    if board[i][y] in row:
                        return False
                    row.add(board[i][y])
            for x in range(9):
                if board[x][j] != ".":
                    if board[x][j] in col:
                        return False
                    col.add(board[x][j])
            xs, ys = i//3 * 3, j//3 * 3
            for x in range(3):
                for y in range(3):
                    if board[xs+x][ys+y] != ".":
                        if board[xs+x][ys+y] in block:
                            return False
                        block.add(board[xs+x][ys+y])
            return True
        self.flag = False
        def go_next(i, j):
            if j == 8: # 最后一行了
                if i == 8: # 不用搜了
                    self.flag = True
                else:
                    dfs(i+1, 0)
            else:
                dfs(i, j+1)
        def dfs(i, j):
            if board[i][j] != ".":
                go_next(i, j)
                if self.flag:
                    return
            else:
                for k in list("123456789"):
                    board[i][j] = k
                    if valid(i, j):
                        go_next(i, j)
                        if self.flag:
                            # 已经找到答案了，跳出去
                            return
                    board[i][j] = "." # 这条路不行，回撤
        dfs(0, 0)
```

39. Combination Sum

```python
class Solution:
    def combinationSum(self, candidates: List[int], target: int):
        # 从candidates选不重复的组合，每个元素可以选多次
        candidates.sort()
        res = []
        def dfs(arr, count):
            if count >= target:
                if count == target:
                    res.append(arr.copy())
                return
            for val in candidates:
                # 选的时候保证元素单调不减，从而去重
                if arr and val < arr[-1]:
                    continue
                arr.append(val)
                dfs(arr, count + val)
                arr.pop(-1)
        dfs([], 0)
        return res
```

40. Combination Sum II

```python
class Solution:
    def combinationSum(self, candidates: List[int], target: int):
        # candidates可能重复，但每个元素只能用一次
        candidates.sort()
        res = []
        def dfs(arr, count, idx):
            if count >= target:
                if count == target:
                    res.append(arr.copy())
                return
            for i in range(idx, len(candidates)):
                # [1, 1, 7] 8
                # 不跳过的话会导致重复[1, 7] [1, 7]
                if i > idx and candidates[i] == candidates[i-1]:
                    continue
                val = candidates[i]
                arr.append(val)
                dfs(arr, count + val, i+1)
                arr.pop(-1)
        dfs([], 0, 0)
        return res
```

46. Permutations

```python
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        res = []
        used = [False] * len(nums)
        def dfs(arr):
            if len(arr) == len(nums):
                res.append(arr.copy())
                return
            for i in range(len(nums)):
                if not used[i]:
                    used[i] = True
                    dfs(arr + [nums[i]])
                    used[i] = False
        dfs([])
        return res
```

47. Permutations II

```
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        # nums可能有重复值，结果需要去重
        res = []
        remain = defaultdict(int)
        for i in nums:
            remain[i] += 1
        def dfs(arr):
            if len(arr) == len(nums):
                res.append(arr.copy())
                return
            for i in remain:
                if remain[i] > 0:
                    remain[i] -= 1
                    dfs(arr + [i])
                    remain[i] += 1
        dfs([])
        return res
```

51. N-Queens

```python
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:

        board = [["."] * n for _ in range(n)]
        visited = [False] * n # 表示这一列是不是有Q了
        res = []

        # 每次都要检查visited（上面）和左上角以及右上角
        def dfs(row, board):
            if row == n: # 到最后一行了
                res.append(["".join(item) for item in board].copy())
                return
            for col in range(n):
                if visited[col]:
                    continue
                Flag= True # 表示是不是能放
                cur_row, cur_col = row, col
                # 左上角
                while cur_row-1 >= 0 and cur_col-1 >= 0:
                    cur_row -= 1
                    cur_col -= 1
                    if board[cur_row][cur_col] == "Q":
                        Flag = False
                        break
                cur_row, cur_col = row, col # 记得复原
                # 右上角
                while cur_row-1 >= 0 and cur_col+1 < n:
                    cur_row -= 1
                    cur_col += 1
                    if board[cur_row][cur_col] == "Q":
                        Flag = False
                        break
                if Flag:
                    board[row][col] = "Q"
                    visited[col] = True
                    dfs(row+1, board)
                    board[row][col] = "."
                    visited[col] = False

        dfs(0, board)
        return res
```

52. N-Queens II

```
class Solution:
    def totalNQueens(self, n: int) -> int:
        self.res = 0
        self.board = [[False] * n for _ in range(n)]
        self.col_valid = [True] * n
        def check(r, c):
            valid = True
            temp_r, temp_c = r-1, c-1
            while temp_r >= 0 and temp_c >= 0:
                valid &= ~self.board[temp_r][temp_c]
                temp_r -= 1
                temp_c -= 1
            temp_r, temp_c = r-1, c+1
            while temp_r >= 0 and temp_c < n:
                valid &= ~self.board[temp_r][temp_c]
                temp_r -= 1
                temp_c += 1
            return valid
        def dfs(row):
            if row == n:
                self.res += 1
                return
            for col in range(n):
                self.board[row][col] = True
                if self.col_valid[col] and check(row, col):
                    self.col_valid[col] = False
                    dfs(row+1)
                    self.col_valid[col] = True
                self.board[row][col] = False
        dfs(0)
        return self.res
```

77. Combinations

```python
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        res = []
        def dfs(i, arr):
            if len(arr) == k:
                res.append(arr.copy())
                return False
            if i == n:
                return True # 说明已经到最后一位的后一位了
            for j in range(i, n):
                # j+1是下一个元素的出现位置
                # 如果这个位置已经越界了，那么后面的都不用看，加速
                if dfs(j+1, arr + [j+1]):
                    break
        dfs(0, [])
        return res
```

78. Subsets

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        res = []
        def dfs(i, arr):
            if i == len(nums):
                res.append(arr)
                return
            # 选择当前元素和不选择当前元素两种
            dfs(i+1, arr)
            dfs(i+1, arr + [nums[i]])
        dfs(0, [])
        return res
```

79. Word Search

```python
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:

        n, m = len(board), len(board[0])
        visited = [[False] * m for _ in range(n)]

        def dfs(i, j, ptr):
            # 首先应该判断是否到底了，bad case: [["a"]], "a"
            # 否则如果先判断边界的话，可能全部四面八方都是不可行的
            if ptr >= len(word):
                return True
            if i<0 or j <0 or i>=n or j>=m:
                return False
            if visited[i][j]:
                return False
            if word[ptr] != board[i][j]:
                return False
            visited[i][j] = True
            res = any([
                dfs(i-1, j, ptr+1),
                dfs(i+1, j, ptr+1),
                dfs(i, j-1, ptr+1),
                dfs(i, j+1, ptr+1),
            ])
            visited[i][j] = False
            return res

        for i in range(n):
            for j in range(m):
                if dfs(i, j, 0):
                    # 最后一个数字是对word的位置指针
                    return True
        return False
```

90. Subsets II

```python
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        # nums可能重复，结果不能重复
        """
        [1,1,1,1]
        # 如果发生重复了，那么只看当前的这一个
        [1]
        [1,1]
        [1,1,1]
        [1,1,1,1]
        """
        n = len(nums)
        nums.sort()
        res = []
        def dfs(i, arr):
            # 进到这步了，肯定是合法
            res.append(arr.copy())
            if i >= n:
                return
            for j in range(i, n):
                if j > i and nums[j] == nums[j-1]:
                    # 去重
                    continue
                # 注意是j+1不是i+1
                dfs(j+1, arr + [nums[j]])
        dfs(0, [])
        return res
```

93. Restore IP Addresses

```
class Solution:
    def restoreIpAddresses(self, s: str) -> List[str]:
        res = set()
        # s1是结果，s2是还没有处理的字符串
        # i代表当前处理的段，4表示处理完了
        def dfs(s1, s2, i):
            if i == 4 and len(s2) == 0:
                res.add(s1)
                return
            for j in range(3):
                # 每个段最多3位
                n = s2[:j+1]
                if n.startswith("0") and len(n) >= 2 or n == "":
                    # 0开头也数字大于2，或者空串，后面都不可能了
                    break
                if int(n) >= 0 and int(n) <= 255:
                    # 是不是要加点取决于是否处理的最后一个段
                    if i < 3:
                        dfs(s1+n+".", s2[j+1:], i+1)
                    else:
                        dfs(s1+n, s2[j+1:], i+1)
        dfs("", s, 0)
        return list(res)
```

130. Surrounded Regions

```python
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        m, n = len(board), len(board[0])
        def dfs(i, j):
            if i < 0 or j < 0 or i >= m or j >= n:
                return
            if board[i][j] == "X" or board[i][j] == "o":
                return
            board[i][j] = "o"
            dfs(i+1, j)
            dfs(i-1, j)
            dfs(i, j-1)
            dfs(i, j+1)
        # 在四周边界上的O用o代替标记
        for i in range(m):
            dfs(i, 0)
            dfs(i, n-1)
        for j in range(n):
            dfs(0, j)
            dfs(m-1, j)
        for i in range(m):
            for j in range(n):
                # 这个时候遇到O了，说明和边界不连通
                if board[i][j] == "O":
                    board[i][j] = "X"
                # 把o恢复成O
                elif board[i][j] == "o":
                    board[i][j] = "O"
```

200. Number of Islands

```python
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        n, m = len(grid), len(grid[0])
        visited = [[False]*m for _ in range(n)]
        def dfs(i, j):
            if (
                i<0 or i>=n or j<0 or j>=m
                or visited[i][j] or grid[i][j] == "0"
            ):
                return
            visited[i][j] = True
            dfs(i+1,j)
            dfs(i-1,j)
            dfs(i,j+1)
            dfs(i,j-1)
        count = 0
        for i in range(n):
            for j in range(m):
                if (not visited[i][j]) and grid[i][j] == "1":
                    dfs(i, j)
                    count += 1
        return count
```

216. Combination Sum III

```
class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        res = []
        def dfs(arr, digit, val):
            # digit是当前数字，val是总和
            if len(arr) == k:
                if val == n:
                    res.append(arr.copy())
                return
            if val >= n:
                # 不满k个已经到了总和，不满足要求
                return
            for i in range(digit, 10):
                dfs(arr + [i], i+1, val+i)
        dfs([], 1, 0)
        return res
```

241. Different Ways to Add Parentheses

```
class Solution:
    def diffWaysToCompute(self, expression: str) -> List[int]:
        # x op y是子问题，x可以是a op b
        # base case就是只有一个数
        if expression.isdigit():
            return [int(expression)]
        # res存放的是所有这个区域内可能的运算结果
        res = []
        for i in range(len(expression)):
            val = expression[i]
            if val in ["+", "-", "*"]:
                left = self.diffWaysToCompute(expression[:i])
                right = self.diffWaysToCompute(expression[i+1:])
                for l in left:
                    for r in right:
                        if val == "+":
                            res.append(l + r)
                        elif val == "-":
                            res.append(l - r)
                        elif val == "*":
                            res.append(l * r)
        return res
```

282. Expression Add Operators

```
class Solution(object):
    def addOperators(self, num, target):

        def calculate(num, target, expr, prev, ans):
            if len(num) == 0 and ans == target:
                self.results.append(expr)
            else:
                # i表示本次处理的是当前num前i位
                for i in range(1, len(num)+1):
                    # 如果长度至少为2了，然后第一位是0
                    # 那么这个不可能（bad case: "1+0+2=3"）
                    if i > 1 and num[0] == '0':
                        continue
                    a = int(num[0:i])
                    if expr == "":
                        # 第一个字符不能是运算符
                        calculate(num[i:], target, num[:i], a, a)
                    else:
                        calculate(
                            num[i:],
                            target,
                            expr+'+'+num[:i],
                            a,
                            ans+a,
                        )
                        calculate(
                            num[i:],
                            target,
                            expr+'-'+num[:i],
                            -a,
                            ans-a,
                        )
                        # 对于乘法而言，1+2*3*(4)，先1+2*3扣掉2*3
                        # 也就是ans-prev，然后再加上2*3*4
                        # 也就是+prev*a，合起来就是ans+prev*(a-1)
                        calculate(
                            num[i:],
                            target,
                            expr+'*'+num[:i],
                            a*prev,
                            ans+prev*(a-1),
                        )
        self.results = []
        calculate(num, target, '', 0, 0)
        return self.results
```

301. Remove Invalid Parentheses

```
class Solution:
    def removeInvalidParentheses(self, s: str) -> List[str]:
        # 每个位置的括号都有选和不选两种策略，从而可以进行dfs
        # 当递归到字符串长度时，需要判断左括号是不是等于右括号
        # 因此，可以将左括号看做1，右括号看做−1，看加起来score是不是为0
        n = len(s)
        self.res = set([""])
        self.cur_max = 0
        # 去掉最少等于保留最多，我们只需找到合法字符串中最长的即可
        def dfs(pos, cur_str, score):
            if score < 0:
                return
            if pos == n:
                if score > 0:
                    return
                if len(cur_str) >= self.cur_max:
                    # 如果发现更长的，则重新初始化结果set
                    # 因此留下来的一定是最长的
                    if len(cur_str) > self.cur_max:
                        self.res = set()
                        self.cur_max = len(cur_str)
                    self.res.add(cur_str)
                return
            if s[pos] == "(":
                dfs(pos+1, cur_str+"(", score+1)
                dfs(pos+1, cur_str, score)
            elif s[pos] == ")":
                dfs(pos+1, cur_str+")", score-1)
                dfs(pos+1, cur_str, score)
            else:
                dfs(pos+1, cur_str+s[pos], score)
        dfs(0, "", 0)
        return list(self.res)
```

306. Additive Number

```python
class Solution:
    def isAdditiveNumber(self, num: str) -> bool:

        def check(s1, s2, s):
            if len(s1) > 1 and s1[0] == "0":
                return False
            if len(s2) > 1 and s2[0] == "0":
                return False
            if len(s) == 0:
                return True
            s3 = str(int(s2) + int(s1))
            # 看看是不是剩下的开头是前面两个之和，然后继续检查后面
            if s.startswith(s3):
                return check(s2, s3, s[len(s3):])
            else:
                return False

        # 固定前两个，后面递归地检查
        n = len(num)
        for i in range(n-2):
            for j in range(i+1, n-1):
                if check(num[:i+1], num[i+1:j+1], num[j+1:]):
                    return True
        return False
```

332. Reconstruct Itinerary

```
class Solution:
    def findItinerary(self, tickets: List[List[str]]) -> List[str]:

        link_graph = defaultdict(list)
        for i, j in tickets:
            link_graph[i].append(j)
        for city in link_graph:
            link_graph[city].sort()

        res = []
        def dfs(city):
            # only have 2 results when a node go deeper
            # 1) to the end of traversal
            # 2) to itself from a circle
            while link_graph[city]: # still have ticket(s) not use
                dfs(link_graph[city].pop(0)) # smaller get fronter
            res.insert(0, city)
        dfs("JFK")
        return res
```

339. Nested List Weight Sum

```
class Solution:
    def depthSum(self, nestedList: List[NestedInteger]) -> int:
        self.res = 0
        def dfs(L, d):
            for i in L:
                if i.isInteger():
                    self.res += i.getInteger() * d
                else:
                    dfs(i.getList(), d+1)
        dfs(nestedList, 1)
        return self.res
```

397. Integer Replacement

```
class Solution:
    def integerReplacement(self, n: int) -> int:
        self.count = float("inf")
        def dfs(n, c):
            if c >= self.count:
                # 剪枝，不用再往下看了
                # 因为c往下肯定大于self.count
                return
            if n == 1:
                # 碰到底部了，取较小值后返回
                self.count = min(c, self.count)
                return
            if n%2 == 0:
                dfs(n/2, c+1)
            else:
                dfs(n+1, c+1)
                dfs(n-1, c+1)
        dfs(n, 0)
        return self.count
```

401. Binary Watch

```
class Solution:
    def readBinaryWatch(self, turnedOn: int) -> List[str]:
        self.t = turnedOn
        self.digit = [8,4,2,1,32,16,8,4,2,1]
        self.res = []
        self.dfs(0, 0, 0)
        return self.res
    def dfs(self, idx, hour, minute):
        # 要开的灯比有的还多
        if self.t > 10 - idx:
            return
        if self.t == 0:
            # 合法时间
            if minute <= 59 and hour <= 11:
                self.res.append("%d:%02d"%(hour, minute))
            return
        # 按照当前决定是否亮灯的位置分类
        if idx <= 3:
            self.t -= 1
            self.dfs(idx+1, hour+self.digit[idx], minute)
            self.t += 1
            self.dfs(idx+1, hour, minute)
        else:
            self.t -= 1
            self.dfs(idx+1, hour, minute+self.digit[idx])
            self.t += 1
            self.dfs(idx+1, hour, minute)
```

417. Pacific Atlantic Water Flow

```python
class Solution:
    def pacificAtlantic(self, heights: List[List[int]]):
        # 本质上就是四周开始做搜索，能不能前进取决于下一步是不是值更小
        m, n = len(heights), len(heights[0])
        record = [[[False, False] for __ in range(n)] for _ in range(m)]
        # 第一（二）个位置代表是不是可以流到上（下）面的海洋
        def dfs(i, j, last_val, ocean):
            if i < 0 or j < 0 or i >= m or j >= n:
                return
            if visited[i][j]:
                return
            if heights[i][j] < last_val:
                return
            record[i][j][ocean] = True
            visited[i][j] = True
            dfs(i+1, j, heights[i][j], ocean)
            dfs(i-1, j, heights[i][j], ocean)
            dfs(i, j+1, heights[i][j], ocean)
            dfs(i, j-1, heights[i][j], ocean)
        visited = [[False] * n for _ in range(m)]
        for i in range(m):
            dfs(i, 0, float("-inf"), 0)
        for j in range(n):
            dfs(0, j, float("-inf"), 0)
        visited = [[False] * n for _ in range(m)]
        for i in range(m):
            dfs(i, n-1, float("-inf"), 1)
        for j in range(n):
            dfs(m-1, j, float("-inf"), 1)
        res = []
        for i in range(m):
            for j in range(n):
                if all(record[i][j]):
                    res.append([i, j])
        return res
```

419. Battleships in a Board

```
class Solution:
    def countBattleships(self, board: List[List[str]]) -> int:
        self.board = board
        self.count = 0
        r, c = len(board), len(board[0])
        def dfs(i, j):
            if i < 0 or j < 0 or i > r-1 or j > c-1:
                return
            if self.board[i][j] == ".":
                return
            self.board[i][j] = "."
            dfs(i-1,j)
            dfs(i+1,j)
            dfs(i,j-1)
            dfs(i,j+1)
        for i in range(r):
            for j in range(c):
                if self.board[i][j] == "X":
                    self.count += 1
                    dfs(i, j)
        return self.count
```

489. Robot Room Cleaner

```
class Solution:
    def cleanRoom(self, robot):
        directions = [(0,1),(1,0),(0,-1),(-1,0)]
        visited = set()
        def dfs(x, y, d):
            # d表示direction
            visited.add("%d-%d"%(x, y))
            robot.clean()
            for _ in range(4):
                # 对于每一个direction，它的up是不一样的
                # 例如面朝右时，那么directions[1]才是up
                # 例如面朝下是，那么directions[2]才是up
                dx = directions[d%4][0]
                dy = directions[d%4][1]
                nx, ny = x + dx, y + dy
                if "%d-%d"%(nx, ny) not in visited and robot.move():
                    dfs(nx, ny, d%4)
                # 右转
                d += 1
                robot.turnRight()
            # 四个方向跑完，向后退，但保持方向（回溯状态）
            robot.turnRight()
            robot.turnRight()
            robot.move()
            robot.turnRight()
            robot.turnRight()
        dfs(0, 0, 0)

# """
# This is the robot's control interface.
# You should not implement it, or speculate about its implementation
# """
#class Robot:
# def move(self):
# """
# Returns true if the cell in front is open and robot moves into the cell.
# Returns false if the cell in front is blocked and robot stays in the current cell.
# :rtype bool
# """
#
# def turnLeft(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
```

```
#
# def turnRight(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def clean(self):
# """
# Clean the current cell.
# :rtype void
# """
```

491. Increasing Subsequences

```python
class Solution:
    def findSubsequences(self, nums: List[int]) -> List[List[int]]:
        # 1 1 2 3 4
        # 使用seen去重的意思其实就是选第二个1为开始
        # 等价于第一个1开始但第二个1不选
        self.res = []
        self.nums = nums
        self.n = len(nums)
        self.dfs(0, [])
        return self.res
    def dfs(self, idx, arr):
        if len(arr) > 1:
            self.res.append(arr.copy())
        seen = set()
        for i in range(idx, self.n):
            cur = self.nums[i]
            if cur in seen:
                continue
            if len(arr) == 0 or arr and cur >= arr[-1]:
                self.dfs(i+1, arr + [cur])
            seen.add(cur)
```

526. Beautiful Arrangement

```python
class Solution:
    def countArrangement(self, n: int) -> int:
        # 暴力解法，直接考虑全排列
        self.n = n
        self.used = [False] * n
        self.count = 0
        self.res = 0
        self.dfs([])
        return self.res
    def dfs(self, arr):
        if self.count == self.n:
            self.res += int(
                all((i+1)%v==0 or v%(i+1)==0
                for i, v in enumerate(arr))
            )
            return
        for i in range(1, self.n+1):
            # before adding, we can check validity for efficiency
            if (self.count+1)%i != 0 and i%(self.count+1) != 0:
                continue
            if not self.used[i-1]:
                self.used[i-1] = True
                arr.append(i)
                self.count += 1
                self.dfs(arr)
                arr.pop(-1)
                self.count -= 1
                self.used[i-1] = False
```

721. Accounts Merge

```
class Solution:
    def accountsMerge(self, accounts: List[List[str]]):
        # 把accounts的元素看成图的节点，这些节点之间通过email关系连接
        # 所以要构建一个从email到节点（account）的映射
        email_to_acc = defaultdict(list)
        for i, acc in enumerate(accounts):
            for j in range(1, len(acc)):
                email_to_acc[acc[j]].append(i)

        visited = [False] * len(accounts)
        def dfs(i):
            if visited[i]:
                return
            visited[i] = True
            cur_emails = accounts[i][1:]
            self.emails = self.emails.union(set(cur_emails))
            for e in cur_emails:
                # 通过这个map转移过去
                for acc in email_to_acc[e]:
                    dfs(acc)

        res = []
        for i, acc in enumerate(accounts):
            if visited[i]:
                continue
            # self.emails存放当轮临时结果
            name, self.emails = acc[0], set()
            dfs(i)
            res.append([name] + sorted(self.emails))
        return res
```

827. Making A Large Island

```
class Solution:
    def largestIsland(self, grid: List[List[int]]) -> int:
        m, n = len(grid), len(grid[0])
        idx_to_area = defaultdict(int)
        idx = 2

        def dfs(i, j):
            if i<0 or j<0 or i>=m or j>=n:
                return
            if grid[i][j] == 1:
                grid[i][j] = idx
                idx_to_area[grid[i][j]] += 1
                dfs(i+1, j)
                dfs(i-1, j)
                dfs(i, j+1)
                dfs(i, j-1)

        # 每个位置的元素都是当前元素所属岛屿的序号
        # 然后构造一个字典映射把序号映射到面积
        for i in range(m):
            for j in range(n):
                if grid[i][j] == 1:
                    dfs(i, j)
                    idx += 1
        def valid(i, j):
            if i<0 or j<0 or i>=m or j>=n:
                return 0
            else:
                return grid[i][j]

        def count(i, j):
            grid_true = set(
                [valid(i,j-1),valid(i,j+1),valid(i-1,j),valid(i+1,j)]
            )
            return sum(idx_to_area[i] for i in grid_true)

        res = 0
        # 对于每个位置都进行上下左右的加和计算
        for i in range(m):
            for j in range(n):
                if grid[i][j] == 0:
                    res = max(count(i, j)+1, res)

        # 为了防止所有位置都是岛屿，因此需
        # 要对这种情况的字典值最大值判断大小
        return max(res, max(idx_to_area.values()))
```

BFS

126. Word Ladder II

```python
class Solution:
    def findLadders(
        self,
        beginWord: str,
        endWord: str,
        wordList: List[str],
    ):
        # 区别在于统计visited路径的时候要当心，当层结束了再更新
        # 另外bfs一层一层做，不要一起做
        visited = set()
        res = []
        q = [[beginWord]]
        words = set(wordList)
        m = len(beginWord) # 注意所有词长度一样
        layer_found = False

        while q:
            if layer_found:
                break
            curLayerWords = set() # 为了统一更新当前的visited
            # 用q的长度控制当前层的for
            for _ in range(len(q)):
                curList = q.pop(0)
                lastword = curList[-1]
                if lastword == endWord:
                    res.append(curList)
                    layer_found = True
                # 逐个位置替换看看是不是在词表里且之前层没有访问
                for i in range(m):
                    for c in string.ascii_lowercase:
                        w = lastword[:i] + c + lastword[i+1:]
                        if w not in visited and w in words:
                            q.append(curList + [w])
                            curLayerWords.add(w)
            visited.update(curLayerWords)

        return res
```

127. Word Ladder

```python
class Solution:
    def ladderLength(
        self,
        beginWord: str,
        endWord: str,
        wordList: List[str],
    ):
        # Every adjacent pair of words differs by a single letter.
        d = defaultdict(list)
        for w in wordList:
            for i in range(len(w)):
                # 每个字符串在d中加入可能的通配符
                # 构造从通配符到真正字符串的映射
                d[w[:i]+"*"+w[i+1:]].append(w)
        queue = [(beginWord, 1)] # 第2个元素代表BFS的level
        visited = {i:False for i in wordList}
        while queue:
            cur_w, level = queue.pop(0)
            for i in range(len(cur_w)):
                # 通过通配符拿到字符串列表
                cur_ws = d[cur_w[:i]+"*"+cur_w[i+1:]]
                for w in cur_ws:
                    if w == endWord:
                        return level+1
                    else:
                        if not visited[w]:
                            queue.append((w, level+1))
                            visited[w] = True
        return 0
```

133. Clone Graph

```python
# BFS
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        if node is None:
            return None
        visited, q = dict(), [node]
        visited[node] = Node(node.val, [])
        # visited里的node映射到新图的node
        while q:
            cur = q.pop(0)
            for nb in cur.neighbors:
                if nb not in visited:
                    q.append(nb)
                    visited[nb] = Node(nb.val, [])
                # 最后才统一加邻居，因为这个时候邻居一定已经放到visited里面了
                visited[cur].neighbors.append(visited[nb])
        return visited[node]

# DFS
class Solution(object):
    def __init__(self):
        self.visited = dict()
    def cloneGraph(self, node):
        if not node:
            return node
        # 如果该节点已经被访问过了，则直接从哈希表中取出对应的克隆节点返回
        if node in self.visited:
            return self.visited[node]
        # 克隆节点，注意到为了深拷贝我们不会克隆它的邻居的列表
        clone_node = Node(node.val, [])
        # 哈希表存储
        self.visited[node] = clone_node
        # 遍历该节点的邻居并更新克隆节点的邻居列表
        if node.neighbors:
            clone_node.neighbors = [
                self.cloneGraph(n) for n in node.neighbors
            ]
        return clone_node
```

317. Shortest Distance from All Buildings

```
class Solution:
    def shortestDistance(self, grid: List[List[int]]) -> int:
        # 思路：考虑每个房子，计算到每个空地的距离，然后对每个距离用一个
        # 统一变量记录加和，所有空地对应的情况记录到字典里，最后求所有
        # 空地对应的最小值
        # 如何查看是否可达：每到新的房子时候，就把这个字典无法到达的位
        # 置给拿掉，最后若字典非空，那么一定这个元素可以到达所有房子
        candidate_empty = dict()
        building_pos = []
        self.pos = [(0,1),(0,-1),(1,0),(-1,0)]
        m, n = len(grid), len(grid[0])
        self.m, self.n, self.grid = m, n, grid
        for i in range(m):
            for j in range(n):
                if grid[i][j] == 0:
                    candidate_empty[(i, j)] = 0 # 记录加和初始化为0
                elif grid[i][j] == 1:
                    building_pos.append((i, j))
        for b_x, b_y in building_pos:
            self.count_dist_for_cur_building(candidate_empty, b_x, b_y)
        return min(candidate_empty.values()) if candidate_empty else -1
    def count_dist_for_cur_building(self, cdt, x, y):
        q = [(x, y)] # 一定要一个level一个level遍历
        visited = set()
        distance = 0
        while q:
            level_num = len(q)
            distance += 1
            for _ in range(level_num):
                cur_x, cur_y = q.pop(0)
                for dx, dy in self.pos:
                    new_x, new_y = cur_x + dx, cur_y + dy
                    new_pos = (new_x, new_y)
                    if (
                        new_x >= 0
                        and new_x < self.m
                        and new_y >= 0
                        and new_y < self.n
                    ):
                        if new_pos not in visited and new_pos in cdt:
                            visited.add(new_pos)
                            q.append(new_pos)
                            cdt[new_pos] += distance
        for i in set(cdt).difference(visited):
            cdt.pop(i) # 删除无法到达的
```

433. Minimum Genetic Mutation

```python
class Solution:
    def minMutation(self, start: str, end: str, bank: List[str]) -> int:
        bank = set(bank)
        chars = list("ACGT")
        q = [(start, 0)]
        while q:
            cur, step = q.pop(0)
            # cur除了初始的值之外，后面的值一定来自于bank
            # 但是end可以不来自于band，没有找到与end匹配会最终return -1
            if cur == end:
                return step
            for i in range(len(cur)):
                for c in chars:
                    if c != cur[i]:
                        new = cur[:i] + c +cur[i+1:]
                        if new in bank:
                            bank.remove(new)
                            q.append((new, step+1))
        return -1
```

778. Swim in Rising Water

```python
class Solution:
    def swimInWater(self, grid: List[List[int]]) -> int:
        # 题目意思：时间到了t才能走<=t的格子
        n, m = len(grid), len(grid[0])
        visited = [[False]*m for _ in range(n)]
        L = [(grid[0][0], 0, 0)]
        direction = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        visited[0][0] = 1
        res = 0
        while L:
            # 维护一个最小堆，每次总是弹出最小的元素
            cur = heapq.heappop(L)
            res = max(res, cur[0])
            x, y = cur[1], cur[2]
            if x == n-1 and y == m-1:
                break
            for i, j in direction:
                _x, _y = x+i, y+j
                if (
                    _x >= 0 and _x < n and _y >= 0 and _y < m
                    and (not visited[_x][_y])
                ):
                    visited[_x][_y] = True
                    heapq.heappush(L, (grid[_x][_y], _x, _y))
        return res
```

815. Bus Routes

```python
class Solution:
    def numBusesToDestination(
        self,
        routes: List[List[int]],
        source: int,
        target: int
    ):
        if source == target:
            return 0
        # 把公交车看做节点，如果有公共站则有边
        # 搜索时候把起点公交全部加到队列里，然后维护一个起点开始做
        # 某个公交需要换多少量的列表
        n = len(routes) # total number of bus lines
        edge = [[0] * n for i in range(n)]
        bus_on_site = defaultdict(list)
        for i in range(n):
            for site in routes[i]:
                for j in bus_on_site[site]:
                    edge[i][j] = 1
                    edge[j][i] = 1
                bus_on_site[site].append(i)

        q = []
        dist = [-1] * n
        for bus in bus_on_site[source]:
            # 出发站初始化为换乘数=1
            dist[bus] = 1
            q.append(bus) # remember that bus is node

        while q:
            cur_bus = q.pop(0)
            for bus in range(n):
                if edge[bus][cur_bus] and dist[bus] == -1:
                    q.append(bus)
                    dist[bus] = dist[cur_bus] + 1

        res = float("inf")
        # 查看终点站所有公交，挑一个换乘数最少的
        for bus in bus_on_site[target]:
            if dist[bus] != -1:
                res = min(res, dist[bus])
        return res if res != float("inf") else -1
```

863. All Nodes Distance K in Binary Tree

```
class Solution:
    def distanceK(self, root: TreeNode, target: TreeNode, k: int):
        # 因为元素唯一的，直接搞个邻接表出来，节点就是数字
        adj_list = defaultdict(set)
        def dfs(root):
            if root is None:
                return
            if root.left is not None:
                adj_list[root.val].add(root.left.val)
                adj_list[root.left.val].add(root.val)
                dfs(root.left)
            if root.right is not None:
                adj_list[root.val].add(root.right.val)
                adj_list[root.right.val].add(root.val)
                dfs(root.right)
        dfs(root)
        q = [target.val]
        res = []
        visited = set()
        while q:
            length = len(q)
            for i in range(length):
                cur = q.pop(0)
                visited.add(cur)
                for node in adj_list[cur]:
                    if node not in visited:
                        q.append(node)
                if k == 0:
                    res.append(cur)
            k -= 1
        return res
```

818. Race Car

```python
class Solution:
    def racecar(self, target: int) -> int:
        # BFS做法 log2(t)时间 log2(t)空间
        q = [(0, 0, 1)] # 当前步数 位置 速度
        visited = set()
        while q:
            step, pos, speed = q.pop(0)
            if pos == target:
                return step
            if (pos, speed) not in visited:
                visited.add((pos, speed))
                q.append((step + 1, pos + speed, speed * 2))
                if (
                    (pos + speed > target and speed > 0)
                    or (pos + speed < target and speed < 0)
                ):
                    # 表示下一步要远离
                    speed = -1 if speed > 0 else 1
                    q.append((step + 1, pos, speed))
```

994. Rotting Oranges

```
class Solution:
    def orangesRotting(self, grid: List[List[int]]) -> int:
        n, m = len(grid), len(grid[0])
        q = []
        count = 0
        res = 0
        for i in range(n):
            for j in range(m):
                if grid[i][j] == 1:
                    # 算一算一共多少个好橘子
                    count += 1
                elif grid[i][j] == 2:
                    # 准备多源BFS
                    q.append((i, j, 0))
        while q:
            i, j, time = q.pop(0)
            res = max(res, time)
            for dx, dy in [(0,1),(0,-1),(1,0),(-1,0)]:
                x, y = i+dx, j+dy
                if x<0 or y<0 or x>=n or y>=m:
                    continue
                # 通过把1改成2，天然标记，不用visited
                if grid[x][y] == 1:
                    count -= 1
                    grid[x][y] = 2
                    q.append((x, y, time+1))
        if count != 0:
            return -1
        else:
            return res
```

1284. Minimum Number of Flips to Convert Binary Matrix to Zero Matrix

```python
class Solution:
    def minFlips(self, mat: List[List[int]]) -> int:
        # 把整个矩阵作为一个状态，然后进行状态的BFS，此时搜到全零则停止
        # 但搜索并不一定要真的搜索矩阵，可以用字符串来代表矩阵
        m, n = len(mat), len(mat[0])
        def to_str(mat):
            s = ""
            for i in range(m):
                for j in range(n):
                    s += "1" if mat[i][j] else "0"
            return s
        final_str = "0" * (m*n)
        cur_mat_str = to_str(mat)
        if cur_mat_str == final_str:
            return 0
        visited = set([cur_mat_str])
        q = [(mat, 0)]
        while q:
            cur_mat, flip = q.pop(0)
            for i in range(m):
                for j in range(n):
                    next_mat = [
                        [cur_mat[s][t] for t in range(n)]
                        for s in range(m)
                    ]
                    next_mat[i][j] = 1 - next_mat[i][j]
                    if i > 0:
                        next_mat[i-1][j] = 1 - next_mat[i-1][j]
                    if j > 0:
                        next_mat[i][j-1] = 1 - next_mat[i][j-1]
                    if i < m-1:
                        next_mat[i+1][j] = 1 - next_mat[i+1][j]
                    if j < n-1:
                        next_mat[i][j+1] = 1 - next_mat[i][j+1]
                    next_mat_str = to_str(next_mat)
                    if next_mat_str in visited:
                        continue
                    elif next_mat_str == final_str:
                        return flip+1 # 这是下一轮的，所以加1
                    else:
                        visited.add(next_mat_str)
                        q.append((next_mat, flip+1))
        return -1
```

1293. Shortest Path in a Grid with Obstacles Elimination

```
class Solution:
    def shortestPath(self, grid: List[List[int]], k: int) -> int:
        # 存放了(i, j , k, step)四个信息
        step = 0
        q = deque([(0,0,k,0)])
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        visited = [
            [
                [
                    False] * (k+1)
                    for _ in range(len(grid[0]))
                ]
            for __ in range(len(grid))
        ]
        visited[0][0][k] = True
        # 这里优化，否则测试过不了
        # 意思是，k很大的时候畅通无阻
        if k > (len(grid)-1 + len(grid[0])-1):
            return len(grid)-1 + len(grid[0])-1
        while q:
            i, j, cur_k, step = q.popleft()
            for di, dj in directions:
                next_i, next_j = i + di, j + dj
                # 已到达目标则返回
                if next_i == len(grid)-1 and next_j == len(grid[0])-1:
                    return step + 1
                if (
                    next_i < 0 or next_j < 0
                    or next_i >= len(grid) or next_j >= len(grid[0])
                ):
                    continue
                next_k = cur_k-1 if grid[next_i][next_j] == 1 else cur_k
                if grid[next_i][next_j] == 1 and next_k < 0:
                    continue
                if visited[next_i][next_j][next_k]:
                    continue
                # visited是加在这里，这代表了当前轮立即到达这里
                # 而不是之后pop之后才visited
                visited[next_i][next_j][next_k] = True
                q.append((next_i, next_j, next_k, step+1))
        return -1
```

1631. Path With Minimum Effort

```python
class Solution:
    def minimumEffortPath(self, heights: List[List[int]]) -> int:

        # dijkstra变体，路径是一路取max而不是sum
        r, c = len(heights), len(heights[0])
        dists = [[float("inf") for _ in range(c)] for _ in range(r)]
        visited = [[False for _ in range(c)] for _ in range(r)]
        direction = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        dists[0][0] = 0

        q = [(0, 0, 0)]
        while q:
            dist, x, y = heappop(q)
            if x == r-1 and y == c-1:
                return dist
            if not visited[x][y]:
                visited[x][y] = True
                h = heights[x][y]
                for dx, dy in direction:
                    cur_x, cur_y = x + dx, y + dy
                    if 0 <= cur_x < r and 0 <= cur_y < c:
                        cur_h = heights[cur_x][cur_y]
                        if not visited[cur_x][cur_y]:
                            cur_dist = max(dist, abs(h - cur_h))
                            if cur_dist < dists[cur_x][cur_y]:
                                dists[cur_x][cur_y] = cur_dist
                                heappush(q, (cur_dist, cur_x, cur_y))
```

1730. Shortest Path to Get Food

```python
class Solution:
    def getFood(self, grid: List[List[str]]) -> int:
        # 正向会超时，应当从食物开始搜索出发点
        r, c = len(grid), len(grid[0])
        arr = []
        for i in range(r):
            for j in range(c):
                if grid[i][j] == "#":
                    arr.append((i, j))
        directions = ((-1, 0), (1, 0), (0, -1), (0, 1))
        step = 0
        while arr:
            next_arr = []
            for x, y in arr:
                for dx, dy in directions:
                    tmp_x, tmp_y = dx + x, dy + y
                    if r > tmp_x >= 0 and c > tmp_y >= 0:
                        if grid[tmp_x][tmp_y] == '*':
                            return step + 1
                        elif grid[tmp_x][tmp_y] == "O":
                            grid[tmp_x][tmp_y] = "X"
                            next_arr.append((tmp_x, tmp_y))
            arr = next_arr
            step += 1
        return -1
```

并查集

399. Evaluate Division

```
class Solution:
    def calcEquation(
        self,
        equations: List[List[str]],
        values: List[float],
        queries: List[List[str]]
    ) -> List[float]:

        alphabet = set()
        for pair in equations:
            alphabet.add(pair[0])
            alphabet.add(pair[1])
        self.root = {char:char for char in alphabet}
        self.val = {char:1 for char in self.root}
        # 初始值时候，所有相对root（也就是自己）的比例都是1

        for i, pair in enumerate(equations):
            self.union(pair[0], pair[1], values[i])

        res = []
        for pair in queries:
            if pair[0] not in alphabet or pair[1] not in alphabet:
                res.append(-1)
                continue
            x_head, x_val = self.find(pair[0])
            y_head, y_val = self.find(pair[1])
            if x_head != y_head:
                res.append(-1)
                continue
            # since x_val and y_val share the same
            # head, thus the ratio is meaningful
            res.append(x_val / y_val)

        return res

    def find(self, x):
        if self.root[x] != x:
            self.root[x], val = self.find(self.root[x])
            # the value in self.val means value of current
            # node over root node, then if we get parent node
            # comparative value from self.find, then the current
            # node value can be computed by cur:parent * parent:root
            # = cur:root
            self.val[x] *= val
        return self.root[x], self.val[x]
```

```
    def union(self, x, y, val):
        x_head, x_val = self.find(x)
        y_head, y_val = self.find(y)
        if x_head != y_head:
            # since all the value in self.value is comparative
            # to its root node, thus we should change the root
            # node's val to right propotion
            # here val[x_head] means x_head:y_head, val means
            # x:y, y_val means y:y_head and x_val means x:x_head
            # then we have the relationship in the following code
            self.root[x_head] = y_head
            self.val[x_head] = y_val / x_val * val
```

547. Number of Provinces

```
class Solution:
    # 看1584
    def findCircleNum(self, isConnected: List[List[int]]) -> int:

        self.root = list(range(len(isConnected)))
        self.rank = [1] * len(isConnected)

        for i in range(len(isConnected)):
            # attention: don't contain i == j
            for j in range(i):
                if isConnected[i][j] == 1:
                    self.union(i, j)

        count = 0
        # 利用并查集root数组的性质，head和值和索引一样
        for i, x in enumerate(self.root):
            if i == x:
                count += 1
        return count

    def find(self, x):
        if self.root[x] == x:
            return x
        self.root[x] = self.find(self.root[x])
        return self.root[x]

    def union(self, x, y):
        x_head = self.find(x)
        y_head = self.find(y)
        if x_head != y_head:
            if self.rank[x_head] > self.rank[y_head]:
                self.root[y_head] = x_head
            elif self.rank[x_head] < self.rank[y_head]:
                self.root[x_head] = y_head
            else:
                self.root[x_head] = y_head
                self.rank[y_head] += 1
```

1101. The Earliest Moment When Everyone Become Friends

```
class Solution:
    def earliestAcq(self, logs: List[List[int]], n: int) -> int:
        # First, we need to sort the events in chronological order.
        logs.sort(key = lambda x: x[0])
        uf = UnionFind(n) # n个节点
        group_cnt = n
        for timestamp, friend_a, friend_b in logs:
            if uf.union(friend_a, friend_b):
                group_cnt -= 1
            if group_cnt == 1:
                return timestamp
        return -1
class UnionFind:
    def __init__(self, size):
        self.group = [group_id for group_id in range(size)]
        self.rank = [0] * size
    def find(self, person):
        if self.group[person] != person:
            self.group[person] = self.find(self.group[person])
        return self.group[person]
    def union(self, a, b):
        # 如果之前就连通，那么返回true
        # 如果不连通，那么连接后返回false
        group_a = self.find(a)
        group_b = self.find(b)
        is_merged = False
        if group_a == group_b:
            return is_merged
        is_merged = True
        if self.rank[group_a] > self.rank[group_b]:
            self.group[group_b] = group_a
        elif self.rank[group_a] < self.rank[group_b]:
            self.group[group_a] = group_b
        else:
            self.group[group_a] = group_b
            self.rank[group_b] += 1
        return is_merged
```

1584. Min Cost to Connect All Points

```python
class Solution:
    def minCostConnectPoints(self, points: List[List[int]]) -> int:

        edge = []
        for i in range(len(points)):
            for j in range(i):
                p1, p2 = points[i], points[j]
                edge.append((i, j, abs(p1[0]-p2[0]) + abs(p1[1]-p2[1])))
        # 先把边给排序了，然后一个个节点加进去
        edge.sort(key=lambda t: t[2])

        self.root = list(range(len(points)))
        self.rank = [1] * len(points) # rank等价于root树的深度
        count, dist = len(points) - 1, 0

        while count > 0:
            cur = edge.pop(0)
            # 连通的话说明不是新节点
            if not self.isConnected(cur[0], cur[1]):
                self.union(cur[0], cur[1])
                dist += cur[2]
                count -= 1

        return dist

    def find(self, x):
        # 向前找头，如果索引和值相等了，说明头到了
        if self.root[x] != x:
            self.root[x] = self.find(self.root[x])
        return self.root[x]

    def union(self, x, y):
        x_head = self.find(x)
        y_head = self.find(y)
        if x_head != y_head:
            if self.rank[x_head] > self.rank[y_head]:
                self.root[y_head] = x_head
            elif self.rank[x_head] < self.rank[y_head]:
                self.root[x_head] = y_head
            else:
                self.root[x_head] = y_head
                self.rank[y_head] += 1

    def isConnected(self, x, y):
        # 头一样则连通
        return self.find(x) == self.find(y)
```

拓扑排序

207. Course Schedule

```python
class Solution:
    def canFinish(
        self,
        numCourses: int,
        prerequisites: List[List[int]]
    ):
        degree = [0] * numCourses
        edge = defaultdict(list)
        for e in prerequisites:
            degree[e[1]] += 1
            edge[e[0]].append(e[1])
        q = []
        for i in range(numCourses):
            if degree[i] == 0:
                q.append(i)
        if len(q) == 0:
            return False
        # 直接使用numCourses来判定
        while q:
            i = q.pop(0)
            numCourses -= 1
            for j in edge[i]:
                degree[j] -= 1
                if degree[j] == 0:
                    q.append(j)
        return numCourses == 0
```

210. Course Schedule II

```
class Solution:
    def findOrder(
        self,
        numCourses: int,
        prerequisites: List[List[int]]
    ):

        # init the indegree and linked graph
        indegree = [0] * numCourses
        graph = {i: [] for i in range(numCourses)}
        for pair in prerequisites:
            indegree[pair[0]] += 1
            graph[pair[1]].append(pair[0])

        # find out the init node with indegree of 0
        queue = []
        for i, d in enumerate(indegree):
            if d == 0:
                queue.append(i)

        res = []
        while queue:
            cur = queue.pop(0)
            res.append(cur)
            for node in graph[cur]:
                indegree[node] -= 1
                if indegree[node] == 0:
                    queue.append(node)

        # why can this detect cycle?
        # since only indegree == 0 will add to queue
        # thus any node in a cycle will not have chance
        # to reduce its indegree to 0, so the final
        # len(res) will be less then n_nodes
        return res if len(res) == numCourses else []
```

269. Alien Dictionary

```
class Solution:
    def alienOrder(self, words: List[str]) -> str:
        # 利用拓扑排序是基本思路，重要的是图怎么出?
        # 关键是想到实际上比较的关系，由于原来的数组是排序的
        # 因此只可能出现在左右相邻的单词之间
        graph, indegree = {}, {}
        chars = set(c for w in words for c in w)
        for c in chars:
            # 首先ab所有出现过的字母对应的邻接图和入度矩阵初始化
            graph[c] = set()
            indegree[c] = 0
        for i in range(len(words)-1):
            w1 = words[i]
            w2 = words[i+1]
            if len(w1) > len(w2) and w1[:len(w2)] == w2:
                # abc ab 不合法，下面的循环无法检测
                return ""
            for j in range(min(len(w1), len(w2))):
                c1, c2 = w1[j], w2[j]
                if c1 != c2:
                    if c1 in graph[c2]: # 产生了a->b和b->a的环，更大的环到拓扑
                        return ""
                    if c2 not in graph[c1]: # 多次出现边的依赖关系，indegree只加1
                        graph[c1].add(c2)
                        indegree[c2] += 1
                    break
        q = []
        for c in chars:
            if indegree[c] == 0:
                q.append(c)
        res = ""
        while q:
            c = q.pop(0)
            res += c
            for next_c in graph[c]:
                indegree[next_c] -= 1
                if indegree[next_c] == 0:
                    q.append(next_c)
                # 这里不用判断indegree[next_c] < 0，这是不可能的
        if any(indegree[c] != 0 for c in indegree):
            # 最后indegree没有归零，说明有环
            return ""
        return res
```

2115. Find All Possible Recipes from Given Supplies

```python
class Solution:
    def findAllRecipes(
        self,
        recipes: List[str],
        ingredients: List[List[str]],
        supplies: List[str]
    ):
        degree = defaultdict(int)
        graph = defaultdict(list)
        for ings, rec in zip(ingredients, recipes):
            for ing in ings:
                graph[ing].append(rec)
            degree[rec] += len(ings) # 每个菜需要多少原料

        # 略有差别，相当于supplies是若干点，它可能是中间层
        # 也可能入度非0，但是他们一定是搜索的起始点
        q, res = supplies, []
        while q:
            cur = q.pop(0)
            if cur in graph:
                # 为什么这里的rec一定合法?
                # 因为graph的values所有元素一定是rec
                # 而且，题目说所有recipes和supplies中的值互不相同
                # 那么supplies的一定是ingredient
                for rec in graph[cur]:
                    degree[rec] -= 1
                    if degree[rec] == 0:
                        q.append(rec)
                        res.append(rec)
        return res
```

动态规划

数组

45. Jump Game II

```python
class Solution:
    def jump(self, nums: List[int]) -> int:
        # ---------
        # ||
        # -------
        # ||
        # ----------
        # 尽可能让这个当前区间里面的元素更新最远的边界
        # end表示下一个能跳的最远边界
        end, max_pos, count = 0, 0, 0
        # 注意最后一个位置不用跳，所以遍历到len(nums)-1
        for i in range(len(nums)-1):
            # 找能跳的最远的
            max_pos = max(max_pos, i+nums[i])
            # 遇到边界了就更新
            if i == end:
                count += 1
                end = max_pos
        return count
```

53. Maximum Subarray

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        dp = nums[0]
        res = nums[0]
        for i in range(1, len(nums)):
            dp = dp + nums[i] if dp > 0 else nums[i]
            res = max(res, dp)
        return res
```

55. Jump Game

```
class Solution:
    def canJump(self, nums: List[int]) -> bool:

        if len(nums) == 1:
            return True

        # 是不是能跳到当前元素取决于，之前是否存在某一个元素（即跳的长度）
        # 不小于这个元素和当前元素的间隔，如果这个元素存在了，那么直接
        # 把它当成当前元素
        # 最后检查cur是不是到起点了

        # 贪心的逻辑：如果这个元素之前还有符合的，那么它一定能够先跳到
        # 这个元素，再跳到当前元素
        cur = len(nums) - 1
        for i in range(len(nums)-2, -1, -1):
            if nums[i] >= cur - i:
                cur = i

        return cur == 0
```

70. Climbing Stairs

```
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        if n == 2:
            return 2
        a, b = 1, 2
        for _ in range(n-2):
            temp = a + b
            a = b
            b = temp
        return temp
```

121. Best Time to Buy and Sell Stock

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:

        """
        # original answer
        buy = prices.copy()
        sell = prices.copy()
        for i in range(1, len(buy)):
            buy[i] = min(buy[i], buy[i-1])
            sell[len(buy)-i-1] = max(sell[len(buy)-i-1], sell[len(buy)-i])
        res = 0
        for i in range(len(buy)):
            res = max(sell[i] - buy[i], res)
        return max(0, res)
        """

        # dp[i][0] refers to the processed money of ith day without holding
        # dp[i][1] refers to ... with holding
        dp = [[0, -prices[0]] for i in range(len(prices))]
        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][1] + prices[i], dp[i-1][0])
            # 为什么不写成下面? 因为只能买一次
            # dp[i][1] = max(dp[i-1][0] - prices[i], dp[i-1][1])
            dp[i][1] = max(-prices[i], dp[i-1][1])
        return dp[len(prices)-1][0]
```

122. Best Time to Buy and Sell Stock II

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        # exactly the same to #121 only changes on dp[i][1] which
        # allows multiple buy and sell
        dp = [[0, -prices[0]] for i in range(len(prices))]
        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][1] + prices[i], dp[i-1][0])
            dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i])
        return dp[len(prices)-1][0]
```

198. House Robber

```python
class Solution:
    def rob(self, nums: List[int]) -> int:

        # 和740一样，但是空间上优化
        # 基本思路：用两个变量保存当前是拿还是不拿
        dp = [0, 0]
        for i in nums:
            temp = dp[0]
            dp[0] = max(dp) # 如果不拿，那么是上一轮里面去个大的
            dp[1] = temp + i # 如果拿了，那么上一轮不拿加上当前价值
        return max(dp)
```

213. House Robber II

```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return nums[0]
        # 构成一个圈，第一个元素和最后一个元素只能拿一个
        # 第一项代表不拿第一个元素，第二项代表不拿最后一个元素
        return max(
            self.get_rob_money(nums[1:]),
            self.get_rob_money(nums[:-1]),
        )

    def get_rob_money(self, nums):
        dp = [0, 0]
        for i in nums:
            temp = dp[0]
            dp[0] = max(dp)
            dp[1] = i + temp
        return max(dp)
```

264. Ugly Number II

```python
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        # 见313
        dp = [0] * (n+1)
        primes = [2, 3, 5]
        ptr = [0, 0, 0]
        nums = [1, 1, 1]

        for i in range(1, n+1):
            val = min(nums)
            dp[i] = val
            for j in range(3):
                if nums[j] == val:
                    # 这两行到底什么意思?
                    # 指针每次动一格，表示上一个*当前对应素数的已经不可能了
                    # 至少从新的这个数*素数才是可能的
                    ptr[j] += 1
                    # 那么为了记录这个更新的新的数字，就要把它放到nums里面去
                    nums[j] = dp[ptr[j]] * primes[j]

        return dp[-1]
"""
# Time Limit Exceeded
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        if n <= 3:
            return n
        pos = [1, 2, 3]
        for i in range(n-3):
            cur_min = float("inf")
            for num in pos:
                for factor in [2, 3, 5]:
                    val = num * factor
                    if val > pos[-1]:
                        cur_min = min(cur_min, val)
            while pos[0] * 5 <= pos[-1]:
                pos.pop(0)
            pos.append(cur_min)
        return pos[-1]
"""
```

279. Perfect Squares

```python
class Solution:
    def numSquares(self, n: int) -> int:
        # 把前面算过的作为dp数组的备忘录
        dp = [float("inf")] * (n+1)
        dp[0], dp[1] = 0, 1
        for i in range(2, n+1):
            j = 1
            while i-j*j >= 0:
                val = dp[i-j*j] + 1
                if dp[i] > val:
                    dp[i] = val
                j += 1
        return dp[-1]
```

309. Best Time to Buy and Sell Stock with Cooldown

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        dp = [[0, 0] for i in range(len(prices))]
        # 首日就买进股票
        dp[0][1] = -prices[0]
        for i in range(1, len(prices)):
            # 当前不持有<-max(昨天不持有，昨天持有今天卖出)
            dp[i][0] = max(dp[i-1][0], dp[i-1][1]+prices[i])
            # 前天不买
            val = dp[i-2][0] if i-2>=0 else 0
            # max(继续持有昨天股票，购买新股票)
            dp[i][1] = max(dp[i-1][1], val-prices[i])
        return dp[-1][0]
```

312. Burst Balloons

```
class Solution:
    def maxCoins(self, nums: List[int]) -> int:
        nums = [1] + nums + [1]
        dp = [[0] * len(nums) for _ in range(len(nums))]

        def compute(i, j):
            val_max = 0
            for k in range(i+1, j):
                # 注意这里因为k是最后一个ij区间里面爆的
                # 所以是nums[i]*nums[k]*nums[j]
                # 而不是nums[k-1]*nums[k]*nums[k+1]
                addition = nums[i]*nums[k]*nums[j]
                val_max = max(val_max, addition+dp[i][k]+dp[k][j])
            dp[i][j] = val_max

        # 外层循环区间长度，由于j-i<=1时dp是0，只需从长度为2开始即可
        for n in range(2, len(nums)):
            # 内层循环左端点位置，从0开始
            for i in range(len(nums)-n):
                compute(i, i+n) # 第n条对角线

        return dp[0][-1]
```

313. Super Ugly Number

```
class Solution:
    def nthSuperUglyNumber(self, n: int, primes: List[int]) -> int:
        # 多个指针的方法:
        # 新的丑数一定是已经存在的丑数和这些质数乘积的最小值
        # 如果这个最小值由某一个质数和当前这个质数还没乘的最小丑数的乘积得来
        # 那这个质数能乘的丑数就要到下一位了



        dp = [0] * (n + 1)
        m = len(primes)
        pointers = [0] * m
        nums = [1] * m # 对应质数能乘的最小丑数

        for i in range(1, n + 1):
            min_num = min(nums)
            dp[i] = min_num
            for j in range(m):
                if nums[j] == min_num:
                    pointers[j] += 1 # 丑数移到下一位
                    nums[j] = dp[pointers[j]] * primes[j]

        return dp[n]
"""
class Solution:
    def nthSuperUglyNumber(self, n: int, primes: List[int]) -> int:
        if n == 1:
            return 1
        heap = primes.copy()
        count = 1
        visited = set([1] + primes)
        while count < n:
            res = heapq.heappop(heap)
            count += 1
            for p in primes:
                nxt = p * res
                if nxt not in visited:
                    visited.add(nxt)
                    heapq.heappush(heap, nxt)
        return res
"""
```

322. Coin Change

```python
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        if amount == 0:
            return 0
        dp = [-1] * (amount + 1)
        dp[0] = 0
        for i in range(len(dp)):
            res = float("inf")
            for c in coins:
                # 注意这个和377区别，如果之前不可达，那么不能+1
                if i-c >= 0 and dp[i-c] != -1:
                    res = min(res, dp[i-c] + 1)
            if res != float("inf"):
                # 这里处理为了迎合-1的输出答案
                dp[i] = res
        return dp[-1]
```

377. Combination Sum IV

```python
class Solution:
    def combinationSum4(self, nums: List[int], target: int) -> int:
        # 和comb sum 1不一样，那个要unique
        dp = [0] * (target + 1)
        dp[0] = 1 # 这代表的是target为0有一种方案：什么都不做
        for i in range(len(dp)):
            for n in nums:
                if i-n >= 0:
                    dp[i] += dp[i-n]
        return dp[-1]
```

413. Arithmetic Slices

```
class Solution:
    def numberOfArithmeticSlices(self, A: List[int]) -> int:
        le = len(A)
        l = [0] * (le) # 有多少个等差以?结尾的
        for i in range(2, le):
            # 只需要考虑最近三个元素，然后根据上一轮的等差情况动态更新
            # [1,2,3,4]
            # [0,0,1,?]
            # 这个?就相当于是1+1，这个1就是自己+上一个+再上一个
            if A[i] - A[i-1] == A[i-1] - A[i-2]:
                l[i] = 1 + l[i-1]
        return sum(l)
```

509. Fibonacci Number

```
class Solution:
    def fib(self, n: int) -> int:
        if n == 0:
            return 0
        if n == 1:
            return 1
        a, b = 0, 1
        for _ in range(n-1):
            temp = a + b
            a = b
            b = temp
        return temp
```

714. Best Time to Buy and Sell Stock with Transaction Fee

```python
class Solution:
    def maxProfit(self, prices: List[int], fee: int) -> int:
        dp = [[0, -prices[0]] for i in range(len(prices))]
        for i in range(1, len(prices)):
            # 手续费在卖出的时候结算
            dp[i][0] = max(dp[i-1][1] + prices[i] - fee, dp[i-1][0])
            dp[i][1] = max(dp[i-1][0] - prices[i], dp[i-1][1])
        return dp[-1][0]
```

718. Maximum Length of Repeated Subarray

```python
class Solution:
    def findLength(self, nums1: List[int], nums2: List[int]) -> int:
        # dp (i, j) is max length end with i and j
        m, n = len(nums1), len(nums2)
        dp = [[0] * n for _ in range(m)]
        for i in range(m):
            dp[i][0] = int(nums1[i] == nums2[0])
        for i in range(n):
            dp[0][i] = int(nums1[0] == nums2[i])
        for i in range(1, m):
            for j in range(1, n):
                if nums1[i] == nums2[j]:
                    dp[i][j] = dp[i-1][j-1] + 1
                else:
                    dp[i][j] = 0
        return max(max(i) for i in dp) # not return [-1][-1]
```

740. Delete and Earn

```
class Solution:
    def deleteAndEarn(self, nums: List[int]) -> int:

        maxVal = max(nums)+1
        count = [0] * maxVal
        for val in nums:
            count[val] += val

        #相当于桶排序，先铺平，然后打家劫舍
        # dp[i][0] means max earn if not choose dp[i]
        # dp[i][1] means max earn if choose dp[i]
        dp = [0, 0]
        for i in range(1, maxVal):
            tmp = dp[0]
            dp[0] = max(dp)
            dp[1] = tmp + count[i]

        return max(dp)
```

746. Min Cost Climbing Stairs

```python
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        n = len(cost)
        if n == 1:
            return cost[0]
        if n == 2:
            return min(cost[0], cost[1])
        a, b, sum_cost = 0, 0, 0
        # 和斐波那契一样，a和b分别是1步之前和2步之前
        for i in range(2, n+1):
            # cost of jumping from x + cost of going to x
            sum_cost = min(cost[i-1] + b, cost[i-2] + a)
            a = b
            b = sum_cost
        return sum_cost
```

918. Maximum Sum Circular Subarray

```python
class Solution:
    def maxSubarraySumCircular(self, nums: List[int]) -> int:
        # when not across, just 53
        dp = nums[0]
        res_across = nums[0]
        for i in range(1, len(nums)):
            dp = dp + nums[i] if dp > 0 else nums[i]
            res_across = max(res_across, dp)

        # when across, then find min sum subarray
        dp = nums[0]
        res_no_across = nums[0]
        for i in range(1, len(nums)):
            dp = dp + nums[i] if dp < 0 else nums[i]
            res_no_across = min(res_no_across, dp)

        # special case when all elements are negative
        if all(i<0 for i in nums):
            return max(nums)

        return max(res_across, sum(nums)-res_no_across)
```

1014. Best Sightseeing Pair

```python
class Solution:
    def maxScoreSightseeingPair(self, values: List[int]) -> int:
        # for each new j, values[j] − j is fixed
        # then we only need to find out the pre_max value of value[i] + i
        pre_max = values.copy()
        for i in range(1, len(values)):
            pre_max[i] = max(pre_max[i] + i, pre_max[i-1])
        res = float("-inf")
        for j in range(1, len(values)):
            res = max(values[j] - j + pre_max[j-1], res)
        return res
```

1137. N-th Tribonacci Number

```python
class Solution:
    def tribonacci(self, n: int) -> int:
        if n == 0:
            return 0
        if n == 1:
            return 1
        if n == 2:
            return 1
        a, b, c = 0, 1, 1
        for _ in range(n-2):
            temp = a + b + c
            a = b
            b = c
            c = temp
        return temp
```

1137. N-th Tribonacci Number

```
class Solution:
    def getMaxLen(self, nums: List[int]) -> int:

        # 设 POS[i] 是以 nums[i] 结尾，乘积为正的最长子数组的长度。
        # 设 NEG[i] 是以 nums[i] 结尾，乘积为负的最长子数组的长度。
        # 每一步都根据当前数组来更新数组POS和NEG
        dp_neg = [1 if nums[0] < 0 else 0]
        dp_pos = [1 if nums[0] > 0 else 0]
        res = dp_pos[0]
        for i in range(1, len(nums)):
            cur = nums[i]
            cur_pos, cur_neg = dp_pos[-1], dp_neg[-1]
            if cur > 0:
                # pos depends on pos
                dp_pos.append(cur_pos + 1)
                # neg depends on neg
                dp_neg.append(0 if cur_neg == 0 else cur_neg + 1)
            elif cur < 0:
                # pos depends on neg
                dp_pos.append(0 if cur_neg == 0 else cur_neg + 1)
                # neg depends on pos
                dp_neg.append(cur_pos + 1)
            else:
                dp_pos.append(0)
                dp_neg.append(0)
            res = max(res, dp_pos[-1])
        return res
```

矩阵

62. Unique Paths

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[1] * n for _ in range(m)]
        for i in range(1, m):
            for j in range(1, n):
                dp[i][j] = dp[i-1][j] + dp[i][j-1]
        return dp[-1][-1]
```

63. Unique Paths II

```python
class Solution:
    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
        row, col = len(obstacleGrid), len(obstacleGrid[0])
        dp = [[0] * col for _ in range(row)]
        for i in range(row):
            for j in range(col):
                if obstacleGrid[i][j] == 1:
                    continue
                elif i == 0 or j == 0:
                    # fail for [[1,0]]
                    if i == 0 and j == 0:
                        dp[i][j] = 1
                    elif i == 0 and dp[0][j-1] != 0:
                        dp[i][j] = 1
                    elif j == 0 and dp[i-1][j] != 0:
                        dp[i][j] = 1
                else:
                    dp[i][j] = dp[i-1][j] + dp[i][j-1]
        return dp[-1][-1]
```

64. Minimum Path Sum

```python
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        r, c = len(grid), len(grid[0])
        dp = [[0] * c for _ in range(r)]
        for i in range(c):
            dp[0][i] = grid[0][0] if i==0 else grid[0][i] + dp[0][i-1]
        for j in range(1, r):
            dp[j][0] = grid[j][0] + dp[j-1][0]
        for i in range(1, r):
            for j in range(1, c):
                dp[i][j] = grid[i][j] + min(dp[i-1][j], dp[i][j-1])
        return dp[-1][-1]
```

120. Triangle

```
class Solution:
    def minimumTotal(self, triangle: List[List[int]]) -> int:
        dp = [0]
        while triangle:
            cur = triangle.pop(0)
            for i in range(len(cur)):
                cur[i] += min(dp[min(len(dp)-1, i)], dp[max(0, i-1)])
            dp = cur
        return min(dp)
```

174. Dungeon Game

```python
class Solution:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
        # 反向dp，每个dp[i][j]代表从当前走到右下角的最少初始血量
        m, n = len(dungeon), len(dungeon[0])
        dp = [[0] * n for _ in range(m)]
        for i in range(m-1, -1, -1):
            for j in range(n-1, -1, -1):
                d = dungeon[i][j]
                if i == m-1 and j == n-1:
                    # 第一个1相当于仿佛外面还有格子，至少血量初始为1
                    dp[i][j] = max(1 - d, 1)
                elif i == m-1:
                    dp[i][j] = max(dp[i][j+1] - d, 1)
                elif j == n-1:
                    dp[i][j] = max(dp[i+1][j] - d, 1)
                else:
                    _min = min(dp[i+1][j], dp[i][j+1])
                    dp[i][j] = max(_min - d, 1) # 初始血量至少为1
        return dp[0][0]
```

221. Maximal Square

```python
class Solution:
    def maximalSquare(self, matrix: List[List[str]]) -> int:
        """
        check the status of left, up and corner
        """
        r, c = len(matrix), len(matrix[0])
        matrix = [[int(matrix[i][j]) for j in range(c)] for i in range(r)]
        for i in range(1, r):
            for j in range(1, c):
                if matrix[i][j] == 0:
                    continue
                matrix[i][j] = 1 + min(
                    matrix[i-1][j-1],
                    matrix[i-1][j],
                    matrix[i][j-1],
                )
        return max(max(i) ** 2 for i in matrix)
```

329. Longest Increasing Path in a Matrix

```
class Solution:
    def longestIncreasingPath(self, matrix: List[List[int]]) -> int:
        row, col = len(matrix), len(matrix[0])
        # record whether visited (−1 if visited) 备忘录
        dp = [[-1] * col for i in range(row)]
        def dfs(r, c, v):
            # border condition
            if (r<0 or r>=row or c<0 or c>=col):
                return 0
            # v is the value from last position
            # then if m[r][c] <= v, we should
            # not make addition to the path length
            if matrix[r][c] <= v:
                return 0
            # if visited, directly use existing value
            if dp[r][c] != -1:
                return dp[r][c]
            res = 1
            res = max(res, 1 + dfs(r-1, c, matrix[r][c]))
            res = max(res, 1 + dfs(r+1, c, matrix[r][c]))
            res = max(res, 1 + dfs(r, c-1, matrix[r][c]))
            res = max(res, 1 + dfs(r, c+1, matrix[r][c]))
            # here record the cur res
            dp[r][c] = res
            return res
        for i in range(row):
            for j in range(col):
                dfs(i, j, -1)
        return max(max(dp[i]) for i in range(row))
```

562. Longest Line of Consecutive One in Matrix

```
class Solution:
    def longestLine(self, mat: List[List[int]]) -> int:
        res = 0
        m, n = len(mat), len(mat[0])
        # horizontal, vertical, diagonal, or anti−diagonal
        info = [[[0, 0, 0, 0] for j in range(n)] for i in range(m)]
        for i in range(m):
            for j in range(n):
                if mat[i][j]:
                    if i == 0:
                        if j == 0:
                            info[i][j] = [1] * 4
                        else:
                            info[i][j][0] = info[i][j-1][0] + 1
                            info[i][j][1] = 1
                            info[i][j][2] = 1
                            info[i][j][3] = 1
                    else:
                        info[i][j][0] = 1
                            if j == 0 else (info[i][j-1][0] + 1)
                        info[i][j][2] = 1
                            if j == 0 else (info[i-1][j-1][2] + 1)
                        info[i][j][1] = info[i-1][j][1] + 1
                        info[i][j][3] = 1
                            if j == n-1 else (info[i-1][j+1][3] + 1)
                    res = max(res, max(info[i][j]))
        return res
```

931. Minimum Falling Path Sum

```
class Solution:
    def minFallingPathSum(self, matrix: List[List[int]]) -> int:
        dp = [0] * len(matrix[0])
        while matrix:
            cur = matrix.pop(0)
            for i in range(len(cur)):
                cur[i] += min(dp[i], dp[min(len(cur)-1, i+1)], dp[max(i-1, 0)]
            dp = cur
        return min(dp)
```

1937. Maximum Number of Points with Cost

```python
class Solution:
    def maxPoints(self, points: List[List[int]]) -> int:
        # dp[i][j] = max(dp[i-1][k] - |j-k|) + points[i][j]
        # 分类讨论:
        # 1) 当j>=k时，max(dp[i-1][k] + k) + points[i][j] - j
        # 2) 当j <k时，max(dp[i-1][k] - k) + points[i][j] + j
        # 由于两个max可以分别在O(n)得到，因此整体的复杂度优化到O(mn)
        m, n = len(points), len(points[0])
        dp_cur = points[0][:]
        dp_last = dp_cur[:] # dp转移只跟上一行有关

        for i in range(1, m):
            # 从左往右
            left_max = float("-inf")
            for j in range(n):
                left_max = max(left_max, dp_last[j] + j)
                dp_cur[j] = left_max + points[i][j] - j
            # 从右往左
            right_max = float("-inf")
            for j in range(n-1, -1, -1):
                right_max = max(right_max, dp_last[j] - j)
                dp_cur[j] = max(dp_cur[j], right_max + points[i][j] + j)
            dp_last = dp_cur[:]
        return max(dp_cur)
```

字符串

10. Regular Expression Matching

```
class Solution:
    def isMatch(self, s: str, p: str) -> bool:

        s_len, p_len = len(s), len(p)
        dp = [[False] * (p_len+1) for _ in range(s_len+1)]
        dp[0][0] = True

        for i in range(p_len):
            if p[i] == "*" and i % 2 == 1:
                dp[0][i+1] = dp[0][i-1]

        for i in range(s_len):
            for j in range(p_len):
                if s[i] == p[j] or p[j] == ".":
                    dp[i+1][j+1] = dp[i][j]
                elif p[j] == "*":
                    if s[i] == p[j-1] or p[j-1] == ".":
                        # * 匹配0次 * 匹配1次 * 多次
                        dp[i+1][j+1] = dp[i+1][j-1] or dp[i][j-1] or dp[i][j+1]
                    else:
                        dp[i+1][j+1] = dp[i+1][j-1]

        return dp[-1][-1]
```

44. Wildcard Matching

```
class Solution:
    def isMatch(self, s: str, p: str) -> bool:

        s_len, p_len = len(s), len(p)
        dp = [[False] * (p_len+1) for _ in range(s_len+1)]
        dp[0][0] = True

        for i in range(s_len):
            dp[i+1][0] = False
        for i in range(p_len):
            if p[i] == "*":
                dp[0][i+1] = True
            else:
                break

        for i in range(s_len):
            for j in range(p_len):
                # 默认情况是False，如果True则一定被改了
                $ 而且下面一格也要传播下去是True
                if dp[i+1][j+1] == True:
                    if i+2 < s_len+1:
                        dp[i+2][j+1] = True
                if s[i] == p[j] or p[j] == "?":
                    dp[i+1][j+1] = dp[i][j]
                elif p[j] == "*":
                    # 和第10题一样，分成0次1次多次讨论
                    if dp[i+1][j] or dp[i][j] or dp[i][j+1]:
                        dp[i+1][j+1] = True
                        if i+2 < s_len+1:
                            # 如果下面还有元素，那么一直往下匹配*
                            dp[i+2][j+1] = True

        return dp[-1][-1]
```

72. Edit Distance

```python
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        # dp[i][j] 代表 word1 到 i 位置转换成 word2 到 j 位置需要最少步数
        # word1[i] == word2[j], dp[i][j] = dp[i-1][j-1]
        # word1[i] != word2[j], dp[i][j] =
        #   min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1]) + 1
        # 其中
        # dp[i-1][j-1] 表示替换操作
        # dp[i-1][j] 表示删除操作
        # dp[i][j-1] 表示插入操作。
        m, n = len(word1), len(word2)
        if m == 0:
            return n
        if n == 0:
            return m
        dp = [[0] * (n+1) for _ in range(m+1)]
        for i in range(n+1):
            dp[0][i] = i
        for i in range(m+1):
            dp[i][0] = i
        for i in range(1, m+1):
            for j in range(1, n+1):
                if word1[i-1] == word2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                else:
                    dp[i][j] = min(
                        dp[i-1][j-1],
                        dp[i][j-1],
                        dp[i-1][j]) + 1
        return dp[-1][-1]
```

87. Scramble String

```python
class Solution:
    def isScramble(self, s1: str, s2: str) -> bool:
        # dp[i][j][len]表示从字符串S中i开始长度为len的字符串
        # 是否能变换为从字符串T中j开始长度为len的字符串
        # 边界条件: 对于长度是1的子串，只有相等才能变过去
        # 相等为true，不相等为false
        # 转移方程讨论两种情况，S1S2->T1T2 S1S2->T2T1
        n = len(s1)
        dp = [
            [[False for length in range(n+1)] for j in range(n)]
            for i in range(n)
        ]
        for i in range(n):
            for j in range(n):
                if s1[i] == s2[j]:
                    dp[i][j][1] = True
        # 枚举区间长度2-n
        for length in range(2, n+1):
            for i in range(n-length+1):
                for j in range(n-length+1):
                    # k代表切割位置 [1,k] [k+1,length], k=1,...,length-1
                    for k in range(1, length):
                        if dp[i][j][k] and dp[i+k][j+k][length-k]:
                            dp[i][j][length] = True
                            break
                        if dp[i][j+length-k][k] and dp[i+k][j][length-k]:
                            dp[i][j][length] = True
                            break
        return dp[0][0][n]
```

91. Decode Ways

```python
class Solution:
    def numDecodings(self, s: str) -> int:
        if len(s) == 0:
            return 0
        if s[0] == "0":
            return 0
        dp = [1] * len(s)
        for i in range(1, len(s)):
            # 首先讨论当前为0
            if s[i] == "0":
                # 如果这样是不可能的，只会有10和20
                if s[i-1] not in ["1", "2"]:
                    return 0
                # 如果是开头，要单独讨论
                if i == 1:
                    dp[i] = 1
                else:
                    dp[i] = dp[i-2]
            # 此处已经排除10和20的可能
            elif s[i-1:i+1] >= "10" and s[i-1:i+1] <= "26":
                if i == 1:
                    dp[i] = 2
                else:
                    dp[i] = dp[i-1] + dp[i-2]
            else:
                # number of 1-9 but s[i-1:i+1] can't be decoded
                dp[i] = dp[i-1]
        return dp[-1]
        """
        总结:
        1. 首先讨论0
        2. 接着讨论10和26之间
        3. 最后1-9
        4. 注意2和3需要对i是否==1讨论
        """
```

97. Interleaving String

```python
class Solution:
    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:

        # dp[i][j]表示用前i个s1和前j个s2能否表示前i+j个s3
        len1, len2, len3 = len(s1), len(s2), len(s3)
        if len1 + len2 != len3:
            return False

        dp = [[False] * (len2+1) for _ in range(len1+1)]
        dp[0][0] = True

        for i in range(len1):
            dp[i+1][0] = dp[i][0] and s1[i]==s3[i]
        for i in range(len2):
            dp[0][i+1] = dp[0][i] and s2[i]==s3[i]
        for i in range(len1):
            for j in range(len2):
                dp[i+1][j+1] = (
                    dp[i][j+1]
                    and s1[i]==s3[i+j+1]
                ) or (
                    dp[i+1][j]
                    and s2[j]==s3[i+j+1]
                )

        return dp[-1][-1]
```

115. Distinct Subsequences

```python
class Solution:
    def numDistinct(self, s: str, t: str) -> int:
        # 经典动态规划
        # dp[i][j]是s[i:]和t[j:]的匹配数
        # 1) 如果s[i]==t[j]
        # a) 当前直接匹配 dp[i+1][j+1]
        # b) 当前不匹配 dp[i+1][j]
        # 2) 反之
        # 只可能是dp[i+1][j]

        # 最后要考虑空串，因此是m+1和n+1

        m, n = len(s), len(t)
        dp = [[0] * (n+1) for _ in range(m+1)]
        for i in range(m+1):
            dp[i][-1] = 1

        for i in range(m-1, -1, -1):
            for j in range(n-1, -1, -1):
                if s[i] == t[j]:
                    dp[i][j] = dp[i+1][j+1] + dp[i+1][j]
                else:
                    dp[i][j] = dp[i+1][j]

        return dp[0][0]
```

131. Palindrome Partitioning

```python
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        # 思路：先预处理，获得ij切片是不是回文串，然后dfs逐个切片
        n = len(s)
        dp = [[False] * n for _ in range(n)]
        for i in range(n):
            dp[i][i] = True
        for i in range(n-1, -1, -1):
            for j in range(i+1, n):
                dp[i][j] = (dp[i+1][j-1] or i+1 >= j-1) & (s[i]==s[j])
        res = []
        part = []
        def dfs(i):
            if i == n:
                res.append(part[:])
                return # 不要忘记return
            for j in range(i, n):
                if dp[i][j]:
                    part.append(s[i:j+1])
                    dfs(j+1) # 不是i+1
                    part.pop(-1)
        dfs(0)
        return res
```

139. Word Break

```python
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        # dp[i] means s[0:i] is valid
        n = len(s)
        dp = [False] * (n+1)
        dp[0] = True # empty string is valid
        for i in range(n):
            for j in range(i+1, n+1):
                # new word is starting from i
                if dp[i] and s[i:j] in wordDict:
                    dp[j] = True
        return dp[-1]
```

140. Word Break II

```python
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
        res = []
        memo = [1 for i in range(len(s)+1)]
        def dfs(arr, pos):
            num = len(res)
            if pos == len(s): # at the position after the last char
                res.append(" ".join(arr))
                return
            for i in range(pos+1, len(s)+1):
                # currently, we have substring pos:i
                # but if we find s[i:] is invalid
                # then we can cut off
                if memo[i] and s[pos:i] in wordDict:
                    arr.append(s[pos:i])
                    dfs(arr, i)
                    arr.pop(-1)
            # for the current level, res not increase
            # thus s[pos:] is not valid
            memo[pos] = 1 if len(res) > num else 0
        dfs([], 0)
        return res
```

516. Longest Palindromic Subsequence

```python
class Solution:
    def longestPalindromeSubseq(self, s: str) -> int:
        """
        for any pair (i, j), note dp[i][j] is the length longest
        palindromic subseq, then if s[i] == s[j], it should be
        dp[i+1][j-1] + 2, else should be max(dp[i][j-1], dp[i+1][j])

        here dp is a upper triangle matrix
        """
        n = len(s)
        dp = [[0] * n for _ in range(n)]
        for i in range(n):
            dp[i][i] = 1
        # iter from bottom to top and left to right
        for i in range(n-2, -1, -1):
            for j in range(i+1, n):
                if s[i] == s[j]:
                    dp[i][j] = dp[i+1][j-1] + 2
                else:
                    dp[i][j] = max(dp[i+1][j], dp[i][j-1])
        return dp[0][-1]
```

552. Student Attendance Record II

```python
class Solution:
    def checkRecord(self, n: int) -> int:
        # 考虑A的数量，与字符串结尾处L的数量，一共6种状态
        # (A=0,结尾处连续L=0),(0,1),(0,2),(1,0),(1,1),(1,2)
        a, b, c, d, e, f = 1, 0, 0, 0, 0, 0
        for i in range(n):
            a, b, c, d, e, f = a+b+c, a, b, a+b+c+d+e+f, d, e
            a %= 1e9 + 7
            b %= 1e9 + 7
            c %= 1e9 + 7
            d %= 1e9 + 7
            e %= 1e9 + 7
            f %= 1e9 + 7
        return int((a+b+c+d+e+f) % (1e9 + 7))
```

650. 2 Keys Keyboard

```python
class Solution:
    def minSteps(self, n: int) -> int:
        # 定义dp[i][j]为经过最后一次操作后，当前记事本上有i个字符
        # 粘贴板上有j个字符的最小操作次数

        # 另外分解因数的想法见题解链接
        dp = [[float("inf")] * (n+1) for i in range(n+1)]
        dp[1][0] = 0 # 啥也不干
        dp[1][1] = 1 # 复制了一个"A"
        for i in range(2, n+1):
            min_val = float("inf")
            # 注意paste只能由比自己长度一般的拷贝过来
            for j in range(1, n//2+1): # 如果是paste，i=0赋值0个字符无意义
                dp[i][j] = dp[i-j][j] + 1 # paste # 粘贴板不变，记事本变
                min_val = min(min_val, dp[i][j])
            # 准备自我复制
            dp[i][i] = min_val + 1
        return min(dp[-1])
```

828. Count Unique Characters of All Substrings of a Given String

```
class Solution:
    def uniqueLetterString(self, s: str) -> int:
        """
        X X X A [X X A X X] A X
              j i k
        当前的字符为总分贡献多少，其实就是统计有多少包含当前字符的无重复子串
        (i − j) ∗ (k − i)
        # 用链表穿起来
        A: −1 −> idx −> idx −> ... −> n
        B: ...
        C: ...
        ...
        Z: ...
        """
        letters = [[-1] for i in range(26)]
        for i, c in enumerate(s):
            idx = ord(c) - ord("A")
            letters[idx].append(i)
        for i in range(26):
            letters[i].append(len(s))
        res = 0
        for i in range(26):
            for j in range(1, len(letters[i])-1):
                res += (letters[i][j] - letters[i][j-1])
                    * (letters[i][j+1] - letters[i][j])
        return res
```

1048. Longest String Chain

```
class Solution:
    def longestStrChain(self, words: List[str]) -> int:
        """
        Sort the words by word's length. (also can apply bucket sort)
        For each word, loop on all possible previous word with 1 letter missing.
        If we have seen this previous word, update the longest chain for the current word.
        Finally return the longest word chain.
        """
        dp = dict()
        for w in sorted(words, key=len):
            dp[w] = max(dp.get(w[:i]+w[i+1:], 0)+1 for i in range(len(w)))
        return max(dp.values())
```

1143. Longest Common Subsequence

```python
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        m, n = len(text1), len(text2)
        dp = [[0] * n for _ in range(m)]
        # be careful about the init, remember dp means :i+1 and :j+1 LCS
        for i in range(m):
            dp[i][0] = int(text2[0] == text1[i])
            if i >= 1 and dp[i-1][0] == 1:
                dp[i][0] = 1
        for i in range(n):
            dp[0][i] = int(text1[0] == text2[i])
            if i >= 1 and dp[0][i-1] == 1:
                dp[0][i] = 1
        for i in range(1, m):
            for j in range(1, n):
                if text1[i] == text2[j]:
                    dp[i][j] = dp[i-1][j-1] + 1
                else:
                    dp[i][j] = max(dp[i-1][j], dp[i][j-1])
        return dp[-1][-1]
```

1216. Valid Palindrome III

```
class Solution:
    def isValidPalindrome(self, s: str, k: int) -> bool:
        # 和516一模一样...只需修改结果
        # 本质上就是找最长回文子序列
        n = len(s)
        dp = [[0] * n for _ in range(n)]
        for i in range(n):
            dp[i][i] = 1
        # iter from bottom to top and left to right
        for i in range(n-2, -1, -1):
            for j in range(i+1, n):
                if s[i] == s[j]:
                    dp[i][j] = dp[i+1][j-1] + 2
                else:
                    dp[i][j] = max(dp[i+1][j], dp[i][j-1])
        return len(s) - dp[0][-1] <= k
```

1653. Minimum Deletions to Make String Balanced

```python
class Solution:
    def minimumDeletions(self, s: str) -> int:
        # dp表示之前子串需要的最少次数
        # 这个时候如果新来一个是"a"而且要保留的，那么要删掉前面所有的b
        # 因此需要统计当前之前所有b的数量
        count_b, dp = 0, 0
        for c in s:
            if c == "a":
                dp = min(count_b, dp+1) # min(保留a, 不保留a)
            else:
                count_b += 1
        return dp
```

2262. Total Appeal of A String

```
class Solution:
    def appealSum(self, s: str) -> int:
        # 和828思路完全一样
        # 只计算每个字符对子串的贡献得分
        # 如果"**a***a**"，那么规定贡献的只会是最左边那个
        # 所以按照这个定义，每个字符选取的右边界可以一直到最右边，
        # 选取的左边界到上一个相同字符位置
        last_seen = {i:-1 for i in "abcdefghijklnmopqrstuvwxyz"}
        left_bound = [-1] * len(s)
        for i in range(len(s)):
            left_bound[i] = last_seen[s[i]]
            last_seen[s[i]] = i
        res = 0
        for i in range(len(s)):
            res += (i - left_bound[i]) * (len(s) - i)
        return res
```

2272. Substring With Largest Variance

```python
class Solution:
    def largestVariance(self, s: str) -> int:
        res = 0
        chars = set(s)
        for a in chars:
            for b in chars:
                if b == a:
                    continue
                diff, diff_with_b = 0, -inf
                for ch in s:
                    if ch == a:
                        diff += 1
                        diff_with_b += 1
                    elif ch == b:
                        diff -= 1
                        diff_with_b = diff # 记录包含 b 时的 diff
                        if diff < 0:
                            diff = 0 # 相当于直接把b扔了从头开始算
                    # 其实等价于当前字符结尾做一个dp
                    if diff_with_b > res:
                        res = diff_with_b
        return res
```

```
"""
class Solution:
    def largestVariance(self, s: str) -> int:
        # 从数据量上考察，应当是常数倍的线性复杂度
        # 考虑两层循环，对所有26*26个字母pair进行循环
        # 如果把a看做1，b看做-1，其他字母当做0
        # 实际上找的就是最大子数组和，"aababbb"->1,1,-1,1,-1,-1,-1
        # 但是这里一个特殊的地方在于子序列必须要同时包含两个字母
        # 所以直接考虑包含-1的最大sub arr的sum
        # 原因在于最后遍历统计最大值时，因为1一定会存在
        # 所以不用担心这个-1或者负数会成为一个不合法的结果
        chars = set(s)
        res = 0
        dp1 = [0] * len(s)
        dp2 = [0] * len(s)
        arr = [0] * len(s)
        for c1 in chars:
            for c2 in chars:
                if c1 != c2:
                    for i in range(len(s)):
                        if s[i] == c1:
                            arr[i] = 1
                        elif s[i] == c2:
                            arr[i] = -1
                        else:
                            arr[i] = 0
                    # 以i结尾的最大值dp
                    for i in range(len(arr)):
                        if i == 0:
                            dp1[i] = arr[i]
                        else:
                            dp1[i] = max(dp1[i-1] + arr[i], arr[i])
                    # 以i开头的最大值dp
                    for i in range(len(arr)-1, -1, -1):
                        if i == len(arr)-1:
                            dp2[i] = arr[len(arr)-1]
                        else:
                            dp2[i] = max(dp2[i+1] + arr[i], arr[i])
                    # 这里两侧相加再扣除当中的重合部分
                    for i in range(len(arr)):
                        if arr[i] == -1:
                            val = dp1[i] + dp2[i] - arr[i]
                            res = max(res, val)
        return res
"""
```

其他

337. House Robber III

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
    def rob(self, root: Optional[TreeNode]) -> int:
        return max(self.dfs(root))
    def dfs(self, root):
        if root is None:
            return (0, 0)
        left = self.dfs(root.left)
        right = self.dfs(root.right)
        choose_cur = root.val + left[1] + right[1]
        skip_cur = max(left) + max(right)
        return (choose_cur, skip_cur)
```

375. Guess Number Higher or Lower II

```python
class Solution:
    def getMoneyAmount(self, n: int) -> int:
        # dp就是要预留的钱，答案是dp[1][n]
        # 思路：对于i到j的范围，里面选了一个k
        # 当前的代价就是k+max(dp[i][k-1], dp[k+1][j])
        # 遍历k=i...j，使得代价最小化
        # 注意dp矩阵在i>=j时都为0，是一个上三角
        dp = [[0] * (n+1) for _ in range(n+1)]
        for i in range(n-1, 0, -1):
            for j in range(i+1, n+1):
                # 先把最后一个k给赋上去，一方面避免越界
                # 一方面自动把需要min的初始化了
                dp[i][j] = j + dp[i][j-1]
                for k in range(i, j):
                    dp[i][j] = min(
                        dp[i][j],
                        k + max(dp[i][k-1], dp[k+1][j]),
                    )
        return dp[1][n]
```

600. Non-negative Integers without Consecutive Ones

```
class Solution:
    def findIntegers(self, n: int) -> int:

        dp = [0] * 32
        dp[0], dp[1] = 1, 2
        # 给定n位数******，有多少含有11?
        # 0*****
        # 10****
        for i in range(2, 32):
            dp[i] = dp[i-1] + dp[i-2]

        k = 31
        pre = 0
        res = 0
        while k >= 0:
            if n & (1<<k) != 0: # 当前位为1
                # 1 (000000)
                # 1 (111111) -> dp[6]
                res += dp[k]
                if pre == 1:
                    return res
                pre = 1
            else:
                pre = 0
            k -= 1
        return res + 1 # 包括自己
```

数据结构

栈与队列

20. Valid Parentheses

```python
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        p_dict = dict(zip(list(")]}"), list("([{")))
        for i in s:
            if i in ["(", "[", "{"]:
                stack.append(i)
            elif i in [")", "]", "}"]:
                if stack == []:
                    return False
                else:
                    temp = stack.pop(-1)
                    if temp != p_dict[i]:
                        return False
        return stack == []
```

32. Longest Valid Parentheses

```
"""
用栈模拟一遍，将所有可以匹配的括号的位置全部置0
例如: "()(()"的mark为[1, 1, 0, 1, 1]
再例如: ")()((()))"的mark为[0, 1, 1, 0, 1, 1, 1, 1]
经过这样的处理后, 此题就变成了寻找最长的连续的1的长度
"""

class Solution:
    def longestValidParentheses(self, s: str) -> int:
        stack = []
        dp = [0] * len(s)
        for i, c in enumerate(s):
            if c == "(":
                stack.append((c, i))
            else:
                if stack:
                    dp[stack[-1][1]] = 1
                    dp[i] = 1
                    stack.pop(-1)
        count, res = 0, 0
        for val in dp:
            if val:
                count += 1
            else:
                res = max(res, count)
                count = 0
        return max(res, count)
```

71. Simplify Path

```python
class Solution:
    def simplifyPath(self, path: str) -> str:
        path = path.split("/")
        stack = []
        while path:
            cur = path.pop(0)
            if cur == "." or cur == "":
                continue
            elif stack and cur == "..":
                stack.pop(-1)
                continue
            elif stack == [] and cur == "..":
                continue
            else:
                stack.append(cur)
        return "/" + "/".join(stack)
```

150. Evaluate Reverse Polish Notation

```
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        stack = []
        for i in tokens:
            if i in list("+-*/"):
                a = int(stack.pop(-1))
                b = int(stack.pop(-1))
                if i == "+":
                    stack.append(b+a)
                elif i == "-":
                    stack.append(b-a)
                elif i == "*":
                    stack.append(b*a)
                elif i == "/":
                    # b // a is wrong when negative
                    stack.append(int(b/a))
            else:
                stack.append(i)
        return stack[-1]
```

155. Min Stack

```python
class MinStack:

    def __init__(self):
        # 两个栈长度相同，第二个栈总是压当前的最小值
        # 即第一个栈的新元素和第二个栈的栈顶对比
        # 如果大了，那么继续压第二个栈的栈顶元素，否则
        # 压第一个栈的元素
        # plus: 如果用元组的话，可以只用一个栈
        self.s1 = [] # 5 6 3 4 1 2 <-
        self.s2 = [] # 5 5 3 3 1 1 <-

    def push(self, val: int) -> None:
        self.s1.append(val)
        if len(self.s2) == 0:
            self.s2.append(val)
        else:
            self.s2.append(min(self.s2[-1], val))

    def pop(self) -> None:
        if self.s1:
            self.s2.pop(-1)
            return self.s1.pop(-1)

    def top(self) -> int:
        return self.s1[-1]

    def getMin(self) -> int:
        return self.s2[-1]
```

224. Basic Calculator

```python
class Solution:
    def calculate(self, s: str) -> int:
        # 基本思路是把所有的展开，每个元素根据之前各个括号前的符号来判断
        res = 0
        # 假设外面是一个大括号
        # 每个元素代表各个左括号前面的符号
        stack = [1]
        sign = 1
        i = 0

        while i < len(s):
            c = s[i]
            if c == " ":
                i += 1
            # 左括号把上一个符号加进去
            elif c == "(":
                stack.append(sign)
                i += 1
            # 右括号弹出
            elif c == ")":
                stack.pop(-1)
                i += 1
            # 当前的符号取决于上一个括号那里的状态
            elif c == "+":
                sign = stack[-1]
                i += 1
            elif c == "-":
                sign = -stack[-1]
                i += 1
            elif c.isdigit():
                temp = ""
                while i<len(s) and s[i].isdigit():
                    temp += s[i]
                    i += 1
                res += sign * int(temp)
        return res
```

227. Basic Calculator II

```python
class Solution:
    def calculate(self, s: str) -> int:
        stack = []
        pre_op = "+"
        num = 0
        for i, c in enumerate(s):
            if c.isdigit():
                num = 10*num + int(c)
            if i==len(s)-1 or c in "+-*/":
                if pre_op == "+":
                    stack.append(num)
                elif pre_op == "-":
                    stack.append(-num)
                elif pre_op == "*":
                    stack.append(stack.pop() * num)
                elif pre_op == "/":
                    # 这个时候num一定是一个正的
                    val = stack.pop(-1)
                    if val < 0:
                        stack.append(int(val / num))
                    else:
                        stack.append(val // num)
                pre_op = c
                num = 0
        return sum(stack)
```

225. Implement Stack using Queues

```
class MyStack:

    def __init__(self):
        self.q = []

    def push(self, x: int) -> None:
        self.q.append(x)

    def pop(self) -> int:
        # 前面出来的直接塞到后面去
        n = len(self.q)
        while n > 1:
            val = self.q.pop(0)
            self.q.append(val)
            n -= 1
        return self.q.pop(0)

    def top(self) -> int:
        if self.empty():
            return None
        return self.q[-1]

    def empty(self) -> bool:
        return len(self.q) == 0
```

232. Implement Queue using Stacks

```
class MyQueue:

    def __init__(self):
        self.s1 = []
        self.s2 = []

    def push(self, x: int) -> None:
        self.s1.append(x)

    def pop(self) -> int:
        # 把s1东西反过来倒入s2里面，再反过来倒入s1
        while self.s1:
            self.s2.append(self.s1.pop(-1))
        res = self.s2.pop(-1)
        while self.s2:
            self.s1.append(self.s2.pop(-1))
        return res

    def peek(self) -> int:
        return self.s1[0]

    def empty(self) -> bool:
        return len(self.s1) == 0
```

239. Sliding Window Maximum

```
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:

        # 队列中存放了元素的下标，目的是方便检查是否超出窗口
        # 思想：对每次新来的元素，首先弹出已经在窗口外的索引
        # 然后弹出所有比它小的元素（对应索引），从而保持queue单调性

        queue = []
        res = []
        for i in range(len(nums)):
            while len(queue) != 0 and queue[0] <= i-k:
                queue.pop(0)
            while len(queue) != 0 and nums[queue[-1]] < nums[i]:
                queue.pop(-1)
            queue.append(i)
            if i >= k-1:
                res.append(nums[queue[0]])
        return res
```

341. Flatten Nested List Iterator

```
# 初始化时，逆序把元素放进栈
# next时，弹出栈顶
# hasNext时：
# 如果顶是int，那么true
# 如果顶是list，那么展开直到int
# 如果栈空了则false

class NestedIterator:
    def __init__(self, nestedList: [NestedInteger]):
        self.stack = nestedList[::-1]

    def next(self) -> int:
        return self.stack.pop(-1).getInteger()

    def hasNext(self) -> bool:
        while self.stack:
            if self.stack[-1].isInteger():
                return True
            cur = self.stack.pop(-1)
            self.stack += cur.getList()[::-1]
        return False
```

394. Decode String

```python
class Solution:
    def decodeString(self, s: str) -> str:

        stack_str = []
        stack_num = []
        cur_str = ""
        cur_num = ""

        for i in s:
            if i.isdigit():
                cur_num += i
            elif i.isalpha():
                cur_str += i
            elif i == "[":
                # when "[", a new record for cur_str should start
                # thus need to append chars before "[" for later
                # concat, and record the duplicating times.
                # e.g. 3[a2[c]]
                stack_num.append(int(cur_num))
                stack_str.append(cur_str)
                cur_num = ""
                cur_str = ""
            elif i == "]":
                # cur_str holds when "]", not into stack, but updated
                cur_str = stack_str.pop(-1) + cur_str * stack_num.pop(-1)

        return cur_str
```

362. Design Hit Counter

```
class HitCounter:

    def __init__(self):
        self.q = [] # 队列

    def hit(self, timestamp: int) -> None:
        self.q.append(timestamp)

    def getHits(self, timestamp: int) -> int:
        while self.q and timestamp - self.q[0] >= 300:
            self.q.pop(0)
        return len(self.q)
```

636. Exclusive Time of Functions

```
class Solution:
    def exclusiveTime(self, n: int, logs: List[str]) -> List[int]:
        stack = []
        res = [0] * n
        def parse(s):
            s = s.split(":")
            return int(s[0]), s[1], int(s[2])
        for s in logs:
            s_id, statue, pos = parse(s)
            if statue == "start":
                stack.append((s_id, pos))
            else:
                _, start = stack.pop(-1)
                res[s_id] += pos - start + 1
                if stack:
                    # [1 2 [3 4 5] 6 7] 在里面那层左括号弹出来结算
                    # 的时候，外面的需要扣掉这段时间
                    res[stack[-1][0]] -= pos - start + 1
        return res
```

678. Valid Parenthesis String

```python
class Solution:
    def checkValidString(self, s: str) -> bool:
        left_stack = []
        star_stack = []
        for i, c in enumerate(s):
            if c == "(":
                left_stack.append(i)
            elif c == "*":
                star_stack.append(i)
            else:
                if left_stack:
                    left_stack.pop(-1)
                else:
                    if star_stack:
                        star_stack.pop(-1)
                    else:
                        return False
        while left_stack and star_stack:
            left = left_stack.pop(-1)
            start = star_stack.pop(-1)
            if left > start:
                return False
        return len(left_stack) == 0
```

726. Number of Atoms

```
class Solution:
    def countOfAtoms(self, formula):
        dic = collections.defaultdict(int)
        coeff, stack, elem, cnt, i = 1, [], "", 0, 0
        # coeff表示字母次数乘以的所有括号累计权重
        # cnt表示当前字母后面数字
        # i记录数字的位数
        for c in formula[::-1]:
            if c.isdigit():
                cnt += int(c) * (10 ** i)
                i += 1
            elif c == ")":
                if cnt:
                    stack.append(cnt)
                    coeff *= cnt
                else:
                    stack.append(1)
                i = cnt = 0
            elif c == "(":
                coeff //= stack.pop()
                i = cnt = 0
            elif c.isupper():
                elem += c
                dic[elem[::-1]] += (cnt or 1) * coeff
                elem = ""
                i = cnt = 0
            elif c.islower():
                elem += c
        return "".join(
            k + str(v > 1 and v or "")
            for k, v in sorted(dic.items())
        )
```

735. Asteroid Collision

```python
class Solution:
    def asteroidCollision(self, asteroids: List[int]) -> List[int]:
        # 当前新来一个元素要不要加?
        # 1) 栈空
        # 2) 栈顶是负的
        # 3) 新来的是正的
        s, p = [], 0
        while p < len(asteroids):
            if len(s) == 0 or s[-1] < 0 or asteroids[p] > 0:
                s.append(asteroids[p])
                p += 1
            else: # 栈顶是正的，新来是负的，碰撞
                val = -asteroids[p]
                if s[-1] > val:
                    p += 1
                elif s[-1] == val:
                    s.pop(-1)
                    p += 1
                else:
                    s.pop(-1)
                    continue
        return s
```

772. Basic Calculator III

```python
class Solution:
    def calculate(self, s: str):
        nums, ops = [], [] # 和1597一模一样
        def merge_nums():
            op = ops.pop()
            right = nums.pop()
            left = nums.pop()
            if op == "+":
                nums.append(left + right)
            elif op == "-":
                nums.append(left - right)
            elif op == "*":
                nums.append(left * right)
            elif op == "/":
                neg = (left > 0 and right < 0) or (left < 0 and right > 0)
                res = abs(left) // abs(right)
                nums.append(-res if neg else res)
        c = ""
        i = 0
        while i < len(s):
            if s[i].isdigit():
                c += s[i]
                if i == len(s)-1 or (not s[i+1].isdigit()):
                    nums.append(int(c))
                    c = ""
            elif s[i] in ["+", "-"]:
                while ops and ops[-1] in list("+-*/"):
                    merge_nums()
                ops.append(s[i])
            elif s[i] in ["*", "/"]:
                while ops and ops[-1] in list("*/"):
                    merge_nums()
                ops.append(s[i])
            elif s[i] == ")":
                while ops[-1] != "(":
                    merge_nums()
                ops.pop()
            elif s[i] == "(":
                ops.append("(")
            i += 1

        while ops:
            merge_nums()

        return nums[0]
```

921. Minimum Add to Make Parentheses Valid

```
class Solution:
    def minAddToMakeValid(self, s: str) -> int:
        # 思路：最后看需要左边和右边分别需要补齐多少括号
        # 什么情况下需要补左括号？当前是右括号而且右括号需
        # 要补充的数量为0（也就是之前没有左括号和它匹配）
        # left和right分别是左边和右边需要补充的括号数量
        left, right = 0, 0
        for c in s:
            if c == ")" and right == 0:
                left += 1
            else:
                if c == "(":
                    right += 1
                else:
                    right -= 1
        return left + right
```

1047. Remove All Adjacent Duplicates In String

```
class Solution:
    def removeDuplicates(self, s: str) -> str:
        stack = []
        for c in s:
            # 遇到和栈顶的就弹出
            if stack and stack[-1] == c:
                stack.pop()
            else:
                stack.append(c)
        return "".join(stack)
```

1249. Minimum Remove to Make Valid Parentheses

```python
class Solution:
    def minRemoveToMakeValid(self, s: str) -> str:
        # 用一个集合把肯定要去掉的记下来
        remove = set()
        stack = []
        for i, c in enumerate(s):
            if c == "(":
                stack.append(i)
            elif c == ")":
                if len(stack) == 0:
                    remove.add(i)
                else:
                    stack.pop(-1)
        # stack留下来的肯定是没有匹配的左括号
        remove = remove.union(set(stack))
        res = ""
        for i, c in enumerate(s):
            if i not in remove:
                res += c
        return res
```

1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

```python
class Solution:
    def longestSubarray(self, nums: List[int], limit: int) -> int:
        l, r = 0, 0
        max_q, min_q = [], [] # 前者非增，后者非减
        res = 0
        # 队列维护了l到r之间可能的最大值和最小值
        while r < len(nums):

            # 不能取等于号，因为希望序列长
            while max_q and nums[max_q[-1]] < nums[r]:
                max_q.pop()
            while min_q and nums[min_q[-1]] > nums[r]:
                min_q.pop()
            min_q.append(r)
            max_q.append(r)

            # 这里是不会空数组的
            while nums[max_q[0]] - nums[min_q[0]] > limit:
                # 当前不符合要求，至少从下一个开始
                l = min(max_q[0], min_q[0]) + 1
                while l > max_q[0]:
                    max_q.pop(0)
                while l > min_q[0]:
                    min_q.pop(0)

            res = max(res, r-l+1)
            r += 1
        return res
```

1597. Build Binary Expression Tree From Infix Expression

```
class Solution:
    def expTree(self, s: str) -> 'Node':
        nums, ops = [], []

        def merge_leaves():
            op = ops.pop()
            right = nums.pop()
            left = nums.pop()
            # 合并之后作为新的叶子节点
            nums.append(Node(op, left, right))

        for c in s:
            if c.isdigit():
                nums.append(Node(c))
            elif c in ["+", "-"]:
                # 这个while把 ( 给排除了
                while ops and ops[-1] in ["+", "-", "*", "/"]:
                    merge_leaves()
                ops.append(c)
            elif c in ["*", "/"]:
                # 乘除法优先级高，只能和乘除一起操作
                while ops and ops[-1] in ["*", "/"]:
                    merge_leaves()
                ops.append(c) # 等待下一个数过来乘
            elif c == "(":
                ops.append(c)
            elif c == ")":
                while ops[-1] != "(":
                    merge_leaves()
                ops.pop() # 把左括号拿掉

        # 剩下还有元素，比如1+2处理完了nums=[1,2] ops=[+]
        while ops:
            merge_leaves()

        return nums[0]
```

前缀树

208. Implement Trie (Prefix Tree)

```
class TrieNode:
    def __init__(self):
        self.children = dict()
        self.end = False

class Trie:

    def __init__(self):
        self.root = TrieNode()

    def insert(self, word: str) -> None:
        cur = self.root
        for c in word:
            if c not in cur.children:
                cur.children[c] = TrieNode()
            cur = cur.children[c]
        cur.end = True

    def search(self, word: str) -> bool:
        cur = self.root
        for c in word:
            if c not in cur.children:
                return False
            cur = cur.children[c]
        return cur.end

    def startsWith(self, prefix: str) -> bool:
        cur = self.root
        for c in prefix:
            if c not in cur.children:
                return False
            cur = cur.children[c]
        return True
```

211. Design Add and Search Words Data Structure

```
class TrieNode:
    def __init__(self):
        self.children = dict()
        self.end = False

class TrieTree:
    def __init__(self):
        self.root = TrieNode()
    def add(self, w):
        cur = self.root
        for c in w:
            if c not in cur.children:
                cur.children[c] = TrieNode()
            cur = cur.children[c]
        cur.end = True
    def search(self, w):
        return self.match(w, 0, self.root)
    def match(self, w, i, root):
        # 原来是for循环自动结束，现在必须使用索引判断结束
        if i == len(w):
            return root.end
        if w[i] != ".":
            if w[i] not in root.children:
                return False
            else:
                return self.match(w, i+1, root.children[w[i]])
        else:
            for child in root.children:
                if self.match(w, i+1, root.children[child]):
                    return True
            return False

class WordDictionary:

    def __init__(self):
        self.Tree = TrieTree()

    def addWord(self, word: str) -> None:
        self.Tree.add(word)

    def search(self, word: str) -> bool:
        return self.Tree.search(word)
```

212. Word Search II

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.end = False
class Trie:
    def __init__(self):
        self.root = TrieNode()

    def add(self, w):
        cur = self.root
        for c in w:
            if c not in cur.children:
                cur.children[c] = TrieNode()
            cur = cur.children[c]
        cur.end = True

    def delete(self, w):
        cur = self.root
        for c in w:
            cur = cur.children[c]
        cur.end = False

        def helper(node, depth): # 删掉自己路径，但是不干扰重合路径
            if depth == len(w):
                return len(node.children) == 0
            if helper(node.children[w[depth]], depth+1):
                node.children.pop(w[depth])
                return len(node.children) == 0 and node.end == False
        helper(self.root, 0)

class Solution:

    def findWords(self, board: List[List[str]], words: List[str]):

        self.tree = Trie()
        for w in words:
            self.tree.add(w)

        self.r, self.c = len(board), len(board[0])
        self.visited = [[False] * self.c for _ in range(self.r)]
        self.board = board
        self.words = words
        self.res = []
        for i in range(self.r):
            for j in range(self.c):
                self.dfs(i, j, self.tree.root, "")
        return self.res
```

```
    def dfs(self, i, j, node, w):

        if i < 0 or i >= self.r or j < 0 or j >= self.c:
            return
        if self.visited[i][j]:
            return
        if len(w) > 10:
            return
        c = self.board[i][j]
        if c not in node.children:
            return
        node = node.children[c]
        w += c
        if node.end:
            self.res.append(w)
            self.tree.delete(w)
        self.visited[i][j] = True
        self.dfs(i+1, j, node, w)
        self.dfs(i-1, j, node, w)
        self.dfs(i, j+1, node, w)
        self.dfs(i, j-1, node, w)
        self.visited[i][j] = False
```

472. Concatenated Words

```
class TrieNode:

    def __init__(self):
        self.children = {}
        self.end = False

class Trie:

    def __init__(self):
        self.root = TrieNode()

    def add(self, w):
        cur = self.root
        for c in w:
            if c not in cur.children:
                cur.children[c] = TrieNode()
            cur = cur.children[c]
        cur.end = True

    def query(self, w):
        if len(w) == 0:
            return True
        cur = self.root
        for i, c in enumerate(w):
            if c not in cur.children:
                return False
            cur = cur.children[c]
            # 找到了某个子单词，匹配一下后面的是不是可以
            if cur.end:
                if self.query(w[i+1:]):
                    return True
        return False # 上面的都无法匹配，表示已经没法匹配了

class Solution:

    def findAllConcatenatedWordsInADict(self, words: List[str]):
        res = []
        T = Trie()
        words.sort(key=lambda x: len(x))
        for word in words:
            if len(word) == 0:
                continue
            if T.query(word):
                res.append(word)
            T.add(word)
        return res
```

588. Design In-Memory File System

```
class TrieNode:

    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.isFile = False
        self.name = "" # 文件名/文件夹名
        self.content = ""

class FileSystem:

    def __init__(self):
        self.root = TrieNode()

    def ls(self, path: str) -> List[str]:
        node = self.root
        path = path.split("/")[1:]
        if path[0] != "": # 防止在根目录
            for p in path:
                node = node.children[p]
        if node.isFile:
            return [node.name]
        else:
            return sorted([i for i in node.children])

    def mkdir(self, path: str) -> None:
        node = self.root
        for p in path.split("/")[1:]:
            node = node.children[p]
            node.name = p

    def addContentToFile(self, filePath: str, content: str) -> None:
        node = self.root
        for p in filePath.split("/")[1:]:
            node = node.children[p]
            node.name = p
        node.isFile = True
        node.content += content

    def readContentFromFile(self, filePath: str) -> str:
        node = self.root
        for p in filePath.split("/")[1:]:
            node = node.children[p]
        return node.content
```

720. Longest Word in Dictionary

```
class TrieNode:
    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.isEnd = False
        self.w = ""

class Trie:
    def __init__(self):
        self.root = TrieNode()
    def insert(self, w):
        node = self.root
        for c in w:
            node = node.children[c]
        node.isEnd = True
        node.w = w
    def bfs(self):
        q = [self.root]
        res = ""
        while q:
            cur = q.pop(0)
            for node in cur.children.values():
                # 最巧妙的地方，这里可以保证每一步下去都是连续地下去
                if node.isEnd:
                    q.append(node)
                    if len(node.w) > len(res) or node.w < res:
                        # 这里由于可以保证队列后面的深度不低于前面
                        # 所以后面那个or无需check len(node.w) == len(res)
                        res = node.w
        return res

class Solution:
    def longestWord(self, words: List[str]) -> str:
        # Trie + BFS
        T = Trie()
        for w in words:
            T.insert(w)
        return T.bfs()
```

1268. Search Suggestions System

```
class TrieNode:

    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.words = []

class Solution:
    def suggestedProducts(self, products: List[str], searchWord: str):

        root = TrieNode()

        def add(w):
            node = root
            for c in w:
                node = node.children[c]
                node.words.append(w)
                node.words.sort()
                if len(node.words) > 3:
                    node.words.pop()

        for w in products:
            add(w)

        res = []
        giveup = False # 不用搜了
        node = root
        for c in searchWord:
            if giveup or c not in node.children:
                giveup = True
                res.append([])
            else:
                node = node.children[c]
                res.append(node.words)
        return res
```

堆

215. Kth Largest Element in an Array

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        heap = []
        for n in nums:
            if len(heap) == k:
                heapq.heappushpop(heap, n)
            else:
                heapq.heappush(heap, n)
        return heap[0]
```

218. The Skyline Problem

```python
class Solution:
    def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:

        import heapq

        line = []
        for item in buildings:
            line.append([item[0], -item[2]])
            line.append([item[1], item[2]])
        line.sort(key=lambda x: (x[0], x[1]))

        # 优先队列，左边界（负数表示）入队列，右边界出队列
        # 若当前位置对应最大高度更新了，此时说明是轮廓分割点
        res = []
        heap = []
        max_height = 0
        for item in line:
            pos, height = item[0], item[1]
            if height < 0:
                heapq.heappush(heap, height)
            else:
                heap.remove(-height)
                heapq.heapify(heap)
            cur_max_height = -heap[0] if len(heap) != 0 else 0
            if cur_max_height != max_height:
                res.append([pos, cur_max_height])
                max_height = cur_max_height

        return res
```

295. Find Median from Data Stream

```python
class MedianFinder:

    def __init__(self):
        self.h_min = [] # 存放不小于中位数的元素
        self.h_max = [] # 存放不大于中位数的元素

    def addNum(self, num: int) -> None:
        # Python的heapq只能操作min heap，因此处理h_max时需要存放相反数
        # 这里规定，h_max中元素最多比h_min多一个
        if len(self.h_max) == 0 or -self.h_max[0] >= num:
            heapq.heappush(self.h_max, -num)
            if len(self.h_max) - len(self.h_min) >= 2:
                temp = -heapq.heappop(self.h_max)
                heapq.heappush(self.h_min, temp)
        else:
            heapq.heappush(self.h_min, num)
            if len(self.h_max) < len(self.h_min):
                temp = heapq.heappop(self.h_min)
                heapq.heappush(self.h_max, -temp)

    def findMedian(self) -> float:
        if len(self.h_min) == len(self.h_max):
            return (self.h_min[0] - self.h_max[0]) / 2
        else:
            return -self.h_max[0]
```

347. Top K Frequent Elements

```
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        counter = defaultdict(int)
        for n in nums:
            counter[n] += 1
        # 注意是小顶堆，因为是当前元素大了，把最小的弹出来
        heap = []
        for n in counter:
            cnt = counter[n]
            if len(heap) == k:
                heapq.heappushpop(heap, (cnt, n))
            else:
                heapq.heappush(heap, (cnt, n))
        return [n for cnt, n in heap]
```

373. Find K Pairs with Smallest Sums

```python
class Solution:
    def kSmallestPairs(
        self,
        nums1: List[int],
        nums2: List[int],
        k: int,
    ) -> List[List[int]]:
        # 想象成一个矩形，从左下角开始
        # 每次pop出来之后，考虑pop的右边元素和上面元素
        # 元素A的右边可能和元素B的上面重复，因此需要去重
        res = []
        heap = [(nums1[0] + nums2[0], 0, 0)]
        visited = set((0, 0))

        while heap and len(res) < k:
            val, i, j = heapq.heappop(heap)
            res.append([nums1[i], nums2[j]])
            if i+1 < len(nums1) and (i+1, j) not in visited:
                visited.add((i+1, j))
                heapq.heappush(heap, (nums1[i+1] + nums2[j], i+1, j))
            if j+1 < len(nums2) and (i, j+1) not in visited:
                visited.add((i, j+1))
                heapq.heappush(heap, (nums1[i] + nums2[j+1], i, j+1))

        return res
```

973. K Closest Points to Origin

```python
class Solution:
    def kClosest(self, points: List[List[int]], k: int) -> List[List[int]]:
        # python是小顶堆，但是要取最小的k个元素
        # 因此每次弹出的要是最大元素，取负用大顶堆
        heap = []
        for x, y in points:
            d = x*x + y*y
            if len(heap) == k:
                heapq.heappushpop(heap, (-d, x, y))
            else:
                heapq.heappush(heap, (-d, x, y))
        return [[x, y] for d, x, y in heap]
```

1606. Find Servers That Handled Most Number of Requests

```
from sortedcontainers import SortedList

class Solution:
    def busiestServers(self, k: int, arrival: List[int], load: List[int]) ->
        available = list(range(k)) # already a min-heap
        busy = []
        res = [0] * k
        for i, a in enumerate(arrival):
            while busy and busy[0][0] <= a:
                # these are done, put them back as available
                _, x = heapq.heappop(busy) # x和时间没关系，是机器编号
                # while x < i:
                # x += k
                # heapq.heappush(available, x)
                # we want the x to be inside the range of [i, i+k)
                # when the i-th job comes in, min(available) is
                # at least i, at most i+k-1.
                heapq.heappush(available, i + (x-i)%k) # 等价于上面
            if available:
                x = heapq.heappop(available) % k
                heapq.heappush(busy, (a+load[i], x))
                res[x] += 1
        a = max(res)
        return [i for i in range(k) if res[i] == a]
```

2102. Sequentially Ordinal Rank Tracker

```
# add和get都是O(log(n))
class MyString:
    def __init__(self, word):
        self.word = word
    def __lt__(self, other):
        # 和正常字符串比较反过来
        return self.word > other.word
    def __eq__(self, other):
        return self.word == other.word
    def __repr__(self):
        return self.word

class SORTracker:
    def __init__(self):
        # 最小堆，把第i的放在顶上
        self.left = []
        # 最大堆，把第i+1的放在顶上
        self.right = []
        # get调用的计数器
        self.get_calls = 1
    def add(self, name: str, score: int) -> None:
        if not self.left:
            self.left.append((score, MyString(name)))
            return
        heappush(self.left, (score, MyString(name)))
        # 为了使最小堆弹出的恰好为第i个，因此超出时需要弹出
        if len(self.left) > self.get_calls:
            sc, w = heappop(self.left)
            heappush(self.right, (-sc, str(w)))
    def get(self) -> str:
        # 取出第i个
        _, location = self.left[0]
        # 把最大堆对顶的元素弹出来再还给最小堆
        if self.right:
            sc, w = heappop(self.right)
            heappush(self.left, (-sc, MyString(w)))
        self.get_calls += 1
        return str(location)

"""
class SORTracker:

    def __init__(self):
        self.data = []
        self.idx = 0
```

```
    def add(self, name: str, score: int) -> None:
        # O(n)复杂度
        insort(self.data,[-score,name])

    def get(self) -> str:
        name = self.data[self.idx][1]
        self.idx += 1
        return name
"""

"""
from sortedcontainers import SortedList

class SORTracker:
    def __init__(self):
        self.d = SortedList()
        self.i = 0

    def add(self, name: str, score: int) -> None:
        # SortedList为O(log(n))
        self.d.add((-score, name))

    def get(self) -> str:
        self.i += 1
        return self.d[self.i - 1][1]
"""
```

其他

题目

149. Max Points on a Line

```python
class Solution:
    def maxPoints(self, points: List[List[int]]) -> int:
        def get_k(x, y):
            if x[0] == y[0]:
                return float("inf")
            else:
                return (y[1] - x[1]) / (y[0] - x[0])
        res = 1
        n = len(points)
        for i in range(n-1):
            d = defaultdict(int)
            for j in range(i+1,n):
                k = get_k(points[i], points[j])
                d[k] += 1
            res = max(res, max(d.values())+1)
        return res
```

164. Maximum Gap

```
class Solution:
    def maximumGap(self, nums: List[int]) -> int:
        # 可以证明相邻数字的最大间距不会小于(max − min) / (N−1)，N是数组长度
        # 区间搞成左闭右开，最后一个区间单独放
        N = len(nums)
        max_x, min_x = max(nums), min(nums)
        if N == 1:
            return 0
        if max_x == min_x:
            return 0
        bucket = [[] for _ in range(N)]
        for x in nums:
            if x == max_x:
                # 单独放
                bucket[-1].append(x)
            else:
                # 最重要的是这个映射公式，向下取整放到左边
                bucket[floor(N * (x - min_x) / (max_x - min_x))].append(x)
        candidates = [(max(arr), min(arr)) for arr in bucket if arr]
        return max(j[1]-i[0] for i, j in zip(candidates[:-1], candidates[1:]))
```

223. Rectangle Area

```python
class Solution:
    def computeArea(
        self, ax1: int, ay1: int, ax2: int, ay2: int,
        bx1: int, by1: int, bx2: int, by2: int
    ) -> int:
        # 首先，我们调整两个矩形，让第一个矩形是靠最左边的
        # 其次，先考虑没有重叠的情况，有三种情况
        # 重叠部分
        # 上边界，取两个矩形的上边界的最小值
        # 下边界，取两个矩形的下边界的最大值
        # 左边界，取两个矩形的左边界的最大值
        # 右边界，取两个矩形的右边界的最小值
        if ax1 > bx1:
            ax1, bx1 = bx1, ax1
            ax2, bx2 = bx2, ax2
            ay1, by1 = by1, ay1
            ay2, by2 = by2, ay2
        area_both = (ax2 - ax1) * (ay2 - ay1) + (bx2 - bx1) * (by2 - by1)
        if by1 >= ay2 or ax2 <= bx1 or ay1 >= by2:
            return area_both
        up = min(ay2, by2)
        down = max(ay1, by1)
        right = min(ax2, bx2)
        left = max(ax1, bx1)
        return area_both - (up - down) * (right - left)
```

277. Find the Celebrity

```
# The knows API is already defined for you.
# return a bool, whether a knows b
# def knows(a: int, b: int) -> bool:

class Solution:
    def findCelebrity(self, n: int) -> int:
        # The first loop will stop to an candidate i who doesn't know anyone
        # from i+1 to n-1.
        # For any previous x<i either know sb else or is not known by sb else.
        # The second loop will check whether i knows anyone from 0 to i-1.
        # The third loop is gonna check whether all party participants know x or not.
        candidate = 0
        for i in range(1, n):
            if knows(candidate, i):
                candidate = i
        condition1 = all(knows(i, candidate) for i in range(n))
        condition2 = all(
            (i==candidate) or (not knows(candidate, i)) for i in range(n))
        return candidate if condition1 & condition2 else -1
```

330. Patching Array

```python
class Solution:
    def minPatches(self, nums: List[int], n: int) -> int:
        # 详解看视频，主要思路在于
        # 新来的数字和当前能表示（左闭右开）的是否有交集
        # 如果没有的话把当前的右区间元素(miss)拿到
        # 然后倍增可表示范围，然后再看新数字情况
        # 如果有可以拿的，直接拿了更新missing
        miss = 1
        ptr = 0
        count = 0
        while miss <= n:
            if ptr < len(nums):
                if miss < nums[ptr]:
                    count += 1
                    miss *= 2
                else:
                    miss += nums[ptr] # 可以表示的右边界增加
                    ptr += 1
            else:
                count += 1
                miss *= 2
        return count
```

365. Water and Jug Problem

```
class Solution:
    def canMeasureWater(self, x: int, y: int, z: int) -> bool:
        # 只需要求出x和y的最大公约数d，并判断z是否是d的整数倍即可
        # ax+by=z
        if x + y < z:
            return False
        if x == 0 or y == 0:
            return z == 0 or x + y == z
        return z % math.gcd(x, y) == 0
"""
# BFS方法
class Solution:
    def canMeasureWater(self, x: int, y: int, z: int) -> bool:
        stack = [(0, 0)]
        self.seen = set()
        while stack:
            remain_x, remain_y = stack.pop()
            if remain_x == z or remain_y == z or remain_x + remain_y == z:
                return True
            if (remain_x, remain_y) in self.seen:
                continue
            self.seen.add((remain_x, remain_y))
            # 把 X 壶灌满。
            stack.append((x, remain_y))
            # 把 Y 壶灌满。
            stack.append((remain_x, y))
            # 把 X 壶倒空。
            stack.append((0, remain_y))
            # 把 Y 壶倒空。
            stack.append((remain_x, 0))
            # 把 X 壶的水灌进 Y 壶，直至灌满或倒空。
            stack.append((remain_x - min(remain_x, y - remain_y),
                remain_y + min(remain_x, y - remain_y)))
            # 把 Y 壶的水灌进 X 壶，直至灌满或倒空。
            stack.append((remain_x + min(remain_y, x - remain_x),
                remain_y - min(remain_y, x - remain_x)))
        return False
"""
```

382. Linked List Random Node

```
class Solution:

    def __init__(self, head: Optional[ListNode]):
        self.head = head

    # 这个蓄水池抽样是在不知道序列长度时候非常有用
    def getRandom(self) -> int:
        node = self.head
        keep = None
        i = 1
        while node:
            random_n = random.randint(0, i-1)
            # 如果是0就换
            if random_n == 0:
                keep = node
            node = node.next
            i += 1
        return keep.val
```

384. Shuffle an Array

```python
class Solution:

    def __init__(self, nums: List[int]):
        # 循环n次，在第i次循环中
        # 1) 在[i,n)中随机抽取一个下标j
        # 2) 将第i个元素与第j个元素交换（理解为把第j个元素放过来）
        self.n = len(nums)
        self.nums = nums
        self._nums = nums[:]

    def reset(self) -> List[int]:
        self.nums = self._nums[:]
        return self.nums

    def shuffle(self) -> List[int]:
        for i in range(self.n):
            j = random.randint(i, self.n-1)
            self.nums[i], self.nums[j] = self.nums[j], self.nums[i]
        return self.nums
```

390. Elimination Game

```python
class Solution:
    def lastRemaining(self, n: int) -> int:
        a = 1 # 等差数列的第一项
        k = 0 # 反转次数
        cnt = n # 元素个数
        step = 1 # 公差
        while cnt > 1:
            if k % 2 == 0: # 正向
                a += step # 无论奇偶元素个数，第一个元素总是加一个公差
            else: # 反向
                if cnt % 2: # 如果是偶数元素个数，第一个元素反向删不掉，不变
                    a += step # 否则加一个公差
            k += 1 # 反转次数增加
            cnt //= 2 # 每次元素个数减半
            step *= 2 # 每次公差翻倍
        return a
```

391. Perfect Rectangle

```python
class Solution:
    def isRectangleCover(self, rectangles: List[List[int]]) -> bool:
        # 四周的顶点只出现1次，其余的都出现两次或者四次
        # 面积加起来和大矩型一致
        max_x, min_x, max_y, min_y = -1e9, 1e9, -1e9, 1e9
        count = defaultdict(int)
        area = 0
        for x1, y1, x2, y2 in rectangles:
            if x2 > max_x:
                max_x = x2
            if x1 < min_x:
                min_x = x1
            if y2 > max_y:
                max_y = y2
            if y1 < min_y:
                min_y = y1
            count[(x1, y1)] += 1
            count[(x2, y1)] += 1
            count[(x1, y2)] += 1
            count[(x2, y2)] += 1
            area += (x2 - x1) * (y2 - y1)
        if area != (max_x - min_x) * (max_y - min_y):
            return False
        for x in [max_x, min_x]:
            for y in [max_y, min_y]:
                if count[(x, y)] != 1:
                    return False
                count.pop((x, y))
        return all(count[i]==2 or count[i]==4 for i in count)
```

398. Random Pick Index

```
class Solution:

    def __init__(self, nums: List[int]):
        d = defaultdict(list)
        for idx, i in enumerate(nums):
            d[i].append(idx)
        self.d = d

    """
    # 蓄水池抽样，但是超时
    def pick(self, target: int) -> int:
        # 一共k个target在nums里
        # 第i次见到target，在[0,i)上返回整数，如果是0，那么当前ans为当前索引
        # 那么这个索引（即第i次出现target的位置）能够被最终保留到ans的概率为
        # 第i次抽0，后面都不抽0，(1/i)*(1-1/(i+1))*...*(1-1/(k))=1/k
        ans, count = 0, 0
        for i, n in enumerate(self.nums):
            if n == target:
                count += 1
                if random.randint(0, count-1) == 0:
                    ans = i
        return ans
    """
    def pick(self, target: int) -> int:
        n = len(self.d[target])
        return self.d[target][random.randint(0, n-1)]
```

497. Random Point in Non-overlapping Rectangles

```python
class Solution:

    def __init__(self, rects: List[List[int]]):
        self.rects = rects
        self.cum_val = [0]
        for rect in rects:
            width = rect[3] - rect[1] + 1
            height = rect[2] - rect[0] + 1
            self.cum_val.append(self.cum_val[-1] + width * height)
        self.cum_val.pop(0)

    def pick(self) -> List[int]:
        idx = self.get_random_idx()
        i = random.randint(self.rects[idx][0], self.rects[idx][2])
        j = random.randint(self.rects[idx][1], self.rects[idx][3])
        return [i, j]

    def get_random_idx(self):
        # [---|---------|--|-----]
        n = random.randint(1, self.cum_val[-1]) # 闭区间
        left, right = 0, len(self.cum_val)-1
        # 找的是比不小于自身且最小的，所以用模板1
        while left < right:
            mid = left + (right - left) // 2
            if self.cum_val[mid] >= n:
                right = mid
            else:
                left = mid + 1
        return left
```

528. Random Pick with Weight

```
class Solution:

    def __init__(self, w: List[int]):
        # 原来的数组是a1, a2, ..., an
        # cumsum之后变成s1, s2, ..., sn
        # 对应概率是(s_i-s_{i-1})/s_i=a_i/sum(a)
        self.cum = list(accumulate(w))

    def pickIndex(self) -> int:
        x = random.randint(1, self.cum[-1])
        return bisect_left(self.cum, x)
```

593. Valid Square

```
class Solution:
    def validSquare(self, p1, p2, p3, p4):
        # 菱形+对角线相等
        if p1==p2==p3==p4:return False
        def dist(x,y):
            return (x[0]-y[0])**2+(x[1]-y[1])**2
        ls=[
            dist(p1,p2),
            dist(p1,p3),
            dist(p1,p4),
            dist(p2,p3),
            dist(p2,p4),
            dist(p3,p4),
        ]
        ls.sort()
        if ls[0]==ls[1]==ls[2]==ls[3]:
            if ls[4]==ls[5]:
                return True
        return False
```

1344. Angle Between Hands of a Clock

```
class Solution:
    def angleClock(self, hour: int, minutes: int) -> float:
        h = 30*hour + minutes/60 * 30
        m = 360*(minutes/60)
        diff = abs(h - m)
        return diff if diff < 180 else 360 - diff
```

算法

基本算法

bucket sort

```
def bucketsort(arr, n_bins):
    max_val, min_val = max(arr), min(arr)
    bins = [[] for _ in range(n_bins + 1)]
    for i in arr:
        bins[int((i - min_val) / (max_val - min_val) * n_bins)].append(i)
    for i in range(len(bins)):
        bins[i].sort()
    res = []
    for bin in bins:
        res += bin
    return res
```

count sort

```
def countsort(arr):
    # 值需要是整数
    max_val, min_val = max(arr), min(arr)
    count = [0] * (max_val - min_val + 1)
    for i in arr:
        count[i - min_val] += 1
    res = []
    for i, count in enumerate(count):
        if count:
            while count > 0:
                res.append(i + min_val)
                count -= 1
    return res
```

quick sort

```
def partion(arr, l, r):
    pivot = arr[r]
    small_ptr = l
    for i in range(l, r):
        if arr[i] < pivot:
            arr[i], arr[small_ptr] = arr[small_ptr], arr[i]
            small_ptr += 1
    arr[r], arr[small_ptr] = arr[small_ptr], arr[r]
    return small_ptr

def quicksort(arr, l_ptr, r_ptr):
    if l_ptr < r_ptr:
        pivot_idx = partion(arr, l_ptr, r_ptr)
        quicksort(arr, l_ptr, pivot_idx-1)
        quicksort(arr, pivot_idx+1, r_ptr)
```

radix sort

```
def radixsort(arr):
    # 假设都是正整数
    def _inner_sort(d):
        bins = [[] for i in range(10)] # 0-9一共10位数字
        for val in arr:
            # 放进去的是值
            bins[(val // d) % 10].append(val)
        idx = 0
        for bin in bins:
            for val in bin:
                arr[idx] = val
                idx += 1

    max_val = max(arr)
    cur_digit = 1
    while max_val // cur_digit > 0:
        _inner_sort(cur_digit)
        cur_digit *= 10
```

quick select

```
def quickselect(arr, k):
    n = len(arr)
    if n == 1:
        return arr[0]

    pivot = arr[-1]
    small, equal, big = [], [], []
    for i in arr:
        if i < pivot:
            small.append(i)
        elif i > pivot:
            big.append(i)
        else:
            equal.append(i)

    if k <= len(small):
        return quickselect(small, k)
    elif k <= len(small) + len(equal):
        return equal[0]
    else:
        return quickselect(big, k - len(small) - len(equal))
```

merge select

```
def merge(arr):
    n = len(arr)
    if n == 1:
        return arr
    l = merge(arr[:n//2])
    r = merge(arr[n//2:])
    res = []
    l_ptr, r_ptr = 0, 0
    while l_ptr < len(l) and r_ptr < len(r):
        num_l, num_r = l[l_ptr], r[r_ptr]
        if num_l < num_r:
            res.append(num_l)
            l_ptr += 1
        else:
            res.append(num_r)
            r_ptr += 1
    if l_ptr < len(l):
        res += l[l_ptr:]
    else:
        res += r[r_ptr:]
    return res
```

heap sort

```
def heapify(arr, n, i):
    largest = i # Initialize largest as root
    l = 2 * i + 1 # left = 2*i + 1
    r = 2 * i + 2 # right = 2*i + 2
    # See if left child of root exists and is
    # greater than root
    if l < n and arr[i] < arr[l]:
        largest = l
    # See if right child of root exists and is
    # greater than root
    if r < n and arr[largest] < arr[r]:
        largest = r
    # Change root, if needed
    if largest != i:
        (arr[i], arr[largest]) = (arr[largest], arr[i]) # swap
    # Heapify the root.
        heapify(arr, n, largest)

def heapSort(arr):
    n = len(arr)

    # Build a maxheap.
    # Since last parent will be at ((n//2)-1)
    # we can start at that location.
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    # One by one extract elements
    for i in range(n - 1, 0, -1):
        (arr[i], arr[0]) = (arr[0], arr[i]) # swap
        heapify(arr, i, 0)
```

dijkstra

```
def dijkstra_heap(G, src):

    num_v = len(G)
    dist = [float("inf")] * num_v # src到各节点距离
    fixed = [False] * num_v # 该节点是否已经确定最短距离
    parent = [None for i in range(num_v)] # 记录前一个节点
    dist[src] = 0 # 源点最短距离设为0
    parent[src] = src # 源点无前一个节点

    q = [(0, src)] # (权重, 父节点)
    while q:
        # 把当前最小最短路径的边弹出
        w_src, src = heappop(q)
        # 如果出发点是还未得到最终最短距离的
        # 那么选择它进行固定，并考察邻居情况
        if not fixed[src]:
            fixed[src] = True
            for w_dest, dest in G[src]:
                # 邻居不能是已经固定的
                if not fixed[dest]:
                    new_dist = w_src + w_dest
                    if new_dist < dist[dest]:
                        dist[dest] = new_dist
                        parent[dest] = src
                        # 把新的这条边加入最小堆中
                        heappush(q, (new_dist, dest))
    return dist, parent
```

bellman

```python
def BellmanFord(G, src):
    num_v = len(G)
    dist = [float("inf")] * num_v # src到各节点距离
    parent = [None for i in range(num_v)] # 记录前一个节点
    dist[src] = 0 # 源点最短距离设为0
    parent[src] = src # 源点无前一个节点

    edges = []
    for i in G:
        for w, j in G[i]:
            edges.append((i, j, w))

    for _ in range(num_v-1):
        # 这个算法的重点在于第k次一定能找到最短路径长度为k的
        # 所以循环次数为顶点数减一，n个顶点最短路最多为V−1
        for i, j, w in edges:
            new_dist = dist[i] + w
            if new_dist < dist[j]:
                dist[j] = new_dist
                parent[j] = i

    # 如果发现这个时候还有不符合约束的，则有负环
    for i, j, w in edges:
        if dist[j] > dist[i] + w:
            print("Circle!")
            break

    return dist, parent
```

floyd

```
def floydWarshall(graph):
    """ dist[][] will be the output
       matrix that will finally
        have the shortest distances
        between every pair of vertices """
    """ initializing the solution matrix
    same as input graph matrix
    OR we can say that the initial
    values of shortest distances
    are based on shortest paths considering no
    intermediate vertices """

    dist = list(map(lambda i: list(map(lambda j: j, i)), graph))

    """ Add all vertices one by one
    to the set of intermediate
     vertices.
     ---> Before start of an iteration,
     we have shortest distances
     between all pairs of vertices
     such that the shortest
     distances consider only the
     vertices in the set
    {0, 1, 2, .. k-1} as intermediate vertices.
      ----> After the end of a
      iteration, vertex no. k is
     added to the set of intermediate
     vertices and the
    set becomes {0, 1, 2, .. k}
    """
    for k in range(V):

        # pick all vertices as source one by one
        for i in range(V):

            # Pick all vertices as destination for the
            # above picked source
            for j in range(V):

                # If vertex k is on the shortest path from
                # i to j, then update the value of dist[i][j]
                dist[i][j] = min(dist[i][j],
                                 dist[i][k] + dist[k][j])
```

di-connected

```
def directed_connected(G, G_reverse, strong=True):
    # weekly connected意思是只要连起来就ok，不要求方向
    # 所有正过来一遍，然后反过来一遍就行了
    visited_forward = [False] * len(G)
    visited_backward = [False] * len(G)
    def dfs_forward(u):
        if visited_forward[u]:
            return
        visited_forward[u] = True
        for v in G[u]:
            dfs_forward(v)
    def dfs_backward(u):
        if visited_backward[u]:
            return
        visited_backward[u] = True
        for v in G_reverse[u]:
            dfs_backward(v)
    dfs_forward(0)
    dfs_backward(0)
    if strong:
        # 两个方向都要能够走到
        return all(visited_forward) and all(visited_backward)
    for i in range(len(G)):
        # week的话，两边有一个走得到就行
        if (not visited_forward[i]) and (not visited_backward[i]):
            return False
    return True
```

undi-connected

```
def undirected_connected(G):
    # 对于无向图而言，只需要从节点u出发，然后dfs看是不是都能到
    visited = [False] * len(G)
    def dfs(u):
        if visited[u]:
            return
        visited[u] = True
        for v in G[u]:
            dfs(v)
    dfs(0)
    return all(visited)
```