OJ的pylint是静态检查，有时候报的不对。解决方法有两种，如下：
1）第一行加# pylint: skip-file （跳过检查）
2）方法二：如果函数内使用全局变量（变量类型是immutable，如int），则需要在程序最开始声明一下。
如果是全局变量是list类型，则不受影响。 （cnt=0）

```
from functools import lru_cache
@lru_cache(maxsize=128)#tle了加大，mle了减小
def func():
import sys
sys.setrecursionlimit(1<<30)(莫名其妙都可以用，不局限于RE)
```

进制：2进制`bin()`输出0b...，八进制`oct()`输出0o...，十六进制`hex()`输出0x...，`int(str,base)`可以把字符串按照指定进制转为十进制默认base=10

字符串：

1. str.find()查找指定字符，注意如果有的话会返回第一个找到的，如果没有会返回-1
2. str.title()首字母大写（每个单词） str.lower()/upper()每个字母小/大写 str.strip()去除空格，有rstrip/lstrip去掉尾部/头部的空格
3. 赋值 if (a := reversed_cube[b]+reversed_cube[c]+reversed_cube[d]) in cube:(赋值操作)

```
print("Hello", end=" ")
print("World!")#Hello World!
print("Python", "is", "fun", sep="-")#Python-is-fun
```

```
lst=[1,1,2,3]                   #count
lst_a=[1,1,3,4]
from collections import Counter
counter = Counter(lst)
for k in lst_a:
    c = counter[k]#快
    d=lst.count(k)
lst_a=[1,1,3,4]
from collections import Counter
counter = Counter(lst)
for k in lst_a:
    c = counter[k]#快
    d=lst.count(k)
```

怎么把一串字母分成单个的字母添加到集合里：

```
    lst=['zbcd','efg']
    good.update(lst[0])
```

```
from collections import defaultdict
a=defaultdict(list)#以空列表为默认值的defaultdict,类似地,括号里是int,set也可以
```

```
a = 5,b = 3
print(f"The sum of {a} and {b} is {a + b}.")
value = 12345.6789
print(f"科学计数法（小写）：{value:.2e}")  # 小写形式1.23e+04
print(f"百分比格式化：{value:.2%}")  # 保留 2 位小数的百分比
pi = 3.14159
print(f"Pi is approximately {pi:.2f}")
print(f"Adult status: {'Yes' if age >= 18 else 'No'}")
```

字典 注意字典无序！

```
print(my_dict.get('address', 'Not Found')) # 输出：Not Found(如果没有就输出这个)
#通过值来搜索键
找到所有的键：
法一：keys = [key for key, value in my_dict.items() if value == search_value]
法二：keys = list(filter(lambda key: my_dict[key] == search_value, my_dict))
找到第一个符合条件的键：
key = next((key for key, value in my_dict.items() if value == search_value), None)
#删除键值对
法一：del my_dict['city'] # 删除 'city' 键值对
法二：age = my_dict.pop('age')
print(age) # 输出：26
#排序
sorted_dict = dict(sorted(my_dict.items(), key=lambda x: x[1], reverse=True))
```

集合

```
my_set = {1, 2, 3, 4, 5}
another_set = set([3, 4, 5, 6, 7])
# 删除元素（不存在元素可抛出错误）
my_set.remove(2)
# 删除不存在的元素，不会抛出错误
my_set.discard(10)
```

列表

```
1. join对象是str
2. print(*res)（把列表解包为空格分隔）
words = ["banana", "apple", "pear", "cherry"]
sorted_words = sorted(words, key=len)#.sort只能对列表
res.sort(key=lambda x:(x[0],x[1])) #首先根据元组的第一个元素进行排序，如果第一个元素相等，则会根据第二个元素进行排序
enumerate(iterable, start=0)，iterable：这是想要遍历的可迭代对象（如列表、元组、字符串等）。start：可选，指定索引从哪里开始，默认从 0 开始。返回元组，第一个是索引，第二个是对应的值。
```

二分

```
import bisect
a = [1, 2, 4, 4, 8]
print(bisect.bisect_left(a, 4)) # 输出: 2
print(bisect.bisect_right(a, 4)) # 输出: 4
print(bisect.bisect(a, 4)) # 输出: 4
bisect.insort(arr, 5) # 将 5 插入到正确的位置
print(arr) # 输出 [1, 3, 4, 5, 10, 12]
```

优先队列

```
import heapq
data = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]
heapq.heapify(data)
print(data) # 输出: [0, 1, 2, 3, 9, 5, 4, 6, 8, 7]
heapq.heappush(data, -5)
print(data) # 输出: [-5, 0, 2, 3, 1, 5, 4, 6, 8, 7, 9]
print(heapq.heappop(data)) # 输出: -5
```

日期与时间

```
import calendar
calendar.setfirstweekday(calendar.SUNDAY)  # 更改周初为星期日(默认为星期一)
calendar.isleap(2024)  # 返回True
calendar.leapdays(2000, 2025)  # 返回6(含首不含尾)
calendar.month(2024, 1)  # 返回str类型2024-1日历
calendar.monthrange(2024, 1)  # 返回(0, 31)
calendar.monthcalendar(2024, 1)  # 返回二维数组weeks,缺日(不在此月)为0
```

```
from datetime import datetime
datetime.now()  # 返回"2024-01-01 12:34:56.789123";.today()/.time()同
dt = datetime(2024, 1, 1, 12, 30, 0)  # 2024-01-01 12:30:00
dt.strftime('%Y-%m-%d %H:%M:%S')  # 2024-01-01 12:30:00
dt.strftime('%A, %B %d, %Y')  # Monday, January 01, 2024
date_str = "2024-01-01 12:30:00"
parsed_date = datetime.strptime(date_str, '%Y-%m-%d %H:%M:%S')
print(parsed_date)  # 2024-01-01 12:30:00
now = datetime.now()
future_date = now + timedelta(days=7)
past_date = now - timedelta(hours=1)
d1 = datetime(2024, 1, 1)
d2 = datetime(2023, 12, 31)
d1 > d2  # True
d1 - d2  # 1 day, 0:00:00
print(datetime(2023, 10, 5).weekday()) # 输出: 3 (星期四)
```

数学

print(math.isclose(a, b)) # True（浮点数比较

- `math.pi`：圆周率 (π)，约为 3.14159。
- `math.e`：自然对数的底 (e)，约为 2.71828。
- `math.tau`：2π，约为 6.28318。
- `math.inf`：正无穷大。
- `math.nan`：浮点型的 "非数字"（Not a Number）。

---

2. 基本函数

- **幂和对数**
  - `math.sqrt(x)`：返回 $\sqrt{x}$。
  - `math.pow(x, y)`：返回 x^y。
  - `math.exp(x)`：返回 e^x。
  - `math.log(x[, base])`：返回 $log_{base}(x)$，默认是自然对数 ($\ln x$)。
  - `math.log2(x)`：返回以 2 为底的对数 ($log_2(x)$)。
  - `math.log10(x)`：返回以 10 为底的对数 ($log_{10}(x)$)。
- `math.sin(x)`、`math.cos(x)`、`math.tan(x)`：正弦、余弦、正切。
- `math.asin(x)`、`math.acos(x)`、`math.atan(x)`：反三角函数。
- `math.atan2(y, x)`：返回点 (x, y) 的极角。
- `math.hypot(x, y)`：计算 $\sqrt{x^2 + y^2}$。

双曲函数

- `math.sinh(x)`、`math.cosh(x)`、`math.tanh(x)`：双曲正弦、余弦、正切。
- `math.asinh(x)`、`math.acosh(x)`、`math.atanh(x)`：反双曲函数。

特殊函数

- `math.factorial(x)`：计算 $x!$（x 的阶乘）。
- `math.gamma(x)`：伽玛函数。
- `math.lgamma(x)`：伽玛函数的自然对数。

- `math.ceil(x)`：返回大于等于 x 的最小整数。
- `math.floor(x)`：返回小于等于 x 的最大整数。
- `math.trunc(x)`：返回 x 的整数部分，向零方向截断。
- `math.fabs(x)`：返回 x 的绝对值。
- `math.fmod(x, y)`：返回 $x \mod y$。
- `math.copysign(x, y)`：返回具有 y 符号的 x。
- `math.isfinite(x)`：判断 x 是否为有限数。
- `math.isinf(x)`：判断 x 是否为无穷大。
- `math.isnan(x)`：判断 x 是否为 NaN。
- `math.isclose(a, b, *, rel_tol=1e-9, abs_tol=0.0)`：判断 a 和 b 是否接近。

`math.degrees(x)`：将弧度转为角度。

`math.radians(x)`：将角度转为弧度。

新增函数（Python 3.8 引入）

`math.prod(iterable[, start])`：返回可迭代对象中所有元素的乘积。

`math.isqrt(n)`：计算整数平方根（返回整数）。

`math.dist(p, q)`：计算两点 p 和 q 间的欧几里得距离。

`math.hypot(*coordinates)`：扩展支持 n 维欧几里得距离。

```
import math
math.pi      math.e
math.sqrt(isqrt带取整)
                              math.pow(x,y)=x**y       math.log(x,base)
math.floor()          math.ceil()      math.factorial()#阶乘
math.gcd(*integers)       math.lcm(*integers)       math.comb(n,k)#Cnk
math.dist(p,q)#p,q为两个大小相同数组，计算欧氏距离
math.prod(iterable,start)#计算iterable所有数乘积在start上
math.isclose(a,b)#浮点数避免a==b可以a-b==0或者用这个判断
```

- `math.dist(p, q)`：
  计算两个点 p 和 q 的欧几里得距离，点的表示是大小相同的序列（如元组或列表）。
  示例：`math.dist([0, 0], [3, 4])` 返回 `5.0`。
- `math.prod(iterable, start=1)`：
  计算 `iterable` 中所有数字的乘积，并乘以可选的初始值 `start`。
  示例：`math.prod([2, 3, 4])` 返回 `24`。

拷贝

import copy

original = [1, 2, [3, 4]]

copied = copy.deepcopy(original)

print(copied) # 输出: [1, 2, [3, 4]]

数组操作

```
squared = list(map(lambda x: x**2, [1, 2, 3, 4]))
print(squared) # 输出: [1, 4, 9, 16]

a = [1, 2, 3]
b = ['a', 'b', 'c']
zipped = list(zip(a, b))
print(zipped) # 输出: [(1, 'a'), (2, 'b'), (3, 'c')]

filtered = list(filter(lambda x: x > 2, [1, 2, 3, 4]))
print(filtered) # 输出: [3, 4]
```

```
enumerated = list(enumerate(['a', 'b', 'c']))
print(enumerated) # 输出: [(0, 'a'), (1, 'b'), (2, 'c')]

lt = sorted(enumerate(lst,1),key=lambda x:x[1])
```

```
import itertools
for item in itertools.product('AB', repeat=2):  # 生成笛卡尔积
    print(item)  # ('A', 'A')\n('A', 'B')\n('B', 'A')\n('B', 'B')
for item in itertools.product('AB', '12'):
    print(item)  # ('A', '1')\n('A', '2')\n('B', '1')\n('B', '2')
result = list(permutations([1,2,3],3))  # [(1, 2, 3), (1, 3, 2), ……];生成全排列
b=combinations(list,k)#生成list的k元组合（无序）（每个以元组形式存在）
```

```
**按照右端点排序**：
（1）不相交区间数目最大
（2）区间选点问题（给出一堆区间，取**尽量少**的点，使得每个区间内**至少有一个点**）——尽量选择当前区间最右
边的点，同不相交区间数目最大
(3)区间覆盖问题：给出一堆区间和一个目标区间，问最少选择多少区间可以**覆盖**掉题中给出的这段目标区间。
**按照左端点排序**：
（1）区间合并（左端点由小到大排序，维护前面区间右端点ed）
（2）区间分组问题：从**前往后**依次枚举每个区间，判断当前区间能否被放到某个现有组里面。
```

ASCII表

```
n = int(input())%26
s = input()
ans = ''
for _ in s:
    if 'a'<=_<='z':
        if 'a'<=chr(ord(_)-n)<='z':
            _ = chr(ord(_)-n)
        else:
            _= chr(ord(_)-n+26)
    else:
        if 'A'<=chr(ord(_)-n)<="Z":
            _ = chr(ord(_) - n)
        else:
            _= chr(ord(_)-n+26)
    ans += _
print(ans)
```

欧拉筛

```
    is_prime = [0 for i in range(l+1)]
    primes = []
    for i in range(2, l+1):
        if is_prime[i] == 0:
            primes.append(i)
        for p in primes:
            if i * p > l:
                break
            is_prime[i * p] = 1
            if i % p == 0:
                break
    return is_prime
```

埃氏筛（比欧拉慢

```
    for i in range(2, int(n**0.5) + 1):
        if primes[i]:  # 如果 i 是素数，将 i 的所有倍数标记为非素数
            for j in range(i * i, n + 1, i):
                primes[j] = False
```

辅助栈，堆猪，一个栈单独存储min

字典标记 懒删除

```
import heapq
from collections import defaultdict
out = defaultdict(int)
pigs_heap = []
pigs_stack = []
while True:
    try:
        s = input()
    except EOFError:
        break
    if s == "pop":
        if pigs_stack:
            out[pigs_stack.pop()] += 1
    elif s == "min":
        if pigs_stack:
            while True:
                x = heapq.heappop(pigs_heap)
                if not out[x]:
                    heapq.heappush(pigs_heap, x)
                    print(x)
                    break
                out[x] -= 1
    else:
        y = int(s.split()[1])
        pigs_stack.append(y)
        heapq.heappush(pigs_heap, y)
```

dp 01背包

```
def knapsack_1d(weights, values, W):  # 一维视为二维的滚动数组实现
    n = len(weights)
    dp = [0] * (W + 1)  # 初始化 dp 数组，容量从 0 到 W
    for i in range(n):  # 遍历每件物品
        for j in range(W, weights[i] - 1, -1):  # 倒序遍历背包容量(保证每件物品只能选一次)
            dp[j] = max(dp[j], dp[j - weights[i]] + values[i])
    return dp[W]
```

完全背包

```
def knapsack_complete(weights, values, capacity):
    dp = [0] * (capacity + 1)  # dp[j]为当背包容量为j时,背包所能容纳的最大价值
    dp[0] = 0
    for i in range(len(weights)):  # 遍历所有物品
        for j in range(weights[i], capacity + 1):  # 从当前物品的重量开始，计算每个容量的最大价值
            dp[j] = max(dp[j], dp[j - weights[i]] + values[i])
    return dp[capacity]
```

装满

```
def knapsack_complete_fill(weights, values, capacity):
    dp = [-float('inf')] * (capacity + 1)  # 初始值为负无穷，表示不能达到该容量
    dp[0] = 0  # 容量为 0 时，价值为 0
    for i in range(len(weights)):  # 遍历所有物品
        for w in range(weights[i], capacity + 1):  # 遍历所有容量，从 weights[i] 开始
            dp[w] = max(dp[w], dp[w - weights[i]] + values[i])
    # 如果 dp[capacity] 仍为 -inf，说明无法填满背包
    return dp[capacity] if dp[capacity] != -float('inf') else 0
```

有上限

```
def binary_optimized_multi_knapsack(weights, values, quantities, capacity):
```

```
    # 使用二进制优化解决多重背包问题
    n = len(weights)
    items = []
    # 将每种物品根据数量拆分成若干子物品（使用二进制优化）
    for i in range(n):
        w, v, q = weights[i], values[i], quantities[i]
        k = 1
        while k < q:
            items.append((k * w, k * v))  # 添加子物品(weight, value)
            q -= k
            k << 1  # 位运算,相当于k *= 2,按二进制拆分,物品时间复杂度由q变为log(q)
        if q > 0:
            items.append((q * w, q * v))  # 添加剩余部分,如果有的话
    # 动态规划求解0-1背包问题
    dp = [0] * (capacity + 1)
    for w, v in items:  # 遍历所有子物品
        for j in range(capacity, w - 1, -1):  # 01背包的倒序遍历
            dp[j] = max(dp[j], dp[j - w] + v)
    return dp[capacity]
```

最长公共子序列

```
for i in range(len(A)):
    for j in range(len(B)):
        if A[i] == B[j]:
            dp[i][j] = dp[i-1][j-1]+1
        else:
            dp[i][j] = max(dp[i-1][j],dp[i][j-1])
```

最长单调子序列

```
dp = [1]*n
for i in range(1,n):
    for j in range(i):
        if A[j]<A[i]:
            dp[i] = max(dp[i],dp[j]+1)
ans = sum(dp)
```

多dp（土豪）

```
value = list(map(int, input().split(",")))
dp_keep = value[0]      # 不放回
dp_remove = value[0]    # 放回一件商品
ans = value[0]  # 答案记录
for i in range(1, len(value)):  # 对结尾为第i件商品的选法
    previous_dp_keep = dp_keep  # 保存结尾尾i-1时的选法
    dp_keep = max(dp_keep + value[i], value[i])  # 维护dp_keep
    dp_remove = max(previous_dp_keep, dp_remove + value[i])  # 判断是否放回第i个商品更划算
    ans = max(ans, dp_keep, dp_remove)
print(ans)
```

coins

```
while True:
    n, m = map(int, input().split())
    if n == 0 and m == 0:
        break
    ls = list(map(int, input().split()))
    w = (1 << (m + 1)) - 1                  #e.g., m=10, w=2047
    result = 1
    for i in range(n):
        number = ls[i+n] + 1                # e.g., number = 10
        limit = int(math.log(number, 2))    # limit = 3
        rest = number - (1 << limit)        # rest = 3
        for j in range(limit):
```

```
            result = (result | (result << (ls[i] * (1 << j)))) & w
        if rest > 0:
            result = (result | (result << (ls[i] * rest))) & w
    print(bin(result).count('1') - 1)
```

最大子矩阵Kadane,extend

```
def max_submatrix(matrix):
    def kadane(arr):
        max_end_here = max_so_far = arr[0]
        for x in arr[1:]:
            max_end_here = max(x, max_end_here + x)
            max_so_far = max(max_so_far, max_end_here)
        return max_so_far
    rows = len(matrix)
    cols = len(matrix[0])
    max_sum = float('-inf')
    for left in range(cols):
        temp = [0] * rows
        for right in range(left, cols):
            for row in range(rows):
                temp[row] += matrix[row][right]
            max_sum = max(max_sum, kadane(temp))
    return max_sum
n = int(input())
nums = []
while len(nums) < n * n:
    nums.extend(input().split())
matrix = [list(map(int, nums[i * n:(i + 1) * n])) for i in range(n)]
max_sum = max_submatrix(matrix)
print(max_sum)
```

单调栈 接雨水

```
    def trap(self, height: List[int]) -> int:
        stack = []
        water = 0
        for i in range(len(height)):
            while stack and height[i] > height[stack[-1]]:
                top = stack.pop()
                if not stack:
                    break
                distance = i - stack[-1] - 1
                bounded_height = min(height[i], height[stack[-1]]) - height[top]
                water += distance * bounded_height
            stack.append(i)
        return water
```

dfs

```
def dfs(x, y):
    stack = [(x, y)]
    while stack:
        x, y = stack.pop()
        if field[x][y] != 'W':
            continue
        field[x][y] = '.'  # 标记当前位置为已访问
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < n and 0 <= ny < m and field[nx][ny] == 'W':
                stack.append((nx, ny))
n, m = map(int, input().split())
field = [list(input()) for _ in range(n)]
directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
cnt = 0
for i in range(n):
```

```
    for j in range(m):
        if field[i][j] == 'W':
            dfs(i, j)
            cnt += 1
print(cnt)
```

bfs

```
from collections import deque#矩阵中的块
def bfs(x, y):
    q = deque([(x, y)])
    inq_set.add((x,y))
    while q:
        front = q.popleft()
        for i in range(MAXD):
            next_x = front[0] + dx[i]
            next_y = front[1] + dy[i]
            if matrix[next_x][next_y] == 1 and (next_x,next_y) not in inq_set:
                inq_set.add((next_x, next_y))
                q.append((next_x, next_y))
matrix=[[-1]*(m+2)]+[[-1]+list(map(int,input().split()))+[-1] for i in range(n)]+[[-1]*(m+2)]
inq_set = set()
counter = 0
for i in range(1,n+1):
    for j in range(1,m+1):
        if matrix[i][j] == 1 and (i,j) not in inq_set:
            bfs(i, j)
            counter += 1
```

dijkstra

走山路，heappop的时候，才能入inq，这样才能保证最短的优先访问，否则最短路径可能加不进inq

```
import heapq
m,n,p = map(int,input().split())
ma = []
dir = [(0,1),(0,-1),(1,0),(-1,0)]
for _ in range(m):
    ma.append(input().split())
for i in range(p):
    try:
        a,b,c,d = map(int,input().split())
        queue = []
        flag = False
        inq = set()
        heapq.heappush(queue,(0,a,b))
        inq.add((a,b))
        while queue:
            step,x,y=heapq.heappop(queue)
            inq.add((x,y))
            if x==c and y==d:
                flag=True
                break
            for (dx,dy) in dir:
                nx,ny = x+dx,y+dy
                if 0<=nx<m and 0<=ny<n and ma[nx][ny]!='#' and (nx,ny) not in inq:
                    heapq.heappush(queue,(step+abs(int(ma[nx][ny])-int(ma[x][y])),nx,ny))
        if flag:
            print(step)
        else:
            print('NO')
    except:
        print('NO')
```

滑雪升级

```
import heapq
from typing import List
class Solution:
    def updateMatrix(self, mat: List[List[int]]) -> List[List[int]]:
        m, n = len(mat), len(mat[0])
        dp = [[float('inf')] * n for _ in range(m)]
        heap = []
        # 初始化，所有的0加入到堆中
        for i in range(m):
            for j in range(n):
                if mat[i][j] == 0:
                    dp[i][j] = 0
                    heapq.heappush(heap, (0, i, j))  # (distance, x, y)
        # 定义四个方向的移动
        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        # 使用堆进行更新
        while heap:
            dist, x, y = heapq.heappop(heap)
            # 如果当前的距离大于 dp[x][y]，说明这个位置已经被更新过，不需要再次处理
            if dist > dp[x][y]:
                continue
            # 对当前点的四个方向进行处理
            for dx, dy in directions:
                nx, ny = x + dx, y + dy
                if 0 <= nx < m and 0 <= ny < n:
                    # 如果新位置的dp值可以更新（即发现更短的路径）
                    if dp[nx][ny] > dp[x][y] + 1:
                        dp[nx][ny] = dp[x][y] + 1
                        heapq.heappush(heap, (dp[nx][ny], nx, ny))
        return dp
if __name__ == "__main__":
    mat = [[0,0,0],[0,1,0],[1,1,1]]
    print(Solution().updateMatrix(mat))
```

冒泡排序

```
n = int(input())
nums = input().split()
for i in range(n - 1):
    for k in range(i+1, n):
        if nums[i] + nums[k] < nums[k] + nums[i]:
            nums[i], nums[k] = nums[k], nums[i]
ans = "".join(nums)
nums.reverse()
print(ans + " " + "".join(nums))
```

M28700: 罗马数字与整数的转换，思路：先isdigit()判断输入是否是数字，再用字典和列表转化

```
n = input()
dic = [('IV',4),('IX',9),('XL',40),('XC',90),('CD',400),('CM',900)]
d = [('I',1),('IV',4),('V',5),('IX',9),('X',10),('XL',40),('L',50),('XC',90),('C',100),('CD',400),
('D',500),('CM',900),('M',1000)]
if n[:1].isdigit():
    n = int(n)
    ans = ''
    for i in range(-1, -14, -1):
        a = n // d[i][1]
        ans += d[i][0] * a
        n -= a * d[i][1]
    print(ans)
else:
    di = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
    res = 0
    for i in range(6):
        if dic[i][0] in n:
            res += dic[i][1]
            n = n.replace(dic[i][0],'')
```

```
    for letter in n:
        res += di[letter]
    print(res)
```

02808: 校门外的树，列表处理

```
L, M = map(int, input().split())
lst = [1]*(L+1)
for i in range(M):
    a, b = map(int, input().split())
    for n in range(a, b+1):
        lst[n] = 0
print(lst.count(1))
```

T25353: 排队 （选做）类似分组，每一组每个数一定和该组其他数差别不超过d，排序后再合并（用*解包）；

```
from collections import deque
n, d = map(int, input().split())
deq = deque(int(input()) for i in range(n))
res = []
while deq:
    lst = []
    max_h = deq[0]
    min_h = deq[0]
    for i in range(len(deq)):
        height = deq.popleft()
        if abs(height - min_h)<=d and abs(height - max_h)<=d:
            lst.append(height)
        else:
            deq.append(height)
        if min_h > height:
            min_h = height
        if max_h < height:
            max_h = height
    res += sorted(lst)
print(*res,sep='\n')
```

双指针

18211: 军备竞赛 greedy, two pointers, 先将列表排序，后不断制作前面的，卖出后面的；但是双指针会简单很多

```
def make_tool(n,lst):
    ans = 0
    res = 0
    for i in range(len(lst)):
        if n - lst[0]>=0:
            ans+=1
            n-=lst[0]
            res = i
            lst = lst[1:]
        else:
            break
    return res,ans,n,lst
n = int(input())
lst = list(map(int,input().split()))
lst.sort()
if lst[0] > n:
    print(0)
else:
    res, ans,n,lst = make_tool(n, lst)
    for k in range(-1, -len(lst)-2, -1):
        try:
            if n+lst[k] >= lst[0]+lst[1]:
                n += lst[k]
                res, anss,n,lst= make_tool(n, lst)
                ans += anss-1
```

```
            else:
                if ans >0:
                    n += lst[k]
                    res, anss, n, lst = make_tool(n, lst)
                    ans += anss - 1
                else:
                    break
        except:
            break
    print(ans)
```

贪心

545C. Woodcutters，dp, greedy, 1500,按顺序算每棵树，倒下后更新坐标

```
n = int(input())
if n == 1:
    print(1)
else:
    ans = 2
    lst = []
    for i in range(n):
        lst.append(list(map(int,input().split())))
    for k in range(1,n-1):
        if lst[k][0]-lst[k-1][0]>lst[k][1]:
            ans += 1
        elif lst[k+1][0]-lst[k][0]>lst[k][1]:
            ans += 1
            lst[k][0]+=lst[k][1]
    print(ans)
```

01328: Radar Installation，greedy：从左到右计算每个岛能被覆盖的范围，有重叠就减去

```
import math
def radar(n,d,il):
    r = []
    if d < 0:
        return -1
    for x,y in il:
        if y>d:
            return -1
        h = math.sqrt(d**2-y**2)
        r.append((x-h,x+h))
    r.sort(key=lambda x:x[1])
    ans = 1
    now = r[0][1]
    for k in range(1,n):
        if r[k][0]>now:
            ans += 1
            now = r[k][1]
    return ans
num = 0
while True:
    try:
        num+=1
        il=[]
        n,d = map(int,input().split())
        for i in range(n):
            il.append(list(map(int,input().split())))
        ans = radar(n,d,il)
        input()
        print(f'Case {num}: {ans}')
    except:
        break
```

递归

sy119: 汉诺塔，recursion, https://sunnywhy.com/sfbj/4/3/119

```
n = int(input())
#@lru_cache(maxsize=None)
def hanno(n,fr,to,md):
    if n ==0:
        return
    hanno(n-1,fr,md,to)
    print(f'{fr}->{to}')
    hanno(n-1,md,to,fr)
print(2**n-1)
hanno(n,'A','C','B')
```

sy132: 全排列I，用一个列表记录每个数字是否被使用，输出一组时再从后往前逐一标记为未使用，不断递归。

```
n = int(input())
lst = [False]*(n+1)
def quanpai(n,pre=[]):
    if len(pre)==n:
        return [pre]
    res = []
    for i in range(1,n+1):
        if lst[i]:
            continue
        lst[i] = True
        res += quanpai(n, pre+[i])
        lst[i]=False
    return res
res = quanpai(n)
for r in res:
    print(' '.join(map(str,r)))
```

04123: 马走日dfs, http://cs101.openjudge.cn/practice/04123，dfs,遍历，将走过的标记，直至没有下一步

```
T=int(input())
x1 = [-2, -1, 1, 2, 2, 1, -1, -2]
y1 = [1, 2, 2, 1, -1, -2, -2, -1]
def dfs(step, x, y, ans, chess):
    if n * m == step:
        return ans + 1
    for r in range(8):
        s = x + x1[r]
        t = y + y1[r]
        if 0 <= s < n and 0 <= t < m and not chess[s][t]:
            chess[s][t] = True
            ans = dfs(step + 1, s, t, ans, chess)
            chess[s][t] = False
    return ans
for _ in range(T):
    n, m, x, y = map(int, input().split())
    chess = [[False] * T for _ in range(T)]
    chess[x][y] = True
    result = dfs(1, x, y, 0, chess)
    print(result)
#非递归
def knight_tour_non_recursive(test_cases):
    results = []
    for case in test_cases:
        n, m, start_x, start_y = case
        total_squares = n * m
        stack = [(start_x, start_y, [(start_x, start_y)])]
        paths_count = 0
        while stack:
            x, y, path = stack.pop()
```

```
            if len(path) == total_squares:
                paths_count += 1
                continue
            for dx, dy in moves:
                nx, ny = x + dx, y + dy
                if 0 <= nx < n and 0 <= ny < m and (nx, ny) not in path:
                    stack.append((nx, ny, path + [(nx, ny)]))
        results.append(paths_count)
    return results
```

02754: 八皇后，dfs and similar, 从第一行开始，判断后面每一行是否满足，不满足则右移（都不满足时会自动回溯到上一层）

```
lst = []
def queen(s):
    if len(s)==8:
        lst.append(s)
        return
    for i in range(1,9):
        if all(str(i)!=s[k] and abs(len(s)-k)!=abs(i-int(s[k])) for k in range(len(s))):
            queen(s+str(i))
queen('')
n = int(input())
for i in range(n):
    b = int(input())
    print(lst[b-1])
```

dfs

sy358: 受到祝福的平方dfs，从前往后看是否是平方数

```
def dfs(l):
    if l == len(n):
        return True
    num = 0
    for i in range(l, len(n)):
        num = num * 10 + n[i]
        if num in squares:
            if dfs(i + 1):
                return True
    return False
```

双十一dfs，暴力遍历所有商品、商店，回溯

```
result = float("inf")
n, m = map(int, input().split())
store_prices = [input().split() for _ in range(n)]
disc = [input().split() for _ in range(m)]
lst = [0] * m
def dfs(i, price):
    global result
    if i == n:
        jian = 0
        for t in range(m):
            st = 0
            for k in disc[t]:
                a, b = map(int, k.split('-'))
                if lst[t] >= a:
                    st = max(st, b)
            jian += st
        result = min(result, price - (price // 300) * 50 - jian)
        return
    for u in store_prices[i]:
        idx, p = map(int, u.split(':'))
        lst[idx-1] += p
        dfs(i + 1, price + p)
        lst[idx-1] -= p
```

```
dfs(0, 0) print(result)
```

12029: 水淹七军，dfs, 用input = sys.stdin.read一次性读入

```
import sys
sys.setrecursionlimit(300000)
input = sys.stdin.read
def dfs(x, y, height, m, n, ma, water):
    dx = [-1, 1, 0, 0]
    dy = [0, 0, -1, 1]
    for i in range(4):
        nx, ny = x + dx[i], y + dy[i]
        if is_valid(nx, ny, m, n) and ma[nx][ny] < height:
            if water[nx][ny] < height:
                water[x][y] = height
                dfs(nx, ny, height, m, n, ma, water)
def main():
    data = input().split()
    idx = 0
    k = int(data[idx])
    idx += 1
    results = []
    for _ in range(k):
        m, n = map(int, data[idx:idx + 2])
        idx += 2
        ma = []
        for i in range(m):
            ma.append(list(map(int, data[idx:idx + n])))
            idx += n
        water = [[0] * n for _ in range(m)]
        i, j = map(int, data[idx:idx + 2])
        idx += 2
        i, j = i - 1, j - 1
        p = int(data[idx])
        idx += 1
        for _ in range(p):
            x, y = map(int, data[idx:idx + 2])
            idx += 2
            x, y = x - 1, y - 1
            if ma[x][y] <= ma[i][j]:
                continue
        dfs(x, y, ma[x][y], m, n, ma, water)
        results.append("Yes" if water[i][j] > 0 else "No")
    sys.stdout.write("\n".join(results) + "\n")
if __name__ == "__main__":
    main()
```

T12757: 阿尔法星人翻译官，implementation,思路：储存hundred,遍历到thousand,million再结算

```
n = list(map(str,input().split()))
dic = {"zero": 0, "one": 1, "two": 2, "three": 3, "four": 4, "five": 5, "six": 6,
       "seven": 7, "eight": 8, "nine": 9, "ten": 10, "eleven": 11, "twelve": 12,
       "thirteen": 13, "fourteen": 14, "fifteen": 15, "sixteen": 16, "seventeen": 17,
       "eighteen": 18, "nineteen": 19, "twenty": 20, "thirty": 30, "forty": 40,
       "fifty": 50, "sixty": 60, "seventy": 70, "eighty": 80, "ninety": 90,
       "hundred": 100, "thousand": 1000, "million": 1000000}
first = 1
if n[0] == "negative":
    first = -1
    del n[0]
total = 0
t = 0
for i in n:
    if i in ("thousand", "million"):
        total += t * dic[i]
        t = 0
        continue
```

```
    if i == "hundred":
        t *= dic[i]
    else:
        t += dic[i]
print(first * (total + t))
```

矩阵

OJ18106: 螺旋矩阵，思路：循环更改上下左右四个边界，顺时针添加到列表中

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        ans = []
        if not matrix:
            return []
        a,b,c,d=0,len(matrix[0])-1,len(matrix)-1,0
        while True:
            for i in range(d,b+1):
                ans.append(matrix[a][i])
            a+=1
            if a>c:break
            for i in range(a,c+1):
                ans.append(matrix[i][b])
            b-=1
            if b<d:break
            for i in range(b,d-1,-1):
                ans.append(matrix[c][i])
            c-=1
            if a>c:break
            for i in range(c,a-1,-1):
                ans.append(matrix[i][d])
            d+=1
            if d>b:break
        return ans
```

```
n = int(input())
# 初始化变量N用于跟踪当前的方向索引，初始值为0
N = 0
# 构建(n+2) x (n+2)的矩阵'onion'，在原n x n矩阵外围添加一圈-1e9作为边界，防止越界
onion = [[-1e9 for i in range(n + 2)]] + \
        [[-1e9] + list(map(int, input().split())) + [-1e9] for i in range(n)] + \
        [[-1e9 for i in range(n + 2)]]
# 初始化移动方向为向右，起始位置在第一行第零列（实际是外围边界的内部）
dx, dy = DIRECTIONS[0]
x, y = 1, 0
# 创建一个列表来保存每一层的累加和，长度为n // 2 + 1，因为对于n x n矩阵最多有(n // 2) + 1层
layer = [0 for i in range(n // 2 + 1)]
# 开始遍历整个矩阵，总共需要遍历n * n次
for i in range(1, 1 + n * n):
    # 如果下一个位置是边界(-1e9)，则改变方向
    if onion[x + dx][y + dy] == -1e9:
        N += 1  # 改变方向，N % 4确保方向索引循环在0到3之间
        dx, dy = DIRECTIONS[N % 4]
    # 更新当前位置
    x, y = x + dx, y + dy
        # 将当前位置的值累加到对应的层中，并将当前位置标记为已访问（设置为-1e9）
    layer[N // 4] += onion[x][y]
    onion[x][y] = -1e9
# 输出最大层的总和
print(max(layer))
```

04133:垃圾炸弹，matrices, 创建一个矩阵，算出每个位置d范围内垃圾数目的和

```
d = int(input())
n = int(input())
map_1 = [[0] * 1025 for i in range(1025)]
for _ in range(n):
    x, y, i = map(int, input().split())
    for k in range(max(0, x - d), min(1025, x + d + 1)):
        for j in range(max(0, y - d), min(1025, y + d + 1)):
            map_1[k][j] += i
max_num = max(max(l) for l in map_1)
num = sum(l.count(max_num) for l in map_1)
print(num, max_num)
```

dp

26976:摆动序列, dp,对每个数前面是增还是减分别考虑

```
        if len(nums)<2:
            return len(nums)
        up = [1] + [0] * (len(nums) - 1)
        down = [1] + [0] * (len(nums) - 1)
        for i in range(1, len(nums)):
            if nums[i] > nums[i - 1]:
                up[i] = max(up[i - 1], down[i - 1] + 1)
                down[i] = down[i - 1]
            elif nums[i] < nums[i - 1]:
                up[i] = up[i - 1]
                down[i] = max(up[i - 1] + 1, down[i - 1])
            else:
                up[i] = up[i - 1]
                down[i] = down[i - 1]
        return max(up[len(nums) - 1], down[len(nums) - 1])
```

CF455A: Boredom，dp, 1500, https://codeforces.com/contest/455/problem/A

思路：dp,计算每个数出现次数，求不相邻的数的最大和

```
n = int(input())
lst=list(map(int,input().split()))
m = max(lst)
num = [0]*(m+1)
for i in lst:
    num[i]+=1
dp = [[0, 0] for i in range(m+ 1)]
for i in range(1, m + 1):
    dp[i][0] = max(dp[i - 1][0], dp[i - 1][1]) #dp[i][0]表示不选择数字i时的最大和
    dp[i][1] = dp[i - 1][0] + num[i] * i
print(max(dp[m][0], dp[m][1]))
```

02287: Tian Ji ， dp,将田忌和齐王的马速度逆序排列，田忌前i匹马和齐王前j匹马根据（i-1,j),(i-1,j-1),(i,j-1)得出

```
from functools import lru_cache
@lru_cache(maxsize=None)
def horse():
    while True:
        n = int(input())
        if n == 0:
            break
        a = sorted([int(x) for x in input().split()], reverse=True)
        b = sorted([int(x) for x in input().split()], reverse=True)
        c = [[0]*(n+1) for _ in range(n+1)]
        for i in range(1, n+1):
            for j in range(1, n+1):
                if a[i-1] > b[j-1]:
                    c[i][j] = max(c[i-1][j], c[i][j-1], c[i-1][j-1]+2)
                elif a[i-1] == b[j-1]:
                    c[i][j] = max(c[i-1][j], c[i][j-1], c[i-1][j-1]+1)
```

```
                else:
                    c[i][j] = max(c[i-1][j], c[i][j-1], c[i-1][j-1])
        print((c[n][n]-n)*200)
horse()
```

马拉车

```
def manacher(s):
    # 1.预处理字符串
    t = '^#' + '#'.join(s) + '#$'  # 字符间插入#,从而对于偶数子串也可以中心扩展
    n = len(t)  # 得到新字符串长度
    P = [0] * n  # P[i]表示以t[i]为中心的回文半径
    C, R = 0, 0  # C为当前回文中心,R为当前回文的右边界
    # 2.计算回文半径
    for i in range(1, n - 1):  # i位置为中心
        # 如果 i 在 R 范围内，用对称位置的回文半径初始化 P[i]
        P[i] = min(R - i, P[2 * C - i]) if i < R else 0
        # 中心扩展，尝试扩展回文半径
        while t[i + P[i] + 1] == t[i - P[i] - 1]:
            P[i] += 1
        # 更新回文的中心和右边界
        if i + P[i] > R:
            C, R = i, i + P[i]
    # 3.找到最长回文
    max_len = max(P)  # 最长回文半径
    center_index = P.index(max_len)  # 最长回文对应的中心索引
    # 原始字符串中的起始索引
    start = (center_index - max_len) // 2
    return s[start:start + max_len]
```

数论

```
# 计算最大公约数
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
# 计算最小公倍数
def lcm(a, b):
    return a * b // gcd(a, b)
# 扩展欧几里得算法 （使用递归方式实现扩展欧几里得算法。返回最大公约数 gcd 以及满足 a·x+b·y=gcd(a,b) 的整数 x 和 y）
def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    else:
        gcd, x1, y1 = extended_gcd(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return gcd, x, y
# 计算模逆元（）
def mod_inverse(a, m):
    gcd, x, y = extended_gcd(a, m)
    if gcd != 1:
        raise ValueError(f"No modular inverse for {a} modulo {m}")
    else:
        # 确保结果为正数
        return x % m
# 使用中国剩余定理求解
def find_next_triple_peak(p, e, i):
    m1, m2, m3 = 23, 28, 33
    M = lcm(lcm(m1, m2), m3)
    M1, M2, M3 = M // m1, M // m2, M // m3
    y1, y2, y3 = mod_inverse(M1, m1), mod_inverse(M2, m2), mod_inverse(M3, m3)
    n = (p * M1 * y1 + e * M2 * y2 + i * M3 * y3) % M（就是“三人同行七十稀”那个公式）
    return n, M
# 主程序
m = 1
```

```
while True:
    p, e, i, d = map(int, input().split())
    if p == -1 and e == -1 and i == -1 and d == -1:
        break
    # 找到从d开始的下一个三重峰值
    n, M = find_next_triple_peak(p, e, i)
    if n <= d:
        # 如果n小于d，我们需要加上最小公倍数
        n += M
    print(f"Case {m}: the next triple peak occurs in {n - d} days.")
    m += 1
```