Cheat sheet for 雷薇霓，数学科学学院 （ 26/12 计算概论上机期末）

拿取 input 的几种方式

1. Matrices (list , map)
   rows, cols = map(int,input().split())
   matrix= [list(map(int,input().split())) for _ in range (rows)]
2. Int(x) [same as 4]
   man= [int(x) for x in input().split()]
3. list,map
   Police Recruit
   n= int,input()
   a= list.map(int,input().split())

Basic Ways to Read Input in Python

1.Reading a Single Integer
```
n = int(input())
```
2.Reading a Single String
```
s = input()
```
3.Reading Multiple Space-Separated Integers as a List
```
nums = list(map(int, input().split()))
```
4.Reading Two Integers (e.g., for a pair of coordinates)
```
x, y = map(int, input().split())
```
5.Reading Multiple Lines of Input
```
lines = []
for _ in range(n):  # 'n' is the number of lines
    lines.append(input())
```
6.Reading a 2D List (Matrix)
```
matrix = [list(map(int, input().split())) for _ in range(rows)]
```
几个重要 Knowledge

1. Sort （ 小到大排列）
   ss=s.sort()
2. Len （长度）
3. Import math(math.ceil, math.floor)

```
N,m,n= [int(x) for x in input().split()]

P= math.ceil(n/a)

Q= math.ceil(m/a)

Print(p*q)
```

4. 向下取整 (//)
5. Lower Case
   S1.lower>s2.lower
6. a.index (index start from 0,1,2,….)
   (Beautiful Matrices 让 1 到中间的)
   a= input().split()
   if '1' in a:
   print (abs(a.index('-1')-2)+2)
7. float
8. append (加) (2048)

#Police Recruit
```
n= int(input())

a= list(map(int,input().split()))

case=0

officers=0

for i in range a:

   if i== -1 and officers ==0:

      case +=1

      continue

    elif i>0:

      officers +=i

       continue

  officers -=1

 print(case)
```

```
n,m = map(int,input().split())
```

01003: Hangover
```
import math

while True:
    n = float(input())
    if math.isclose(n, 0.00, rel_tol=1e-5) :
        break
```

```
    cnt = 0
    tot = 0
    while  True:
        cnt += 1
        tot += 1/(1+cnt)
        if tot>n:
            break

    print(cnt, "card(s)")
```

01218: THE DRUNK JAILER
```
for _ in range(int(input())):
    n = int(input())
    lst = [0]*n

    for j in range(2,n+1):
        for i in range(j-1,n,j):
            lst[i] ^= 1

    print(lst.count(0))
```

01922: Ride to School
```
import math

while True:
    n = int(input())
    if n == 0:
        break

    max_time = float("inf")
    for _ in range(n):
        speed, time = map(int, input().split())
        if time < 0:
            continue
        arrival_time = math.ceil((4500 / speed) * 3.6 + time)
        max_time = min(max_time, arrival_time)

    print(max_time)
```

02159: Ancient Cipher
```
def main():
    import sys
    input = sys.stdin.read
    data = input().strip().split()

    str1 = data[0]
    str2 = data[1]

    cnt1 = [0] * 26
    cnt2 = [0] * 26
```

```
    for char in str1:
        cnt1[ord(char) - ord('A')] += 1

    for char in str2:
        cnt2[ord(char) - ord('A')] += 1

    cnt1.sort()
    cnt2.sort()

    if cnt1 == cnt2:
        print("YES")
    else:
        print("NO")

if __name__ == "__main__":
    main()
```

02707: 求一元二次方程的根
```
import math
n = int(input())
for i in range(n):
    a, b, c = map(float, input().split())
    if b == 0:
        b = -b
    delta = b ** 2 - 4 * a * c
    if delta > 0:
        x1 = (-b + math.sqrt(delta)) / (2 * a)
        x2 = (-b - math.sqrt(delta)) / (2 * a)
        print(f"x1={x1:.5f};x2={x2:.5f}")
    elif delta == 0:
        t = (-b) / (2 * a)
        print(f"x1=x2={t:.5f}")
    else:
        d = math.sqrt(-delta) / (2 * a)
        re = (-b) / (2 * a)
        print(f"x1={re:.5f}+{d:.5f}i;x2={re:.5f}-{d:.5f}i")
```

02734: 十进制到八进制
```
decimal = int(input())  # 读取十进制数

# 创建一个空栈
stack = []

# 特殊情况：如果输入的数为0，直接输出0
if decimal == 0:
    print(0)
else:
    # 不断除以8，并将余数压入栈中
    while decimal > 0:
        remainder = decimal % 8
        stack.append(remainder)
        decimal = decimal // 8

    # 依次出栈，构成八进制数的各个位
    octal = ""
    while stack:
        octal += str(stack.pop())

    print(octal)
```

02748:全排列
```
def dfs(s, path, used, res):
    if len(path) == len(s):
        res.append(''.join(path))
        return
    for i in range(len(s)):
        if not used[i]:
            used[i] = True
            path.append(s[i])
            dfs(s, path, used, res)
            path.pop()
            used[i] = False

def print_permutations(s):
    res = []
    dfs(s, [], [False]*len(s), res)
    for perm in sorted(res):
        print(perm)

# Test the function
print_permutations(input())
```

02753: 菲波那契数列
```
def f(n):
    if n <= 2:
        return 1
    else:
        return f(n-1)+f(n-2)


n = int(input())
ans = []
for _ in range(n):
    num = int(input())
    ans.append(f(num))

print('\n'.join(map(str, ans)))
```

02783: Holiday Hotel
```
while True:
    n=int(input())
    if n==0:
        break
    hotels=[tuple(map(int,input().split())) for _ in range(n)]
    hotels.sort(key=lambda x:(x[0],x[1]))
    candidates=1
    max_cost_so_far=hotels[0][1]
    for i in range(n):
        if hotels[i][1]<max_cost_so_far:
            candidates+=1
            max_cost_so_far=hotels[i][1]
    print(candidates)
```

02786: Pell 数列
```
dp = [0]*(1000000+1)
dp[1], dp[2] = 1, 2
for i in range(3, 1000000+1):
    dp[i] = (2*dp[i-1] + dp[i-2])%32767

for _ in range(int(input())):
    k = int(input())
    print(dp[k])
```

02792:集合加法
```
from collections import Counter

def calculate_pairs(arr1, arr2, target_sum):
    counter1 = Counter(arr1)
    counter2 = Counter(arr2)

    ans = 0
    for item in counter1:
        if target_sum - item in counter2:
            ans += counter1[item] * counter2[target_sum - item]

    return ans

for _ in range(int(input())):
    s = int(input())
    input()
    l1 = list(map(int, input().split()))
    input()
    l2 = list(map(int, input().split()))

    ans = calculate_pairs(l1, l2, s)
    print(ans)
```

02806:公共子序列
```
while True:
    try:
```

```
        a, b = input().split()
    except EOFError:
        break

    alen = len(a)
    blen = len(b)

    dp = [[0]*(blen+1) for i in range(alen+1)]

    for i in range(1, alen+1):
        for j in range(1, blen+1):
            if a[i-1]==b[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])


    print(dp[alen][blen])
```

02810:完美立方

```
n = int(input())
cube = {i**3: i for i in range(2, n+1)}
reversed_cube = {v: k for k, v in cube.items()}
ans = []
for b in range(2, n):
    for c in range(b, n):
        for d in range(c, n):
            if (a :=
reversed_cube[b]+reversed_cube[c]+reversed_cube[d]) in cube:
                ans.append((cube[a], b, c, d))
ans.sort()
for s in ans:
    print(f"Cube = {s[0]}, Triple = ({s[1]},{s[2]},{s[3]})")
```

02913: 加密技术

```
def encrypt(text):
    # 数字序列"4962873"
    pattern = "4962873"
    encrypted_text = []
    for i, char in enumerate(text):
        # ASCII码范围限制在32到122之间，超出范围进行模运算
        shift = int(pattern[i % len(pattern)])
        new_char = chr((ord(char) + shift - 32) % (122 - 32 +
1) + 32)
        encrypted_text.append(new_char)
    return ''.join(encrypted_text)

def decrypt(encrypted_text):
    # 数字序列"4962873"
    pattern = "4962873"
    decrypted_text = []
    for i, char in enumerate(encrypted_text):
        # 解密时反向操作
        shift = int(pattern[i % len(pattern)])
        new_char = chr((ord(char) - shift - 32) % (122 - 32 +
1) + 32)
        decrypted_text.append(new_char)
    return ''.join(decrypted_text)


text = input()

encrypted = encrypt(text)
print(encrypted)

decrypted = decrypt(encrypted)
print(decrypted)
```

02981:大整数加法

```
s1 = list(reversed(list(map(int, list(input())))))
s2 = list(reversed(list(map(int, list(input())))))
if len(s1) > len(s2):
    s2 += [0]*(len(s1) - len(s2))
else:
    s1 += [0]*(len(s2) - len(s1))

sum = [0]*(len(s1) + 1)
for i in range(len(s1)):
    sum[i+1] += (sum[i]+s1[i]+s2[i])//10
    sum[i] = (s1[i]+s2[i]+sum[i])%10

sum.reverse()
cur = 0
while cur<len(sum) and sum[cur] == 0:
    cur += 1
out = ''
if cur == len(sum):
    print(0)
else:
    for i in range(cur,len(sum)):
        out += str(sum[i])
    print(out)
```

03248: 最大公约数

```
def comfac(a, b):
    n = 1
    for i in range(1, min(a, b) + 1):
        if a % i == 0 and b % i == 0:
            n = i
    return n

while True:
    try:
        a, b = map(int, input().split())
    except:
        break
    print(comfac(a, b))
```

03253: 约瑟夫问题No.2

```
def josephus(n, p, m):
    if n == 1:
        return [1]

    out_order = []
    pos = p - 1  # 将起始位置调整为0-based索引
    kids = list(range(1, n + 1))  # 孩子们的编号列表

    while len(kids) > 0:
        pos = (pos + m - 1) % len(kids)  # 计算当前出圈的孩子的位
置
        out_order.append(kids.pop(pos))  # 将出圈的孩子添加到结果
列表中

    return out_order

while True:
    n, p, m = map(int, input().split())
    if n + p + m == 0:
        break
    result = josephus(n, p, m)
    print(','.join(map(str, result)))
```

03704:括号匹配

```
lines = []
while True:
    try:
        lines.append(input())
    except EOFError:
        break

ans = []
for s in lines:
    stack = []
    Mark = []
    for i in range(len(s)):
        if s[i] == '(':
            stack.append(i)
            Mark += ' '
        elif s[i] == ')':
            if len(stack) == 0:
                Mark += '?'
            else:
                Mark += ' '
                stack.pop()
        else:
            Mark += ' '

    while(len(stack)):
        Mark[stack[-1]] = '$'
        stack.pop()


    print(s)
    print(''.join(map(str, Mark)))
```

04067:回文数字

```
def isPalindrome(s):
    if len(s) < 1:
        return False
    if len(s) == 1:
        return True

    front = 0
    back = len(s) - 1
    while front < back:
        if s[front] != s[back]:
            return False
        else:
            front += 1
            back -= 1

    return True

while True:
    try:
        s = input()
        print('YES' if isPalindrome(s) else 'NO')
    except:
        break
```

04070:全排列

```
def dfs(n, path, used, res):
    if len(path) == n:
        res.append(path[:])
        return
    for i in range(1, n+1):
        if not used[i]:
            used[i] = True
            path.append(i)
            dfs(n, path, used, res)
            path.pop()
            used[i] = False

def print_permutations(n):
    res = []
    dfs(n, [], [False]*(n+1), res)
    for perm in sorted(res):
        print(' '.join(map(str, perm)))

nums = []
while True:
    num = int(input())
    if num == 0:
        break
    nums.append(num)

for num in nums:
    print_permutations(num)
```

04099:队列和栈

```
m = int(input())
for _ in range(m):
    queue = []
    stack = []
    error = False
    n = int(input())
    for _ in range(n):
        operation = input().split()
        if operation[0] == 'push':
            queue.append(int(operation[1]))
            stack.append(int(operation[1]))
        elif operation[0] == 'pop':
            if queue:
                queue.pop(0)
            else:
                error = True
            if stack:
                stack.pop()
            else:
                error = True
    if error:
        print('error')
        print('error')
    else:
        print(' '.join(map(str, queue)))
        print(' '.join(map(str, stack)))
```

04109:公共朋友-Common Friends

```
def count_common_friends(n, m, k, friend_connections,
queries):
    # Create a dictionary to store friend connections
    friends_dict = {}
    for i in range(1, n + 1):
        friends_dict[i] = set()

    # Update the dictionary with friend connections
    for i, j in friend_connections:
        friends_dict[i].add(j)
        friends_dict[j].add(i)

    # Count common friends for each query
    results = []
    for i, j in queries:
        common_friends =
len(friends_dict[i].intersection(friends_dict[j]))
        results.append(common_friends)

    return results


def main():
    test_cases = int(input())
    for case in range(1, test_cases + 1):
        n, m, k = map(int, input().split())
        friend_connections = []
        queries = []

        # Read friend connections
        for _ in range(m):
            i, j = map(int, input().split())
            friend_connections.append((i, j))

        # Read queries
        for _ in range(k):
            i, j = map(int, input().split())
            queries.append((i, j))

        # Count common friends and output the results
        print(f"Case {case}:")
        results = count_common_friends(n, m, k,
friend_connections, queries)
        for result in results:
            print(result)


if __name__ == "__main__":
    main()
```

04138: 质数的和与积

```
S = int(input())

def isprime(n):
    for i in range(2, int(n**.5) + 1):
        if n%i == 0:
            return False
    else:
        return True

maxmultiple = 0
f = 2
while f<S:
    if isprime(S - f):
        maxmultiple = max(maxmultiple, f*(S-f))
```

```
        f += 1
        while isprime(f)==False:
            f += 1

    print(maxmultiple)
```

04141:砝码称重

```
    weights = [1, 2, 3, 5, 10, 20]

    def dfs(index, cur_w):
     # 已尝试所有可能砝码，递归结束
        if index == 6:
            if cur_w != 0:
                w.add(cur_w)
            return
        #遍历所有可能的使用该砝码个数
        for i in range(max_w[index]+1):
            dfs(index+1, cur_w+i*weights[index])


    max_w = list(map(int, input().split()))
    #使用set自动去重
    w = set()
    dfs(0, 0)
    print(f'Total={len(w)}')
```

04146:数字方格

```
    n = int(input())
    m=0
    for i in range(n, -1, -1):
        for j in range(n, -1, -1):
            for k in range(n, -1, -1):
                if (i + j) % 2 == 0 and (j + k) % 3 == 0 and (i +
j + k) % 5 == 0:
                    m=max(m,i+j+k)
    print(m)
```

04147: 汉诺塔问题 (tower of Hanoi)

```
    # https://blog.csdn.net/geekwangminli/article/details/7981570

    # 将编号为numdisk的盘子从init杆移至desti杆
    def moveOne(numDisk : int, init : str, desti : str):
        print("{}:{}->{}".format(numDisk, init, desti))

    #将numDisks个盘子从init杆借助temp杆移至desti杆
    def move(numDisks : int, init : str, temp : str, desti :
str):
        if numDisks == 1:
            moveOne(1, init, desti)
        else:
            # 首先将上面的 (numDisk-1) 个盘子从init杆借助desti杆移至
temp杆
            move(numDisks-1, init, desti, temp)

            # 然后将编号为numDisks的盘子从init杆移至desti杆
            moveOne(numDisks, init, desti)

            # 最后将上面的 (numDisks-1) 个盘子从temp杆借助init杆移至
desti杆
            move(numDisks-1, temp, init, desti)

    n, a, b, c = input().split()
    move(int(n), a, b, c)
```

04148:生理周期

```
    def shenglizhouqi():
        case=1
        while True:
            p, e, i, d = map(int, input().split())
            if p == e == i == d == -1:
                break

            # 算出出现再周期的那一天
            p_ = p % 23
            e_ = e % 28
            i_ = i % 33
            for a in range(d + 1, d+21253):
                if a % 23 == p_ and a % 28 == e_ and a % 33 ==
i_:
                    print(f'Case {case}: the next triple peak
occurs in {a-d} days.')
                    case+=1
                    break
    shenglizhouqi()
```

05902:双端队列

```
    from collections import deque

    for _ in range(int(input())):
        n=int(input())
        q=deque([])
        for i in range(n):
            a,b=map(int,input().split())
            if a==1:
                q.append(b)
            else:
                if b==0:
                    q.popleft()
                else:
                    q.pop()
        if q:
            print(*q)
        else:
            print('NULL')
```

06374:文字排版

```
    int(input())
    L = input().split()

    ans = []
    tmp = L[0] + ' '
    for i in L[1:]:
        if len(tmp) + len(i) > 80:
            ans.append(tmp.rstrip())
            tmp = i + ' '
        else:
            tmp += i + ' '
    else:
        ans.append(tmp.rstrip())

    print('\n'.join(ans))
```

07618:病人排队

```
    # Read the number of patients
    n = int(input())

    # Initialize lists for elderly and non-elderly patients
    elderly = []
    non_elderly = []

    # Read patient information
    for _ in range(n):
        patient_id, age = input().split()
        age = int(age)
        if age >= 60:
            elderly.append((patient_id, age))
        else:
            non_elderly.append((patient_id, age))

    # Sort elderly patients by age in descending order
    elderly.sort(key=lambda x: -x[1])

    # Concatenate elderly and non-elderly lists
    sorted_patients = elderly + non_elderly

    # Print the sorted patient IDs
    for patient in sorted_patients:
        print(patient[0])
```

12556:编码字符串

```
    sentence = input().lower()
    list = []

    pre = sentence[0]
    count = 1
    for i in range(1, len(sentence)):
     if sentence[i] != pre:
         list.append('(' + pre + ',' + str(count) + ')')
         pre = sentence[i]
         count = 1
     else:
         count += 1
    list.append('(' + pre + ',' + str(count) + ')')
    print(''.join(list))
```

12065:方程求解

```
    left = 0
    right = 10
    eps = 1e-12

    func = lambda x:x**3 - 5*(x**2) + 10*x - 80
    while right - left > eps:
        mid = (left + right)/2
        if func(mid) > 0:
            right = mid
        else:
            left = mid
    print(format(right, ".9f"))
```

16529:股票

```
    def max_profit(prices):
        # 保证有至少两天，否则无法完成一次买卖
        if len(prices) < 2:
            return 0.0

        # 初始化最小价格为第一天价格，最大利润为0
        min_price = prices[0]
        max_profit = 0.0

        # 遍历价格数组
        for price in prices:
            # 更新最小价格
            min_price = min(min_price, price)

            # 当前价格与最低购买价格比较，当前可能获得的最大利润
            profit = price / min_price

            # 更新最大利润
            max_profit = max(max_profit, profit)


        return 100 * max_profit

    # 样例输入
    days = int(input().strip())
    prices = list(map(float, input().split()))

    # 计算最大利润并输出，保留两位小数
    result = max_profit(prices)
    print("{:.2f}".format(result))
```

18224: 找魔数

```
    l = []
    i = 1
    while i*i < 1000:
     l.append(i*i)
     i += 1

    magics = []
    for i in l:
     for j in l:
         magics.append(i+j)


    m = int(input())
    x = [int(i) for i in input().split()]

    for num in x:
     if num in magics:
         print(bin(num), oct(num), hex(num))
```

19963:买学区房

```
    import statistics
    n = int(input())
    distance = list(map(lambda x:sum(eval(x)),input().split()))
    prize = list(map(int,input().split()))
    average = list(distance[i]/prize[i] for i in range(n))
    prize_median = statistics.median(prize)
    average_median = statistics.median(average)
    num = 0
    for i in range(n):
        if average[i] > average_median and prize[i] <
prize_median:
            num += 1
    print(num)
```

19757: Saruman's Army

```
    while True:
        R, N = map(int, input().split())
        if R == -1 and N == -1:
            break

        X = [int(i) for i in input().split()]
        X.sort()
        i=0
        ans = 0
        while i < N:
            s = X[i]
            i += 1
            while i < N and X[i] <= s + R:
                i += 1
            p = X[i-1]  # position labeled
            while i < N and X[i] <= p + R:
                i += 1

            ans += 1

        print(ans)
```

20352: 找出全部子串位置

```
    n = int(input())
    for _ in range(n):
        s1, s2 = input().split()
        positions = []
        start = 0
        while True:
            pos = s1.find(s2, start)
            if pos == -1:
                break
            positions.append(pos)
            start = pos + len(s2)
        if positions:
            for pos in positions:
                print(pos, end=" ")
            print("")
        else:
            print("no")
```

21462:加密的称赞

```
    n = int(input())

    s = [[-1]*(n+2)]
    mx = s + [[-1]+ [int(x) for x in input().split()]+[-1] for _
in range(n)] + s

    row = 1
    col = 1

    dirL = [[1,0], [0,1], [-1,0], [0,-1]]
    N = 0

    drow, dcol = dirL[0]

    for _ in range(1, n*n+1):
        if mx[row][col]!=0:
            print(chr(mx[row][col]), end='')
            mx[row][col] = -1

        if mx[row+drow][col+dcol]==-1:
```

```
            N += 1
            drow, dcol = dirL[N%4]

        row += drow
        col += dcol
```

21532:数学密码

```
n=int(input())
i = 1 + 2 + 3
while n%i != 0:
    i += 1
else:
    print(n // i)
```

21759:P大卷王查询系统

```
# gpt
def find_juanwang(n, x, y, grades, m, queries):
    # 创建一个字典用于存储学生的课程和成绩
    student_grades = {}

    # 遍历成绩单，将学生的成绩添加到字典中
    for i in range(n):
        course, student, grade = grades[i]
        if student not in student_grades:
            student_grades[student] = []
        student_grades[student].append(grade)

    # 遍历查询列表，判断每个学生是否为卷王
    results = []
    for i in range(m):
        student = queries[i]
        if student in student_grades and
len(student_grades[student]) >= x:
            average_grade = sum(student_grades[student]) /
len(student_grades[student])
            if average_grade > y:
                results.append("yes")
            else:
                results.append("no")
        else:
            results.append("no")

    return results

# 读取输入
n, x, y = map(int, input().split())
grades = []
for _ in range(n):
    course, student, grade = input().split()
    grade = int(grade)
    grades.append((course, student, grade))
```

```
m = int(input())
queries = []
for _ in range(m):
    query = input()
    queries.append(query)

# 调用函数进行查询
results = find_juanwang(n, x, y, grades, m, queries)

# 输出结果
for result in results:
    print(result)
```

22491:冲刺GPA的贪心之路

```
def max_gpa_increase(h, courses):
    # 总复习时间，扣除每门课的基础复习时间
    total_time = 2 * h - 0.5 * len(courses)

    # 计算每门课程的性价比：每增加一小时复习时间所能提高的分数乘以学分
    for course in courses:
        course.append(course[0] * course[1])  # 将性价比添加到每
个课程的信息中

    # 按性价比从高到低排序课程
    courses.sort(key=lambda x: -x[2])

    total_increase = 0  # 初始化总分提高
    for course in courses:
        if total_time <= 0:
            break
        # 计算当前课程最多可以分配的复习时间
        max_time_for_course = min(5 / course[0], total_time)
        total_time -= max_time_for_course
        # 计算当前课程的分数提高并累加到总分提高
        total_increase += max_time_for_course * course[0] *
course[1]

    return total_increase


# 输入
h = int(input())
m = int(input())
courses = []
for _ in range(m):
    s, c = map(float, input().split())
    courses.append([s, c])

# 输出
print(f"{max_gpa_increase(h, courses):.1f}")
```

22548:机智的股民老张

```
*a, = map(int, input().split())
min_price = float('inf')
max_profit = 0

for price in a:
    min_price = min(min_price, price)  # 更新最小值
    max_profit = max(max_profit, price - min_price)  # 更新最大
利润

print(max_profit)
```

23421:小偷背包

```
n,b=map(int, input().split())
price=[0]+[int(i) for i in input().split()]
weight=[0]+[int(i) for i in input().split()]
bag=[[0]*(b+1) for _ in range(n+1)]
for i in range(1,n+1):
    for j in range(1,b+1):
        if weight[i]<=j:
            bag[i][j]=max(price[i]+bag[i-1][j-weight[i]],
bag[i-1][j])
        else:
            bag[i][j]=bag[i-1][j]
print(bag[-1][-1])
```

23554:小朋友春游

```
# Read input
n = int(input().strip())
remaining_children = list(map(int, input().strip().split()))

# Initialize sets
original_class = set(range(1, n + 1))
remaining_set = set(remaining_children)

# Calculate missing children
missing_children = sorted(original_class - remaining_set)

# Calculate other class children
other_class_children = sorted([child for child in
remaining_children if child > n])

# Print results
print(" ".join(map(str, missing_children)))
print(" ".join(map(str, other_class_children)))
```

23563:多项式时间复杂度

```
ps = input().split('+')
ns = [i.split('n^') for i in ps]
result = 0
```

```
for i in ns:
    if int(i[1]) > result and i[0] != '0':
        result = int(i[1])
print('n^{}'.format(result))
```

24588:后序表达式求值

```
def evaluate_postfix(expression):
    stack = []
    tokens = expression.split()

    for token in tokens:
        if token in '+-*/':
            # 弹出栈顶的两个元素
            right_operand = stack.pop()
            left_operand = stack.pop()
            # 执行运算
            if token == '+':
                stack.append(left_operand + right_operand)
            elif token == '-':
                stack.append(left_operand - right_operand)
            elif token == '*':
                stack.append(left_operand * right_operand)
            elif token == '/':
                stack.append(left_operand / right_operand)
        else:
            # 将操作数转换为浮点数后入栈
            stack.append(float(token))

    # 栈顶元素就是表达式的结果
    return stack[0]

# 读取输入行数
n = int(input())

# 对每个后序表达式求值
for _ in range(n):
    expression = input()
    result = evaluate_postfix(expression)
    # 输出结果，保留两位小数
    print(f"{result:.2f}")
```

25301:生日相同

```
birthday_list = [[[] for j in range(32)] for i in range(13)]
for _ in range(int(input())):
    student_id,m,d = input().split()
    m = int(m); d = int(d)
    birthday_list[m][d].append(student_id)
for month in range(1,13):
    for day in range(1,32):
        if len(birthday_list[month][day]) > 1:
            print(month, day, *birthday_list[month][day])
```

24834:通配符匹配

```
#23n2300017735(夏天明BrightSummer)
import re

for i in range(int(input())):
    s, p = input(), input().replace("?", ".{1}").replace("*",
".*") + "$"
    print("yes" if re.match(p, s) else "no")
```

26999:2023找出全部子串位置

```
n = int(input())
for _ in range(n):
    s1, s2 = input().split()
    positions = []
    start = 0
    while True:
        pos = s1.find(s2, start)
        if pos == -1:
            break
        positions.append(pos)
        start = pos + 1
    if positions:
        for pos in positions:
            print(pos, end=" ")
        print("")
    else:
        print("no")
```

27273:简单数学题

```
import math
t = int(input())
for _ in range(t):
    n = int(input())
    if n % 2 == 1:
        sumv = (1 + n - 1)*(n-1)//2 + n
    else:
        sumv = (1 + n)*n//2

    maxp = int(math.log2(n))

    for i in range(maxp+1):
        sumv -= 2*(2**i)

    print(sumv)
```

27274:字符串提炼

```
import math
s = input()

slen = len(s)
maxp = int(math.log2(slen))
```

```
extracted = ""
for i in range(maxp+1):
    extracted += s[2**i - 1]

left, right = 0, len(extracted)-1
ns = ""
while left < right:
    ns = ns + extracted[left] + extracted[right]
    left += 1
    right -= 1

if len(extracted) % 2 != 0:
    ns += extracted[right]
print(ns)
```

27300:模型整理

```
from collections import defaultdict

n = int(input())
d = defaultdict(list)
for _ in range(n):
    #name, para = input().strip().split('-')
    name, para = input().split('-')
    if para[-1]=='M':
        d[name].append((para, float(para[:-1])/1000) )
    else:
        d[name].append((para, float(para[:-1])))


sd = sorted(d)
#print(d)
for k in sd:
    paras = sorted(d[k],key=lambda x: x[1])
    #print(paras)
    value = ', '.join([i[0] for i in paras])
    print(f'{k}: {value}')
```

27301:给植物浇水

```
def mininumRefill(plants, capacityA, capacityB):
    l, r = 0, len(plants) - 1
    Alice, Bob = capacityA, capacityB
    ans = 0
    while l <= r:
        if l == r:
            if Alice >= plants[l] or Bob >= plants[r]:
                break

            Alice = capacityA
            ans += 1
            if Alice >= plants[l]:
                break
            ans -= 1
```

```
                Bob = capacityB
                ans += 1
                if Bob >= plants[r]:
                    break

            if Alice < plants[l]:
                Alice = capacityA
                ans += 1

            if Bob < plants[r]:
                Bob = capacityB
                ans += 1

            if Alice >= plants[l]:
                Alice -= plants[l]
                l += 1
            if Bob >= plants[r]:
                Bob -= plants[r]
                r -= 1

        return ans

    n, AliceRaw, BobRaw = map(int, input().split())
    *plants, = map(int, input().split())
    print(mininumRefill(plants, AliceRaw, BobRaw))
```

28664:验证身份证号

```
    l = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
    n = int(input())
    for _ in range(n):
        s = input()
        if len(s) != 18:
            print('NO')
            continue
        x = sum(int(s[i]) * l[i] for i in range(17)) % 11
        x = (12 - x) % 11
        if x == 10:
            x = 'X'
        if s[17] == str(x):
            print('YES')
        else:
            print('NO')
```

28678:角谷猜想

```
    def collatz_sequence(n):
        if n == 1:
            print("End")
            return

        while n != 1:
            if n % 2 == 1:
                next_n = 3 * n + 1
                print(f"{n}*3+1={next_n}")
            else:
                next_n = n // 2
                print(f"{n}/2={next_n}")
            n = next_n

        print("End")

    # Sample input
    n = int(input())

    # Calculate and print the result
    collatz_sequence(n)
```

28691:字符串中的整数求和

```
    def main():
        import sys
        input = sys.stdin.read
        data = input().strip().split()

        num1 = int(data[0][:2])
        num2 = int(data[1][:2])

        result = num1 + num2
        print(result)

    if __name__ == "__main__":
        main()
```

01008: Maya Calendar

```
    month1 =\
        "pop, no, zip, zotz, tzec, xul, yoxkin, mol, chen, yax,
zac, ceh, mac, kankin, muan, pax, koyab, cumhu".split(", ")
    month2 =\
        "imix, ik, akbal, kan, chicchan, cimi, manik, lamat,
muluk, ok, chuen, eb, ben, ix, mem, cib, caban, eznab, canac,
ahau".split(", ")
    chk1 = {}
    for i in range(18): chk1[month1[i]] = i
    chk1["uayet"] = 18
    n = int(input()); print(n)
    for i in range(n):
        x, y1, y2 = input().split()
        day = int(y2) * 365 + chk1[y1] * 20 + int(x.rstrip('.'))
        print("{} {} {}".format(day % 13 + 1, month2[day % 20],
day // 260))
```

01011:Sticks

```
    def dfs(unused, left, len):
        if unused == 0 and left == 0:
            return True
        if left == 0:
            left = len
        for i in range(N):
            if used[i] == False and length[i] <= left:
                if i > 0:
                    if used[i - 1] == False and length[i] ==
length[i - 1]:
                        continue  # 不要在同一个位置多次尝试相同长度的
木棒，剪枝1

                used[i] = True
                if dfs(unused - 1, left - length[i], len):
                    return True
                used[i] = False


                if length[i] == left or left == len:
                    break

        return False


    while True:
        N = int(input())
        if N == 0:
            break

        length = [int(x) for x in input().split()]
        length.sort(reverse=True)  # 排序是为了从长到短拿木棒进行尝试

        totalLen = sum(length)

        for L in range(length[0], totalLen//2 + 1):
            if totalLen % L:
                continue  # 不是木棒长度和的因子的长度，直接否定

            used = [False] * 65
            if dfs(N, 0, L):
                print(L)
                break
        else:
            print(totalLen)
```

01019: Number Sequence

```
    # 生成递增的字符串序列
    sequence = ""
    s = 0
    ss = 0
    sums = []


    for j in range(1, 33000):
        sequence += str(j)  # 将当前数字转换为字符串并追加到序列中
        s += len(str(j))
        ss += s  # 累加当前数字的长度
        sums.append(ss)  # 将累加和添加到列表中

    #print(ss)

    # 处理测试用例
    test_cases = int(input())

    for _ in range(test_cases):
        x = int(input())

        if x == 1:
            print(1)
        else:
            # 在累加和列表中查找第一个大于等于 x 的元素
            for i in range(len(sums)):
                if x <= sums[i]:
                    # 计算偏移量并从序列中输出数字
                    offset = x - sums[i-1] - 1
                    print(sequence[offset])
                    break
```

01026:Ciphers

```
    def move(st, t, a):
        for i in range(t):
            st = a[st]
        return st

# 计算周期，即加密多少次后导致的效果是相同的
    def find_cir(a, n):
        ret = []  # 保存每个位置的周期，即循环节
        for i in range(n):
            x = a[i]
            cnt = 1
            while (x != i):
                x = a[x]
                cnt += 1
            ret.append(cnt)
        return ret


    while (1):
        n = int(input())
        if (n == 0):
            break
        a = list(map(int, input().split()))
        for i in range(n):
            a[i] -= 1
        cir = find_cir(a, n)

        while (1):
            st = input().split(' ', 1)
            k = int(st[0])
            if (k == 0):
                break
            st = list(st[1])
            while (len(st) < n):
                st.append(' ')
            ans = [''] * n
            for i in range(n):
                # 取模省略了之前的多次不必要计算
                ans[move(i, k % cir[i], a)] = st[i]
            print(''.join(ans))
        print()
```

01042:Gone Fishing

```
    from heapq import heappush, heappop
    while True:
        n = int(input())
        if n == 0:
            break
        h = int(input()) * 12
        ans = -1 #最大钓鱼数量
        res = [0] * n #每个湖上所花费的时间
        f = list(map(int, input().split()))
        d = list(map(int, input().split()))
        t = [0] + (list(map(int, input().split())) if n > 1 else
[])
        for i in range(n):
            now = 0 #该情况钓鱼数量
            q = []  #优先队列
            lakes = [{'id': j, 'f': f[j], 'd': d[j]} for j in
range(i + 1)]
            for lake in lakes:
                # 使得鱼的数量多的湖排在优先队列的前面（取负实现）。如果鱼
的数量相同，那么ID小的湖会排在前面。
                heappush(q, (-lake['f'], lake['id']))
            tmp = [0] * n #该情况每个湖钓鱼时间
            time_left = h - sum(t[:i + 1]) #湖上剩余的时间
            while time_left > 0:
                fish_count, idx = heappop(q)
                fish_count = -fish_count #变回正数
                tmp[idx] += 1
                now += fish_count
                lakes[idx]['f'] -= lakes[idx]['d']
                if lakes[idx]['f'] < 0:
                    lakes[idx]['f'] = 0
                heappush(q, (-lakes[idx]['f'], idx))
                time_left -= 1
            if now > ans:
                ans = now
                res = tmp.copy()
        print(", ".join(str(val * 5) for val in res))
        print("Number of fish expected:", ans)
        print()
```

01047: Round and Round We Go

```
    def generate_rotations(s):
        return {s[i:] + s[:i] for i in range(len(s))}


    while True:
        try:
            n = input()
            length, num = len(n), int(n)
            rotations = generate_rotations(n)

            flag = True
            for i in range(2, length + 1):
                n_ = str(num * i)
                if n_ not in rotations and "0" + n_ not in
rotations:
                    flag = False
                    break

            if flag:
                print(f"{n} is cyclic")
            else:
                print(f"{n} is not cyclic")

        except EOFError:
            break
```

01062: 昂贵的聘礼

```
    def dfs(num, max_level, min_level):
        if item_list[num][3]:
            return -1
        if item_list[num][1] < max_level - m or
item_list[num][1] > min_level + m:
            return -1
        item_list[num][3] = True
        max_level_updated = max(max_level, item_list[num][1])
        min_level_updated = min(min_level, item_list[num][1])
        price = item_list[num][0]
        for replace_item in item_list[num][2]:
            pr = dfs(replace_item[0], max_level_updated,
min_level_updated)
            if pr == -1:
                continue
            else:
                price = min(price, replace_item[1] + pr)
        item_list[num][3] = False
        return price
```

```
    m, n = map(int, input().split())
    item_list = []
    for i in range(n):
        p, l, x = map(int, input().split())
        replace_options = []
        for j in range(x):
            t, v = map(int, input().split())
            replace_options.append([t - l, v])
        item_list.append([p, l, replace_options, False])
    print(dfs(0, item_list[0][1], item_list[0][1]))
```

01065: Wooden Sticks

```
    def min_setup_time(sticks):
        n = len(sticks)
        check = [0]*n
        setup_time = 0
        while (0 in check):
            #print(check)
            #print(sticks)
            i = 0
            for j in range(n):
                if check[j] == 0:
                    i = j
                    break
            current = sticks[i]
            check[i] = 1
            setup_time += 1
            i += 1
            while  i < n:
                if  check[i]==0 and current[0]<=sticks[i][0] and
current[1]<= sticks[i][1]:
                    check[i] = 1
                    current = sticks[i]

                i +=1

        return setup_time


    T = int(input())
    for _ in range(T):
        n = int(input())
        data = list(map(int, input().split()))
        sticks = [(data[i], data[i + 1]) for i in range(0, 2 * n,
2)]
        sticks.sort()
        print(min_setup_time(sticks))
```

01067: 取石子游戏

```
    import math

    def wythoff(a, b):
        if a > b:
            a, b = b, a  # Make sure a <= b.
        k = b - a
        ak = k * (math.sqrt(5) + 1) / 2  # ak is the k-th element
in the Beatty sequence.
        return 1 if a != int(ak) else 0

    while True:
        try:
            a, b = map(int, input().split())
        except:
            break


        ans = wythoff(a, b)

        print(ans)
```

01160: Post Office

```
    v, p = map(int, input().split())
    x = [0] + list(map(int, input().split()))
    dis = [[0] * (v + 1) for _ in range(v + 1)]
    dp = [[0] * (v + 1) for _ in range(v + 1)]
    for i in range(1, v + 1):
        for j in range(i + 1, v + 1):
            dis[i][j] = dis[i][j - 1] + x[j] - x[(i + j) // 2]
    for i in range(1, v + 1):
        dp[i][1] = 0
        dp[i][1] = dis[1][i]
    for j in range(2, p + 1):
        for i in range(j + 1, v + 1):
            dp[i][j] = float("inf")
            for k in range(j - 1, i):
                dp[i][j] = min(dp[i][j], dp[k][j - 1] + dis[k +
1][i])
    print(dp[v][p])
```

01191:棋盘分割

```
    from collections import defaultdict

    def f(n, x1, y1, x2, y2):
        if dp[(n, x1, y1, x2, y2)] > 0:
            return dp[(n, x1, y1, x2, y2)]
        if n == 1:
            su = 0
            for i in range(x1, x2+1):
                for j in range(y1, y2+1):
                    su += l[i][j]
            dp[(n, x1, y1, x2, y2)] = su*su
            return su*su
        #mi = 10000000
        mi = float('inf')
        for i in range(x1, x2):
            mi = min(mi, f(n-1, x1, y1, i, y2)+f(1, i+1, y1, x2,
y2))
            mi = min(mi, f(1, x1, y1, i, y2)+f(n-1, i+1, y1, x2,
y2))
        for i in range(y1, y2):
            mi = min(mi, f(n-1, x1, y1, x2, i)+f(1, x1, i+1, x2,
y2))
            mi = min(mi, f(1, x1, y1, x2, i)+f(n-1, x1, i+1, x2,
y2))
        dp[(n, x1, y1, x2, y2)] = mi
        return mi


    n = int(input())
    l = []
    for i in range(8):
        l.append([int(x) for x in input().split()])
    s = 0
    for i in l:
        for j in i:
            s += j
    dp = defaultdict(int)

    print("%.3f"%(f(n, 0,0,7,7)/n-s*s/n/n)**0.5)
```

01321:棋盘问题

```
    # https://www.cnblogs.com/Ayanowww/p/11555193.html
    n, k, ans = 0, 0, 0
    chess = [['' for _ in range(10)] for _ in range(10)]
    take = [False] * 10

    def dfs(h, t):
        global ans

        if t == k:
            ans += 1
            return

        if h == n:
            return

        for i in range(h, n):
            for j in range(n):
                if chess[i][j] == '#' and not take[j]:
                    take[j] = True
                    dfs(i + 1, t + 1)
                    take[j] = False

    while True:
        n, k = map(int, input().split())
        if n == -1 and k == -1:
            break
```

```
        for i in range(n):
            chess[i] = list(input())

        take = [False] * 10
        ans = 0
        dfs(0, 0)
        print(ans)
```

01328:Radar Installation

```
    import math

    def solve(n, d, islands):
        if d < 0:
            return -1

        ranges = []
        for x, y in islands:
            if y > d:
                return -1
            delta = math.sqrt(d * d - y * y)
            ranges.append((x - delta, x + delta))

        if not ranges:
            return -1

        ranges.sort(key=lambda x:x[1])

        number = 1
        r = ranges[0][1]
        for start, end in ranges[1:]:
            if r < start:
                r = end
                number += 1

        return number

    case_number = 0
    while True:
        n, d = map(int, input().split())
        if n == 0 and d == 0:
            break

        case_number += 1
        islands = []
        for _ in range(n):
            islands.append(tuple(map(int, input().split())))

        result = solve(n, d, islands)
        print(f"Case {case_number}: {result}")
        input()
```

01384:Piggy-Bank

```
    INF = float("inf")
    TC = int(input())
    for _ in range(TC):
        E, F = map(int, input().split())
        N = int(input())
        coins = []
        for _ in range(N):
            p, w = map(int, input().split())
            coins.append((p, w))

        amount = F - E
        dp = [0] + [INF]*amount

        for i in range(N):
            p, w = coins[i]
            for j in range(w, amount+1):
                if dp[j-w] != INF:
                    dp[j] = min(dp[j], dp[j-w] + p)

        #print(dp)
        if dp[-1] != INF:
            print(f"The minimum amount of money in the piggy-bank
is {dp[-1]}.")
        else:
            print(f"This is impossible.")
```

01664:放苹果

```
    def apple_distribution(t, cases):
        # 最大苹果数和盘子数
        max_m = max(c[0] for c in cases)
        max_n = max(c[1] for c in cases)

        # 初始化DP数组
        dp = [[0] * (max_n + 1) for _ in range(max_m + 1)]

        # 基础情况
        for m in range(max_m + 1):
            dp[m][1] = 1  # 只有一个盘子
        for n in range(max_n + 1):
            dp[0][n] = 1  # 没有苹果

        # 填表
        for m in range(1, max_m + 1):
            for n in range(2, max_n + 1):
                if n > m:
                    dp[m][n] = dp[m][m]  # 盘子多于苹果
                else:
                    dp[m][n] = dp[m][n-1] + dp[m-n][n]

        # 处理每个测试用例
        results = []
        for m, n in cases:
            results.append(dp[m][n])

        return results

    # 主函数
    def main():
        t = int(input())  # 测试数据数目
        cases = []
        for _ in range(t):
            m, n = map(int, input().split())
            cases.append((m, n))
        results = apple_distribution(t, cases)
        for res in results:
            print(res)

    # 样例测试
    if __name__ == "__main__":
        main()
```

01737: A Decorative fence

```
    UP = 0
    DOWN = 1
    MAXN = 25

    arr = lambda m,n,l : [ [ [0 for k in range(l)] for j in
range(n)] for i in range(m) ]
    #m = arr(2,3,4)

    # C[i][k][DOWN] 是S(i)中以第k短的木棒打头的DOWN方案数,C[i][k][UP]
是S(i)中以第k短的木棒打头的UP方案数,第k短指i根中第k短
    C = arr(MAXN, MAXN, 2)

    def Init(n: int):
        C[1][1][UP] = C[1][1][DOWN] = 1
        for i in range(2, n+1):
            for k in range(1, i+1):          # 枚举第一根木棒的长度
                for M in range(k, i):        # 枚举第二根木棒的长度
                    C[i][k][UP] += C[i-1][M][DOWN]
                for N in range(1, k):        # 枚举第二根木棒的长度
                    C[i][k][DOWN] += C[i-1][N][UP]

    # 总方案数是 Sum( C[n][k][DOWN] + C[n][k][UP] ) k = 1.. n;

    def Print(n: int, cc: int):
        skipped = 0          #已经跳过的方案数
        seq = [0]*MAXN       #最终要输出的答案
        used = [False]*MAXN      #木棒是否用过

        for i in range(1, n+1):      # 依次确定每一个位置i的木棒序号
            oldVal = skipped
```

```
            k = 0
            No = 0      # k是剩下的木棒里的第No短的,No从1开始算
            for k in range(1, n+1):      # 枚举位置i的木棒 , 其长度为k
                oldVal = skipped
                if used[k]==False:
                    No += 1      # k是剩下的木棒里的第No短的
                    if i == 1:
                        skipped += C[n][No][UP] + C[n][No][DOWN]
                    else:
                        if k > seq[i-1] and ( i <=2 or seq[i-
2]>seq[i-1]): #合法放置
                            skipped += C[n-i+1][No][DOWN]
                        elif k < seq[i-1] and (i<=2 or seq[i-
2]<seq[i-1]): #合法放置
                            skipped += C[n-i+1][No][UP]

                    if skipped >= cc:
                        break


            used[k] = True
            seq[i] = k
            skipped = oldVal

        print(' '.join(map(str, seq[1:n+1])))
        '''
        for i in range(1, n+1):
            print("{}".format(seq[i]), end=' ')
        print()
        '''

    Init(20);
    for _ in range(int(input())):
        n, c = map(int, input().split())

        Print(n,c)
```

01742: Coins

```
    import math

    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(sum_2(result) - 1)
        print(bin(result).count('1') - 1)
```

01833: 排列

```
    from typing import List


    def nextPermutation(nums: List[int]) -> None:
        i = len(nums) - 2
        while i >= 0 and nums[i] >= nums[i + 1]:
            i -= 1
        if i >= 0:
            j = len(nums) - 1
            while j >= 0 and nums[i] >= nums[j]:
                j -= 1

            nums[i], nums[j] = nums[j], nums[i]

        left, right = i + 1, len(nums) - 1
        while left < right:
            nums[left], nums[right] = nums[right], nums[left]
            left += 1
            right -= 1


    m = int(input())
    for _ in range(m):
        n, k = map(int, input().split())
        a = list(map(int, input().split()))

        for _ in range(k):
            nextPermutation(a)

        print(*a)
```

01958: Strange Tower of Hanoi

```
    def hanoi_four_towers(n, source, target, auxiliary1,
auxiliary2):
        if n == 0:
            return 0
        if n == 1:
            return 1
        min_moves = float('inf')
        for k in range(1, n):
            three_tower_moves = 2**(n-k)-1
            moves = hanoi_four_towers(k, source, auxiliary1,
auxiliary2, target) +\
                    three_tower_moves +\
                    hanoi_four_towers(k, auxiliary1, target, source,
auxiliary2)
            min_moves = min(min_moves, moves)
        return min_moves


    for n in range(1, 13):
        print(hanoi_four_towers(n, 'A','D','B','C'))
```

01961:前缀中的周期

```
    case = 0
    while n := int(input()):
        case += 1
        print(f"Test case #{case}")
        s = input()
        ans = {}
        for i in range(1, n):
            k = 1
            while (k+1)*i <= n and s[:i] == s[k*i:(k+1)*i]:
                k += 1
                if i*k not in ans:
                    ans[i*k] = k
        for r in ans.items():
            print(*r)
        print()
```

02181: Jumping Cows

```
    P = int(input())
    potions = []
    for i in range(P):
        potions.append((int(input())))
    result = 0
    sign = 1
    for i in range(P-1):
        if (potions[i + 1] - potions[i]) * sign < 0:
            result += sign * potions[i]
            sign = -sign
    if sign == 1:
        result += potions[P-1]
    print(result)
```

02229: Sumsets

```
    import math

    N = int(input())
    ln = int(math.log2(N))
    dp = [1] + [0]*N
    MOD = 10**9
```

```
    items = []
    for i in range(ln+1):
        items.append(pow(2, i))

    for i in items:
        for j in range(i, N+1):
            dp[j] = (dp[j] + dp[j-i]) % MOD

    print(dp[-1])
```

02385: Apple Catching

```
    T, W = map(int, input().split())
    trees = [0] + [int(input()) for _ in range(T)]
    dp = [[0]*(W+1) for _ in range(T+1)]
    for i in range(1, T + 1):
        for j in range(min(i, W) + 1):
            if j % 2 + 1 == trees[i]: # 在树下
                dp[i][j] = max(dp[i][j], dp[i-1][j] + 1) #本来就在
                if j > 0:
                    dp[i][j] = max(dp[i][j], dp[i-1][j-1] + 1) #在
上一分钟结束时刻移动到这里
            else:
                dp[i][j] = dp[i-1][j] # 不在树下
                if j > 0:
                    dp[i][j] = max(dp[i][j], dp[i-1][j-1]) #在上一
分钟结束时刻离开这里
    print(max(dp[T]))
```

02386: Lake Counting

```
    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))

    # 读取输入
    n, m = map(int, input().split())
    field = [list(input()) for _ in range(n)]

    # 初始化8个方向
    directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1),
(1, -1), (1, 0), (1, 1)]

    # 计数器
    cnt = 0

    # 遍历地图
    for i in range(n):
        for j in range(m):
            if field[i][j] == 'W':
                dfs(i, j)
                cnt += 1

    print(cnt)
```

02431: Expedition

```
    import heapq

    N = int(input())
    rawL = [list(map(int, input().split())) for _ in range(N)]
    rawL.append([0, 0])
    rawL.sort()

    L, P = map(int, input().split())

    N += 1

    que = []
    ans = 0
    pos = L
    tank = P

    for i in range(N - 1, -1, -1):
        d = pos - rawL[i][0]  # 接下去要前进的距离

        while tank - d < 0:  # 不断加油直到油量足够行驶到下一个加油站
            if not que:
                print(-1)
                exit()

            tank += -heapq.heappop(que)
            ans += 1

        tank -= d
        pos = rawL[i][0]
        heapq.heappush(que, -rawL[i][1])

    print(ans)
```

02456: Aggressive cows

```
    def can_reach(distance):
        count = 1
        cur = stall[0]
        for i in range(1, n):
            if stall[i] - cur >= distance:
                count += 1
                cur = stall[i]
        return count >= c
    #二分查找最大的能达到的距离


    def binary_search():
        low = 0
        high = (stall[-1] - stall[0])//(c-1)
        while low <= high:
            mid = (low + high) // 2
            if can_reach(mid):
                low = mid + 1
            else:
                high = mid - 1
        return high


    n, c = map(int, input().split())
    stall = sorted([int(input()) for _ in range(n)])
    print(binary_search())
```

02533: Longest Ordered Subsequence

```
    n = int(input())
    *b, = map(int, input().split())
    dp = [1]*n

    for i in range(1, n):
        for j in range(i):
            if b[j] < b[i]:
                dp[i] = max(dp[i], dp[j]+1)

    print(max(dp))
```

02659: Bomb Game

```
    def max_count(matrix):
        maximum = max(max(row) for row in matrix)
        count = sum(row.count(maximum) for row in matrix)
        return count

    def calculate_possible_positions(A, B, K, bombs):
        positions = [[0] * B for _ in range(A)]

        for (R, S, P, T) in bombs:
            for i in range(max(0, R - (P - 1) // 2), min(A, R +
(P + 1) // 2)):
                for j in range(max(0, S - (P - 1) // 2), min(B, S
+ (P + 1) // 2)):
                    if T == 1 :
                        positions[i][j] += 1

                    elif T == 0:
                        positions[i][j] -= 1
```

```
        #for row in positions:
        #    print(row)
        return max_count(positions)

A, B, K = map(int, input().split())
bombs = []
for _ in range(K):
    R, S, P, T = map(int, input().split())
    bombs.append((R - 1, S - 1, P, T))

result = calculate_possible_positions(A, B, K, bombs)
print(result)
```

02692：假币问题

```
n = int(input())

def check(coins, case):
    for item in case:
        left, right, res = item.split()

        left_total = sum(coins[i] for i in left)
        right_total = sum(coins[i] for i in right)

        if left_total == right_total and res != 'even':
            return False
        elif left_total < right_total and res != 'down':
            return False
        elif left_total > right_total and res != 'up':
            return False

    return True

for _ in range(n):
    case = [input().strip() for _ in range(3)]

    for counterfeit in 'ABCDEFGHIJKL':
        found = False
        for weight in [-1, 1]:
            coins = {coin: 0 for coin in 'ABCDEFGHIJKL'}
            coins[counterfeit] = weight

            if check(coins, case):
                found = True
                tag = "light" if weight == -1 else "heavy"
                print(f'{counterfeit} is the counterfeit coin
and it is {tag}.')
                break
        if found:
            break
```

02706：麦森数

```
from math import log10, ceil
M = 10**500
MAXP = 3100000

p = int(input())
print(int(p*log10(2)) + 1)
#print(ceil(p*0.30102999566398812))

a = [2]
for _ in range(len(bin(MAXP)) - 2):
    a.append((a[-1]**2) % M)

s = 1
i = 0
for j in reversed(bin(p)[2:]):
    if j == '1':
        s = (s*a[i]) % M
    i += 1

ans = list(str(s-1))
l = len(ans)
if l < 500:
    ans = ['0']*(500-l)+ans
for i in range(0, 500, 50):
    print(''.join(ans[i:i+50]))
```

02745:显示器

```
while True:
    s,n = input().split()
    if [s,n] == ['0']:
        break
    else:
        s = int(s)
        a = ' '*(s+2)
        b = ' '+'-'*s+' '
        c = '|'+' '*(s+1)
        d = ' '*(s+1)+'|'
        e = '|'+' '*s+'|'
        dic =
{'1':[a,d,a,d,a],'2':[b,d,b,c,b],'3':[b,d,b,d,b],'4':[a,e,b,d,a],
'5':[b,c,b,d,b],

'6':[b,c,b,e,b],'7':[b,d,a,d,a],'8':[b,e,b,e,b],'9':[b,e,b,d,b],'
0':[b,e,a,e,b]}
        lis_1,lis_2,lis_3,lis_4,lis_5 = [],[],[],[],[]
        for i in range(len(n)):
            lis = dic[n[i]]
            lis_1.append(lis[0])
            lis_2.append(lis[1])
            lis_3.append(lis[2])
            lis_4.append(lis[3])
            lis_5.append(lis[4])
        lis_0 = [' '.join(lis_1),' '.join(lis_2),'
'.join(lis_3),' '.join(lis_4),' '.join(lis_5)]
```

```
            for i in range(2*s+3):
                if i == 0:
                    print(lis_0[0])
                elif i < s+1:
                    print(lis_0[1])
                elif i == s+1:
                    print(lis_0[2])
                elif i < 2*s+2:
                    print(lis_0[3])
                else:
                    print(lis_0[4])
            print()
```

02749:分解因数

```
def decompositions(n,minfactor):
    if n==1:
        return 1
    count=0
    for i in range(minfactor,n+1):
        if n%i==0:
            #递归，只找更大的因数，避免重复
            count+=decompositions(n//i,i)
    return count
n=int(input())
for _ in range(n):
    x=int(input())
    print(decompositions(x,2))
```

02766:最大子矩阵

```
def max_submatrix(matrix):
    def kadane(arr):
        # max_ending_here 用于追踪到当前元素为止包含当前元素的最大子
数组和。
        # max_so_far 用于存储迄今为止遇到的最大子数组和。
        max_end_here = max_so_far = arr[0]
        for x in arr[1:]:
            # 对于每个新元素，我们决定是开始一个新的子数组（仅包含当前
元素 x），
            # 还是将当前元素添加到现有的子数组中。这一步是 Kadane 算
法的核心。
            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)
```

02787:算24

```
def find(nums):
    if len(nums) == 1:
        return abs(nums[0] - 24) <= 0.000001

    for i in range(len(nums)):
        for j in range(i+1, len(nums)):
            a = nums[i]
            b = nums[j]
            remaining_nums = []

            for k in range(len(nums)):
                if k != i and k != j:
                    remaining_nums.append(nums[k])

            # 尝试加法和乘法运算
            if find(remaining_nums + [a + b]) or
find(remaining_nums + [a * b]):
                return True

            # 尝试减法运算
            if a > b and find(remaining_nums + [a - b]):
                return True
            if b > a and find(remaining_nums + [b - a]):
                return True

            # 尝试除法运算
            if b != 0 and find(remaining_nums + [a / b]):
                return True
            if a != 0 and find(remaining_nums + [b / a]):
                return True

    return False

while True:
    card = [int(x) for x in input().split()]
    if sum(card) == 0:
        break

    print("YES" if find(card) else "NO")
```

02797:最短前缀

```
words = []
while True:
    try:
        words.append(input())
    except EOFError:
        break
n = len(words)
sorted_words = sorted(words)
buffer = set()
pre_dict = dict()
for i, w in enumerate(sorted_words):
    for j in range(len(w)):
        pre = w[:j+1]
        if (pre in buffer):
            continue
        if i+1 < n and sorted_words[i+1].startswith(pre):
            buffer.add(pre)
        else:
            break
    pre_dict[w] = w[:j+1]
for w in words:
    print(w, pre_dict[w])
```

02802:小游戏

```
from collections import deque
from collections import defaultdict

def bfs(start, end, grid, h, w):
    queue = deque([start])
    in_queue = defaultdict(lambda: float('inf'))
    dirs = [(0, -1), (-1, 0), (0, 1), (1, 0)]
    min_x = float('inf')
    while queue:
        x, y, d, seg = queue.popleft()

        for i, (dx, dy) in enumerate(dirs):
            nx, ny = x + dx, y + dy

            new_seg = seg if i == d else seg + 1
            if (nx, ny) == end:
                min_x = min(min_x, new_seg)
                continue

            if (0 <= nx < h + 2 and 0 <= ny < w + 2 and
new_seg<in_queue[(nx,ny,i)]
                    and grid[nx][ny] != 'X'):
                in_queue[(nx, ny, i)] = new_seg
                queue.append((nx, ny, i, new_seg))

    return min_x

board_num = 1
while True:
    w, h = map(int, input().split())
    if w == h == 0:
        break

    grid = [' ' * (w + 2)] + [' ' + input() + ' ' for _ in
range(h)] + [' ' * (w + 2)]
    print(f"Board #{board_num}:")
    pair_num = 1
    while True:
        y1, x1, y2, x2 = map(int, input().split())
        if x1 == y1 == x2 == y2 == 0:
            break

        start = (x1, y1, -1, 0)
        end = (x2, y2)

        seg = bfs(start, end, grid, h, w)
        if seg == float('inf'):
            print(f"Pair {pair_num}: impossible.")
        else:
            print(f"Pair {pair_num}: {seg} segments.")
        pair_num += 1

    print()
    board_num += 1
```

02811:熄灯问题

```
from copy import deepcopy
from itertools import product
rmap = {0:1, 1:0}
matrix_backup = [[0] * 8] + [[0, *map(int, input().split()),
0] for i in range(5)] \
    + [[0] * 8]

for test in product(range(2), repeat=6):
    matrix = deepcopy(matrix_backup)
    triggers = [list(test)]
    for i in range(1, 6):
        for j in range(1, 7):
            if triggers[i - 1][j - 1]:
                matrix[i][j] = rmap[matrix[i][j]]
                matrix[i - 1][j] = rmap[matrix[i - 1][j]]
                matrix[i + 1][j] = rmap[matrix[i + 1][j]]
                matrix[i][j - 1] = rmap[matrix[i][j - 1]]
                matrix[i][j + 1] = rmap[matrix[i][j + 1]]
```

```
        triggers.append(matrix[i][1:7])
    if matrix[5][1:7] == [0, 0, 0, 0, 0, 0]:
        for trigger in triggers[:-1]:
            print(' '.join(map(str, trigger)))
```

02812:愉人的育蛙

```
import array
def is_valid(x, y):
    return 0 < x <= R and 0 < y <= C
R, C = map(int, input().split())
N = int(input())
#紧凑数组，省内存
flag = [array.array("B", [0] * (C + 1)) for _ in range(R + 1)]
points = [tuple(map(int, input().split())) for _ in range(N)]
for x, y in points:
    flag[x][y] = 1
#排序，先按行升序，再按列升序
points.sort()
max_count = 2
for i in range(N):
    x1, y1 = points[i]
    for j in range(i + 1, N):
        x2, y2 = points[j]
        dx, dy = x2 - x1, y2 - y1
        # x1,y1只是途径点而非起始点，跳过本次循环
        if is_valid(x1-dx, y1-dy):
            continue
        # 行越界，跳出整个循环
        if not (0 < x1 + dx * (max_count - 1) <= R):
            break
        # 列越界，跳出本次循环
        if not (0< y1 + dy * (max_count - 1) <= C):
            continue
        cnt = 2
        while is_valid(x2 + dx, y2 + dy):
            x2 += dx
            y2 += dy
            if not flag[x2][y2]:
                break
            cnt += 1
        else:
            max_count = max(max_count, cnt)
print(max_count if max_count > 2 else 0)
```

02818:密码

Same as 01026 cipher

02979:陪审团的人选

```
    maxn = 400 + 10     # -20 X m <= maxn <= 20 X m, where 1<=m<=20

    cnt = 0
    while True:
        try:
            n, m = map(int, input().split())
        except ValueError:      #跳过空行
            continue

        if n + m == 0:
            break

        p = [0]     # 控方分数
        d = [0]     # 辩方分数
        for _ in range(n):
            tp, td = map(int, input().split())
            p.append(tp)
            d.append(td)

        cnt += 1

        if n == m:
            prosecution = sum(p)
            defence = sum(d)
            print('Jury #()'.format(cnt))
            print("Best jury has value {} for prosecution and value {} for defence:".format(prosecution, defence))
            print(' '.join(map(str, range(1,n+1))))
            continue

        #f = []     转移方程：f[j][k]=f[j-1][x]+d[i]+p[i]；且k=x+p[i]-d[i], f[j-1][x]是满足条件的最大值
        #path = []  记录j个人评审差为k,这种情况下的前一个人j-1是谁

        result = [None] * maxn
        f = [[-1]*5*maxn for _ in range(m+1)]
        path = [[0]*5*maxn for _ in range(m+1)]


        minP_D = 20 * m   # 避免下标为负，所以推广到零以上.题目中的辩控差为0，对应于程序中的辩控差为20*m
        f[0][minP_D] = 0  # 初始化条件. 选0个人使得辩控差为nMinP_D的方案，其辩控和就是0

        for j in range(m):  # 每次循环选出第j个人，共要选出m人
            for k in range(minP_D*2 + 1):   # 可能的辩控差为[0, nMinP_D*2]
                if f[j][k] >= 0:        # 方案 f(j,k)可行
```

```
                for i in range(1, n+1):     # 在方案f(j,k)的基础上，挑下一个人，逐个试
                    if (f[j][k] + p[i] + d[i]) > (f[j+1][k+p[i]-d[i]]):
                        t1 = j
                        t2 = k
                        # 第i个人是否已经在方案f(j,k)中被选上了
                        while (t1 > 0) and (path[t1][t2] != i):
                            t2 = t2 - p[path[t1][t2]] + d[path[t1][t2]]
                            t1 -= 1

                        # 说明第i个人在方案f(j,k)中没有被选上，那么就选它
                        # 形成方案f(j+1,k+p[i]-d[i])
                        if t1 == 0:
                            # 记录新方案的辩控和
                            f[j+1][k+p[i]-d[i]] = f[j][k] + p[i] + d[i]
                            # 选了第i 个人，就要将其记录在path里
                            path[j+1][k+p[i]-d[i]] = i

        #i = minP_D
        j = k = 0
        # determine minimum possible |D(J)-P(J)|
        # 找选了m个人，且实际辩控差绝对值最小的方案。最终方案的程序中辩控差为k
        while (f[m][minP_D+j] < 0) and (f[m][minP_D-j] < 0):
            j += 1

        if f[m][minP_D+j] > f[m][minP_D-j]:                     # 计算k
            k = minP_D + j
        else:
            k = minP_D - j

        # sum(pi) + sum(di) = f(j,k)            (1)
        # sum(pi) - sum(di) = k - minP_D        (2)
        # sum(pi) = ((1) + (2)) // 2
        # sum(di) = ((1) - (2)) // 2
        prosecution = (k - minP_D + f[m][k]) // 2
        defence = (minP_D - k + f[m][k]) // 2

        print('Jury #{}'.format(cnt))
        print("Best jury has value {} for prosecution and value {} for defence:".format(prosecution, defence))

        # 最终方案f(m, k)的最后一个人选记录在Path[m][k]中，
        # 那么从Path[m][k]出发，顺藤摸瓜就能找出方案f(m, k)的所有人选
        for i in range(1, m+1):
```

```
        result[i] = path[m - i + 1][k]
        a = result[i]
        b = p[a] - d[a]
        k = k - b

    ans = []
    for i in range(1, m+1):
        if result[i] != None:
            ans.append(result[i])

    ans = sorted(ans)   # 按人选编号从小到大排序，以便按要求输出
    print(' '.join(map(str, ans)))
```

02945: 拦截导弹

```
def max_intercepted_missiles(k, heights):
    # Initialize the dp array
    dp = [1] * k

    # Fill the dp array
    for i in range(1, k):
        for j in range(i):
            if heights[i] <= heights[j]:
                dp[i] = max(dp[i], dp[j] + 1)

    # The result is the maximum value in dp array
    return max(dp)


if __name__ == "__main__":

    k = int(input())
    heights = list(map(int, input().split()))

    result = max_intercepted_missiles(k, heights)
    print(result)
```

02996:选课

```
from typing import List

def nextPermutation(nums: List[int]) -> None:
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1
    if i >= 0:
        j = len(nums) - 1
        while j >= 0 and nums[i] >= nums[j]:
            j -= 1

        nums[i], nums[j] = nums[j], nums[i]

    left, right = i + 1, len(nums) - 1
```

```
    while left < right:
        nums[left], nums[right] = nums[right], nums[left]
        left += 1
        right -= 1


n = int(input())
m = int(input())
arr = list(map(int, input().split()))
for k in range(m):
    nextPermutation(arr)
print(*arr)
```

03151: Pots

```
from collections import deque

a, b, c = map(int, input().split())

def bfs():
    queue = deque()
    queue.append((0, 0, []))
    visited = set()
    visited.add((0, 0))

    while queue:
        v1, v2, path = queue.popleft()
        if v1 == c or v2 == c:
            return path

        # FILL(1)
        if v1 < a and (a, v2) not in visited:
            queue.append((a, v2, path + ['FILL(1)']))
            visited.add((a, v2))

        # FILL(2)
        if v2 < b and (v1, b) not in visited:
            queue.append((v1, b, path + ['FILL(2)']))
            visited.add((v1, b))

        # DROP(1)
        if v1 > 0 and (0, v2) not in visited:
            queue.append((0, v2, path + ['DROP(1)']))
            visited.add((0, v2))

        # DROP(2)
        if v2 > 0 and (v1, 0) not in visited:
            queue.append((v1, 0, path + ['DROP(2)']))
            visited.add((v1, 0))

        # POUR(1,2)
        if v1 > 0 and v2 < b:
            pour_to_2 = min(v1, b - v2)
            new_v1 = v1 - pour_to_2
```

```
            new_v2 = v2 + pour_to_2
            if (new_v1, new_v2) not in visited:
                queue.append((new_v1, new_v2, path + ['POUR(1,2)']))
                visited.add((new_v1, new_v2))

        # POUR(2,1)
        if v2 > 0 and v1 < a:
            pour_to_1 = min(v2, a - v1)
            new_v1 = v1 + pour_to_1
            new_v2 = v2 - pour_to_1
            if (new_v1, new_v2) not in visited:
                queue.append((new_v1, new_v2, path + ['POUR(2,1)']))
                visited.add((new_v1, new_v2))

    return None

ans = bfs()
if ans:
    print(len(ans))
    for step in ans:
        print(step)
else:
    print('impossible')
```

03376: Best Cow Line, Gold

```
# Time Limit Exceeded
n = int(input())
S = []
for _ in range(n):
    S.append(input())

a = 0
b = len(S) - 1
ans = []
cnt = 0
while a <= b:
    bLeft = False
    for i in range(b-a+1):
        if S[a+i] < S[b-i]:
            bLeft = True
            break
        elif S[a+i] > S[b-i]:
            bLeft = False
            break

    cnt += 1
    if bLeft:
        #print(S[a], end='')
        ans.append(S[a])
        a += 1
    else:
```

```
            #print(S[b], end='')
            ans.append(S[b])
            b -= 1
        if cnt%80 == 0:
            ans.append('\n')

    print(''.join(ans))
```

03406:书架

```
    n, B = map(int, input().split())
    h = []
    for _ in range(n):
        h.append(int(input()))

    h.sort(reverse=True)
    i = 0
    s = 0
    while i <= n:
        s += h[i]
        i += 1
        if s >= B:
            break

    print(i)
```

03441: 4 Values whose sum is 0

```
    n = int(input())
    a = [0]*(n+1)
    b = [0]*(n+1)
    c = [0]*(n+1)
    d = [0]*(n+1)

    for i in range(n):
        a[i],b[i],c[i],d[i] = map(int, input().split())

    dictl = {}
    for i in range(n):
        for j in range(n):
            if not a[i]+b[j] in dictl:
                dictl[a[i] + b[j]] = 0
            dictl[a[i] + b[j]] += 1

    ans = 0
    for i in range(n):
        for j in range(n):
            if -(c[i]+d[j]) in dictl:
                ans += dictl[-(c[i]+d[j])]

    print(ans)
```

03468:电池的寿命

```
    while True:
        try:
            n=int(input())
            l=list(map(int,input().split()))
            l.sort()
            maxi=l.pop()
            sumi=sum(l)
            if sumi<maxi:
                num=sumi
            else:
                num=(sumi+maxi)/2
            print(f"{num:.1f}")
        except:
            break
```

04005:拼点游戏

```
def get_max_profit(a1, a2):
    la1 = 0
    ra1 = len(a1) - 1
    la2 = 0
    ra2 = len(a2) - 1
    ans_max = 0
    ans_min = 0

    while la2 <= ra2:
        if a2[la2] > a1[la1]:
            ans_max += 3
            ans_min += 1
            la1 += 1
            la2 += 1
        elif a2[ra2] > a1[ra1]:
            ans_max += 3
            ans_min += 1
            ra1 -= 1
            ra2 -= 1
        else:
            if a2[la2] < a1[ra1]:
                ans_max += 1
                ans_min += 3
            elif a2[la2] == a1[ra1]:
                ans_max += 2
                ans_min += 2

            la2 += 1
            ra1 -= 1

    return ans_max, ans_min


while True:
    n = int(input())
    if n == 0:
        break
        *C, = map(int, input().split())
        *S, = map(int, input().split())
        C.sort()
        S.sort()

        ans_max, _ = get_max_profit(C, S)
        _, ans_min = get_max_profit(S, C)

        print(ans_max, ans_min)
```

04036:计算系数

```
    import math
    a, b, k, n, m = map(int, input().split());
    print((pow(a, n, 10007) * pow(b, m, 10007) * math.comb(k,
m)) % 10007)
```

04087:数据筛选

```
    import array                                    # Memory
Limit Exceed
    import heapq

    n, k = map(int, input().split())

    q = []
    cnt = 0

    while True:
        x = array.array('i', map(int, input().split()))
        cnt += len(x)
        for i in x:
            heapq.heappush(q, -i)
            if len(q) > k:
                heapq.heappop(q)
        if cnt >= n:
            break

    print(-q[0])
```

04100:进程检测

```
import sys
#from itertools import islice
from collections import namedtuple

# 使用 namedtuple 简化类定义
Pro = namedtuple('Pro', ['s', 'd'])


def main():
    input = sys.stdin.read
    data = input().split()

    k = int(data[0])
    results = []

    index = 1
    for _ in range(k):
        n = int(data[index])
        index += 1

        ex = [Pro(int(data[index + 2 * i]), int(data[index + 2 *
i + 1])) for i in range(n)]
        index += 2 * n

        ex.sort(key=lambda x: x.s)

        m = ex[-1].s
        t = 1

        for i in range(n - 2, -1, -1):
            if ex[i].d >= m:
                continue
            else:
                m = ex[i].s
                t += 1

        results.append(t)

    for result in results:
        print(result)


if __name__ == "__main__":
    main()
```

04102:宠物精灵之收服

```
N, M, K = map(int, input().split())
dp = [[-1] * (M + 1) for i in range(K + 1)]
dp[0][M] = N
for i in range(K):
    cost, dmg = map(int, input().split())
    for p in range(M):
        for q in range(i + 1, 0, -1):
            if p + dmg <= M and dp[q - 1][p + dmg] != -1:
                dp[q][p] = max(dp[q][p], dp[q - 1][p + dmg] -
cost)


def find():
    for i in range(K, -1, -1):
        for j in range(M, -1, -1):
            if dp[i][j] != -1:
                return i, j


captured, remaining_life = find()
print(captured, remaining_life)
```

04103: 跟方格

```
n = int(input())
step = [[1, 0], [-1, 0], [0, 1]]
num = 1


def dfs(x, y, m, visited):
    global num
    if m == 0:
        return
    visited.append([x, y])
    num -= 1
    for j in range(3):
        if [x+step[j][0], y+step[j][1]] not in visited:
            num += 1
            lista = []
            lista += visited
            dfs(x+step[j][0], y+step[j][1], m-1, lista)


dfs(0, 0, n, [])
print(num)
```

04115: 鸣人和佐助

```
from collections import deque

M, N, T = map(int, input().split())
graph = [list(input()) for i in range(M)]
direc = [(0,1), (1,0), (-1,0), (0,-1)]
start, end = None, None
for i in range(M):
    for j in range(N):
        if graph[i][j] == '@':
            start = (i, j)
def bfs():
    q = deque([start + (T, 0)])
    visited = [[-1]*N for i in range(M)]
    visited[start[0]][start[1]] = T
    while q:
        x, y, t, time = q.popleft()
        time += 1
        for dx, dy in direc:
            if 0<=x+dx<M and 0<=y+dy<N:
                if (elem := graph[x+dx][y+dy]) == '*' and t >
visited[x+dx][y+dy]:
                    visited[x+dx][y+dy] = t
                    q.append((x+dx, y+dy, t, time))
                elif elem == '#' and t > 0 and t-1 >
visited[x+dx][y+dy]:
                    visited[x+dx][y+dy] = t-1
                    q.append((x+dx, y+dy, t-1, time))
                elif elem == '+':
                    return time
    return -1
print(bfs())
```

04116: 拯救行动

```
from heapq import heappush, heappop

dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]


def bfs(matrix, start):
    n, m = len(matrix), len(matrix[0])
    visited = [[False for _ in range(m)] for _ in range(n)]
    #q = deque([(start[0], start[1], 0)])
    q = []
    heappush(q, (0, start[0], start[1]))
    visited[start[0]][start[1]] = True
    while len(q) != 0:
        #x, y, time = q.popleft()
        time, x, y = heappop(q)
        for i in range(4):
            nx, ny = x + dx[i], y + dy[i]
            if 0 <= nx < n and 0 <= ny < m and not
visited[nx][ny]:
                if matrix[nx][ny] == "a":
                    #ans.append(time+1)
                    return time + 1
                elif matrix[nx][ny] == "@":
                    #q.append((nx, ny, time + 1))
                    heappush(q, (time + 1, nx, ny))
                    visited[nx][ny] = True
                elif matrix[nx][ny] == "x":
                    #q.append((nx, ny, time + 2))
                    heappush(q, (time + 2, nx, ny))
                    visited[nx][ny] = True

    return "Impossible"


S = int(input())
for _ in range(S):
    N, M = map(int, input().split())
    matrix = [list(input()) for _ in range(N)]
    start = None
    ans = []
    for i in range(N):
        for j in range(M):
            if matrix[i][j] == "r":
                start = (i, j)
                break
    print(bfs(matrix, start))
```

```
    # if ans == []:
    #     print("Impossible")
    # else:
    #     print(min(ans))
```

04117：简单的整数划分问题

```
def partition_count(n):
    # 初始化dp数组，dp[i][j] 表示数字i划分成不超过j的数的划分方式数
    dp = [[0] * (n + 1) for _ in range(n + 1)]

    # 数字0没有实质内容的划分，所以设置dp[0][j]为1
    for j in range(n + 1):
        dp[0][j] = 1

    # 填充dp数组
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if i < j:
                dp[i][j] = dp[i][i]
            else:
                dp[i][j] = dp[i][j - 1] + dp[i - j][j]

    # 返回dp[n][n]即为n的划分数
    return dp[n][n]

# 输入处理部分
try:
    while True:
        N = int(input())
        print(partition_count(N))
except EOFError:
    pass
```

04118：开餐饮

```
def max_profit(n, k, loc, prof):
    dp = prof[:]  # Initialize dp with profits at each location

    for i in range(n):
        for j in range(i):
            if loc[i] - loc[j] > k:  # Check if the distance is
greater than k
                dp[i] = max(dp[i], dp[j] + prof[i])

    return max(dp)

for _ in range(int(input())):
    n, k = map(int, input().strip().split())
    locations = list(map(int, input().strip().split()))
    profits = list(map(int, input().strip().split()))

    print(max_profit(n, k, locations, profits))
```

04119：复杂的整数划分问题

```
# https://blog.csdn.net/hejnhong/article/details/105211551
def divide_k(n, k):
    # dp[i][j]为将i划分为j个正整数的划分方法数量
    dp = [[0]*(k+1) for _ in range(n+1)]
    for i in range(n+1):
        dp[i][1] = 1
    for i in range(1, n+1):
        for j in range(1, k+1):
            if i >= j:
                # dp[i-1][j-1]为包含1的划分的数量
                # 若不包含1，我们对每个数-1仍为正整数，划分数量为dp[i-
j][j]
                dp[i][j] = dp[i-j][j]+dp[i-1][j-1]
    return dp[n][k]


def divide_dif(n):
    # dp[i][j]表示将数字 i 划分，其中最大的数字不大于 j 的方法数量
    dp = [[0] * (n + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            # 比i大的数没用
            if i < j:
                dp[i][j] = dp[i][i]
            # 多了一种：不划分
            elif i == j:
                dp[i][j] = dp[i][j - 1] + 1
            # 用/不用j
            else:
                dp[i][j] = dp[i][j - 1] + dp[i - j][j - 1]
    return dp[n][n]


# 关于分拆数的一个结论，https://zhuanlan.zhihu.com/p/21440865
# 一个数的奇分拆总是等于互异分拆
def divide_odd(n):
    # dp[i][j]整数i的划分里最大的数是j
    dp = [[0] * (n + 1) for _ in range(n + 1)]
    dp[0][0] = 1
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if j % 2 == 0:
                dp[i][j] = dp[i][j-1]
            else:
                if i < j:
                    dp[i][j] = dp[i][i]
                elif i == j:
                    dp[i][j] = dp[i][j - 1] + 1
                # 用/不用j
                else:
```

```
                dp[i][j] = dp[i][j - 1] + dp[i - j][j]
    return dp[n][n]


while True:
    try:
        n, k = map(int, input().split())
        print(divide_k(n, k))
        print(divide_dif(n))
        print(divide_odd(n))
    except EOFError:
        break
```

04129：变换的迷宫

```
import heapq
from math import inf

# 四个基本方向：右、下、左、上
DIRECTIONS = [(0, 1), (1, 0), (-1, 0), (0, -1)]

def find_shortest_path(grid, start, end, dimensions,
cycle_length):

    rows, cols = dimensions
    visited = [[[False] * cols for _ in range(rows)] for _ in
range(cycle_length)]
    priority_queue = [(0,) + start]  # 初始时间为0加上起点坐标

    while priority_queue:
        time, x, y = heapq.heappop(priority_queue)
        if (x, y) == end:  # 到达终点
            return time

        for dx, dy in DIRECTIONS:
            nx, ny = x + dx, y + dy
            new_time = time + 1

            if not (0 <= nx < rows and 0 <= ny < cols):  # 检查是
否在地图内
                continue

            if grid[nx][ny] == 1 and new_time % cycle_length == 0
and not visited[new_time % cycle_length][nx][ny]:  # 穿石头
                visited[new_time % cycle_length][nx][ny] = True
                heapq.heappush(priority_queue, (new_time, nx,
ny))
            elif grid[nx][ny] == 0 and not visited[new_time %
cycle_length][nx][ny]:  # 普通空地
                visited[new_time % cycle_length][nx][ny] = True
                heapq.heappush(priority_queue, (new_time, nx,
ny))

    return "Oop!"

def main():
    test_cases = int(input())
    results = []

    for _ in range(test_cases):
        rows, cols, cycle_length = map(int, input().split())
        grid = []
        start = None
        end = None

        for i in range(rows):
            line = input()
            row = []
            for j, char in enumerate(line):
                if char == "S":
                    start = (i, j)
                    row.append(0)  # 起点当空地
                elif char == "E":
                    end = (i, j)
                    row.append(0)  # 终点当空地
                elif char == "#":
                    row.append(1)  # 石头
                else:
                    row.append(0)  # 空地
            grid.append(row)

        result = find_shortest_path(grid, start, end, (rows,
cols), cycle_length)
        results.append(result)

    for result in results:
        print(result)

if __name__ == "__main__":
    main()
```

04133：垃圾炸弹

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

04136：矩形分割

```
R = int(input())
a = [0] * R
n = int(input())
for i in range(n):
    L, T, W, H = map(int, input().split())
    for j in range(L, L + W):
        a[j] += H

le = 0
ri = R
while True:
    if le >= ri:
        break
    else:
        mi = (le + ri) // 2
        x = sum(a[:mi])
        y = sum(a[mi:])
        if x >= y:
            ri = mi
        else:
            le = mi + 1
while True:
    if le == R:
        print(le)
        break
    elif a[le] == 0: #如果当前位置的小矩形高度为0，则将位置向右移动。
        le += 1
    else:
        print(le)
        break
```

04137：最小新整数

```
def removeKDigits(num, k):
    stack = []
    for digit in num:
        while k and stack and stack[-1] > digit:
            stack.pop()
            k -= 1
        stack.append(digit)
    # 如果还未删除k位，从尾部继续删除
    while k:
        stack.pop()
        k -= 1
    return int(''.join(stack))
t = int(input())
results = []
for _ in range(t):
    n, k = input().split()
    results.append(removeKDigits(n, int(k)))
for result in results:
    print(result)
```

04144：畜栏保留问题

```
import heapq
max_num = 1
n = int(input())
cows = []
number = [0]*n  # 记录每一只牛所在的畜栏
for i in range(n):
    cows.append(list(map(int, input().split())))
for i in range(n):
    cows[i].append(i)  # 为每只牛添加编号后再排序

cows.sort(key=lambda x: x[0]) # 先按开始时间排序
column = []  # 创建列表【畜栏】
heapq.heappush(column, [cows[0][1], 0]) # 初始时只有一个元素，即为第一
只牛的结束时间
number[cows[0][2]] = 1  # 第一只牛默认在第一个畜栏

for i in range(1, len(cows)):  # 对之后的每只牛遍历
    k = heapq.heappop(column)
    if k[0] < cows[i][0]: # 最早结束的已经结束，新的牛可使用该畜栏
        heapq.heappush(column, [cows[i][1], k[1]])
        number[cows[i][2]] = k[1]+1
    else:
        heapq.heappush(column, k)
        heapq.heappush(column, [cows[i][1], max_num])
        max_num += 1
        number[cows[i][2]] = max_num

print(len(column))  # 【畜栏】的长度即为畜栏数量
for i in number:
    print(i)
```

04145：放弃考试

```
def can_achieve(target,a,b,k):
    diffs=[a[i]-target*b[i] for i in range(len(a))]
    diffs.sort()
    #放弃k场考试后可以达到target
    return sum(diffs[k:])>=0
def max_avg_score(k,a,b):
    l,r=0,100
    while r-l>1e-5:
    #非整数二分
        m=(l+r)/2
        if can_achieve(m,a,b,k):
            l=m
        else:
            r=m
    return m*100
```

```
while True:
    n,k=map(int,input().split())
    if n==0 and k==0:
        break
    a = list(map(int, input().split()))
    b = list(map(int, input().split()))
    print(f'{max_avg_score(k,a,b):.0f}')
```

05333：Fence Repair

```
import heapq

def minimum_cost(planks):
    heapq.heapify(planks)  # 将木板列表转换为最小堆
    total_cost = 0

    while len(planks) > 1:
        # 取出最短的两块木板
        shortest1 = heapq.heappop(planks)
        shortest2 = heapq.heappop(planks)

        # 计算切割的成本，并将切割后得到的木板长度加入堆
        cost = shortest1 + shortest2
        total_cost += cost
        heapq.heappush(planks, cost)

    return total_cost

# 读取输入
n = int(input())
planks = []
for _ in range(n):
    length = int(input())
    planks.append(length)

# 调用函数计算最小成本
result = minimum_cost(planks)

# 输出结果
print(result)
```

05585：晶矿的个数

```
dire = [[-1,0], [1,0], [0,-1], [0,1]]

def dfs(x, y, c):
    m[x][y] = '#'
    for i in range(len(dire)):
        tx = x + dire[i][0]
        ty = y + dire[i][1]
        if m[tx][ty] == c:
            dfs(tx, ty, c)

for _ in range(int(input())):
    n = int(input())
    m = [[0 for _ in range(n+2)] for _ in range(n+2)]

    for i in range(1, n+1):
        m[i][1:-1] = input()

    r = 0 ; b=0
    for i in range(1, n+1):
        for j in range(1, n+1):
            if m[i][j] == 'r':
                dfs(i, j, 'r')
                r += 1
            if m[i][j] == 'b':
                dfs(i,j,'b')
                b += 1
    print(r, b)
```

07622：求排列的逆序数

```
mininum=0
def mergesort(arr):
    global mininum
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]

        mergesort(left)
        mergesort(right)

        Lptr = Rptr = ptr = 0
        while len(left) > Lptr and len(right) > Rptr:
            if left[Lptr] <= right[Rptr]:
                arr[ptr] = left[Lptr]
                Lptr += 1
            else:
                arr[ptr] = right[Rptr]
                Rptr += 1
                mininum += len(left) - Lptr
            ptr += 1

        while len(left) > Lptr:
            arr[ptr] = left[Lptr]
            ptr += 1
            Lptr += 1
        while len(right) > Rptr:
            arr[ptr] = right[Rptr]
            ptr += 1
            Rptr += 1


n = int(input())
arr = list(map(int, input().split()))
mergesort(arr)
print(mininum)
```

08758:2的幂次方表示

```
def power_of_two_representation(n):
    # 函数用于找到小于或等于n的最大2的幂次
    def find_max_power(n):
        power = 0
        while (1 << power) <= n:
            power += 1
        return power - 1

    # 函数用于将幂次表示为2的幂次方的表示
    def represent_power(power):
        if power == 1:
            return '2'
        elif power == 0:
            return '2(0)'
        else:
            return '2(' + power_of_two_representation(power) +
')'

    # 特殊情况：如果n是0，直接返回空字符串
    if n == 0:
        return ''

    result = ''
    while n > 0:
        max_power = find_max_power(n)
        # 如果结果字符串不为空，添加加号
        if result:
            result += '+'
        # 把最大幂次转换为2的幂次方的表示
        result += represent_power(max_power)
        # 减去已经表示的数，继续寻找余数的表示
        n -= 1 << max_power

    return result

print(power_of_two_representation(int(input())))
```

09267：核电站

```
n, m = map(int, input().split())
DP = [0] * 60
DP[0] = 1 #DP[i]是第i个位置的方案数。

for i in range(1, n + 1):
    if i < m: #达不到连续放置m个的情况
        DP[i] = DP[i - 1] * 2  # 从第1个到第m-1个，方案都可以选择放/不
放
    elif i == m: #第m个要小心了
        DP[i] = DP[i - 1] * 2 - 1
    else:#i>m
        DP[i] = DP[i - 1] * 2 - DP[i - m - 1]
print(DP[n])
```

12029：水淹七军

```
from collections import deque
import sys
input = sys.stdin.read

# 判断坐标是否有效
def is_valid(x, y, m, n):
    return 0 <= x < m and 0 <= y < n

# 广度优先搜索模拟水流
def bfs(start_x, start_y, start_height, m, n, h, water_height):
    dx = [-1, 1, 0, 0]
    dy = [0, 0, -1, 1]
    q = deque([(start_x, start_y, start_height)])
    water_height[start_x][start_y] = start_height

    while q:
        x, y, height = q.popleft()
        for i in range(4):
            nx, ny = x + dx[i], y + dy[i]
            if is_valid(nx, ny, m, n) and h[nx][ny] < height:
                if water_height[nx][ny] < height:
                    water_height[nx][ny] = height
                    q.append((nx, ny, height))

# 主函数
def main():
    data = input().split()  # 快速读取所有输入数据
    idx = 0
    k = int(data[idx])
    idx += 1
    results = []

    for _ in range(k):
        m, n = map(int, data[idx:idx + 2])
        idx += 2
        h = []
        for i in range(m):
            h.append(list(map(int, data[idx:idx + n])))
            idx += n
        water_height = [[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 h[x][y] <= h[i][j]:
                continue
            bfs(x, y, h[x][y], m, n, h, water_height)

        results.append("Yes" if water_height[i][j] > 0 else "No")

    sys.stdout.write("\n".join(results) + "\n")

if __name__ == "__main__":
    main()
```

12559：母本最小整数

```
n = int(input())
nums = input().split()
for i in range(n - 1):
    for j in range(i+1, n):
        #print(i,j)
        if nums[i] + nums[j] < nums[j] + nums[i]:
            nums[i], nums[j] = nums[j], nums[i]

ans = "".join(nums)
nums.reverse()
print(ans + " " + "".join(nums))
```

16530:改卷子v0.2

```
n = int(input())
words = []
for i in range(n):
    words.append(input())

words.sort()
k = n//2
mid, afterMid = words[k-1:k+1]
ok = False
rst = ""
temp = ""
cur = 0
length = len(mid)
while cur < length:
    for i in range(26):
        rst = temp
        rst += chr(i+65)
        if mid <= rst <afterMid:
            ok = True
            break
    if ok:
        break
    temp += mid[cur]
    cur += 1

print(rst)
```

16532：北大杯台球比赛

```
m,n = map(int,input().split())
s = [[-1]*(n+2)]
num = s + [[-1]+ [int(x) for x in input().split()]+[-1] for _ in
range(m)] + s
grade = [[int(x) for x in input().split()] for i in range(m*n)]

ans = 0
for i in range(1, m+1):
    for j in range(1, n+1):
        a = num[i][j]
        for b in [num[i-1][j], num[i+1][j], num[i][j+1],
num[i][j-1]]:
            if b!=-1 and grade[b]==grade[a]:
                ans += 1
                break

gsum = [sum(i) for i in grade]
gsum.sort(reverse = True)
c = [i > gsum[int(m * n *0.4)] for i in gsum]
print(ans, c.count(True))
```

18146：乌鸦坐飞机

```
# 蒋子轩23工学院
n, _ = map(int, input().split())
a = list(map(int, input().split()))
cnt = [0]*4
for i in a:
    cnt[i % 4] += 1
# add为空位数
add = cnt[1]+cnt[3]  # 余1或3的必定会造成一个空位
# 余2的优先放1、2和7、8，或和余1的组合放中间，不会增加空位数
t = cnt[2]-2*n-cnt[1]
if t > 0:  # 若还有余2的没能放好
    # 分类讨论余2的至少造成多少个空位
    add += t//3*2
    if t % 3 == 1:
        add += 2
    if t % 3 == 2:
        add += 4
print('YES' if sum(a)+add <= n*8 else 'NO')
```

18156：寻找离目标数最近的两数之和

```
tar = int(input())  #target
s = [int(x) for x in input().split()]
s.sort()
```

```
ans = 200000
gap = 100000 - 2

h = 0               #head
t = len(s) - 1      #tail
while h < t:
    mid = s[h] + s[t]
    if mid == tar:
        ans = mid
        break

    if abs(mid - tar) < gap:
        gap = abs(mid - tar)
        ans = mid
    if abs(mid - tar) == gap:
        ans = min(ans, mid)

    if mid > tar:
        t -= 1
    elif mid < tar:
        h += 1

print(ans)
```

18160: 最大连通域面积

```
dire = [[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-1],[1,0],[1,1]]

def dfs(x,y):
    global area
    if matrix[x][y] == '.':return
    matrix[x][y] = '.'
    area += 1
    for i in range(len(dire)):
        dfs(x+dire[i][0], y+dire[i][1])


for _ in range(int(input())):
    n,m = map(int,input().split())

    matrix = [['.' for _ in range(m+2)] for _ in range(n+2)]
    for i in range(1,n+1):
        matrix[i][1:-1] = input()

    sur = 0
    for i in range(1, n+1):
        for j in range(1, m+1):
            if matrix[i][j] == 'W':
                area = 0
                dfs(i, j)
                sur = max(sur, area)
    print(sur)
```

18176: 2050年成绩计算

```
from math import sqrt
N = 10005

s = [True] * N
p = 2
while p * p <= N:
    if s[p]:
        for i in range(p * 2, N, p):
            s[i] = False
    p += 1

m, n = [int(i) for i in input().split()]

for i in range(m):
    x = [int(i) for i in input().split()]
    sum = 0
    for num in x:
        root = int(sqrt(num))
        if num > 3 and s[root] and num == root * root:
            sum += num
    sum /= len(x)
    if sum == 0:
        print(0)
    else:
        print('%.2f' % sum)
```

19164: 移动办公

```
t,m=map(int,input().split())
p,n=[0]*(t+1),[0]*(t+1)
for i in range(1,t+1):
    p[i],n[i]=map(int,input().split())
dpp,dpn=[0]*(t+1),[0]*(t+1)
dpp[1],dpn[1]=p[1],n[1]
for i in range(2,t+1):
    dpp[i]=max(dpp[i-1]+p[i],dpn[i-1]+p[i]-m)
    dpn[i]=max(dpn[i-1]+n[i],dpp[i-1]+n[i]-m)
print(max(dpp[t],dpn[t]))
```

19930: 寻宝

```
import heapq
def bfs(x,y):
    d=[[-1,0],[1,0],[0,1],[0,-1]]
    queue=[]
    heapq.heappush(queue,[0,x,y])
    check=set()
    check.add((x,y))
    while queue:
        step,x,y=map(int,heapq.heappop(queue))
        if martix[x][y]==1:
            return step
        for i in range(4):
            dx,dy=x+d[i][0],y+d[i][1]
            if martix[dx][dy]!=2 and (dx,dy) not in check:
                heapq.heappush(queue,[step+1,dx,dy])
                check.add((dx,dy))
    return "NO"

m,n=map(int,input().split())
martix=[[2]*(n+2)]+[[2]+list(map(int,input().split()))+[2] for i
in range(m)]+[[2]*(n+2)]
print(bfs(1,1))
```

19961: 最大点数 [外太空2048]

```
def slide_and_combine(board, reverse=False, vertical=False):
    result = []
    for line in zip(*board) if vertical else board:
        if reverse:
            line = line[::-1]
        new_line = [i for i in line if i != 0]
        i = len(new_line) - 1
        while i>0:
            if new_line[i] == new_line[i - 1]:
                new_line[i - 1] *= 2
                del new_line[i]
            i -= 1
        new_line += [0] * (len(line) - len(new_line))
        result.append(new_line[::-1] if reverse else new_line)
    return list(zip(*result)) if vertical else result
def max_score(board, moves):
    if moves == 0:
        return max(max(row) for row in board)
    best_score = max(max(row) for row in board)
    for vertical, reverse in [(False, False), (False, True),
(True, False), (True, True)]:
        new_board = slide_and_combine(board, reverse, vertical)
        best_score = max(best_score, max_score(new_board, moves -
1))
    return best_score
m, n, moves = map(int, input().split())
board = [list(map(int, input().split())) for _ in range(m)]
print(max_score(board, moves))
```

20027: 字符串问题

```
st = input()
l = len(st)
num = 0
for i in range(l):
    num += (26**i)*(ord(st[l-1-i])-97)
num += int(input())+1
ans = ''
while num:
    ans = chr(num%26+97)+ans
    num //= 26
print('a'*(l-len(ans))+ans)
```

20089: NBA门票

```
n = int(input())
if n % 50 != 0:
    print('Fail')
    exit()
n //= 50
nums = list(map(int, input().split()))
price = [1, 2, 5, 10, 20, 50, 100]
dp = [float('inf')] * (n + 1)
dp[0] = 0
for i in range(7):
#for i in range(6, -1, -1):
    cur_price = price[i]
    cur_num = nums[i]
    k = 1
    while cur_num > 0:  #二进制分组优化，时间缩短了将近两个数量级。
                                                    #相同物品避免重复工作，「二进
制分组」提高效率。
        use_num = min(cur_num, k)
        cur_num -= use_num
        for j in range(n, cur_price * use_num - 1, -1):
            dp[j] = min(dp[j], dp[j - cur_price * use_num] +
use_num)
        k *= 2
if dp[-1] == float('inf'):
    print('Fail')
else:
    print(dp[-1])
```

20106: 走山路

```
import heapq
m, n, p = map(int, input().split())
martix = [list(input().split())for i in range(m)]
dir = [(-1, 0), (1, 0), (0, 1), (0, -1)]
for _ in range(p):
    sx, sy, ex, ey = map(int, input().split())
    if martix[sx][sy] == "#" or martix[ex][ey] == "#":
        print("NO")
        continue
    in_queue, heap, ans = set(), [], []
    heapq.heappush(heap, (0, sx, sy))
    in_queue.add((sx, sy, -1))
    while heap:
        tire, x, y = heapq.heappop(heap)
        if x == ex and y == ey:
            ans.append(tire)
        for i in range(4):
            dx, dy = dir[i]
            x1, y1 = dx+x, dy+y
            if 0 <= x1 < m and 0 <= y1 < n and martix[x1][y1] !=
"#" and (x1, y1, i) not in in_queue:
                tl = tire+abs(int(martix[x][y])-
int(martix[x1][y1]))
                heapq.heappush(heap, (tl, x1, y1))
                in_queue.add((x1, y1, i))
    print(min(ans) if ans else "NO")
```

20123:7-友好数

```
def dfs(n, i):
    global bo
    if len(n) > 0 and int(n) % 7 == 0:
        bo = True
    if bo:
        return
    if i >= l:
        return
    dfs(n, i+1)
    dfs(n+s[i], i+1)


s = input()
l = len(s)
if l >= 7:
    print('YES')
    exit()
bo = False
dfs('', 0)
if bo:
    print('YES')
else:
    print('NO')
```

20138: 求解多元一次方程组

```
n=int(input())
mat=[]
ans=[]
for i in range(n):
    mat.append([float(x) for x in input().split()])
for i in range(n):
    if mat[i][i]==0:
        for j in range(i+1,n):
            if mat[j][i]!=0:
                mat[i],mat[j]=mat[j][:],mat[i][:]
                break
    mat[i]=[x/mat[i][i] for x in mat[i]]
    for j in range(i+1,n):
        k=mat[j][i]/mat[i][i]
        mat[j]=[mat[j][x]-k*mat[i][x] for x in range(n+1)]
for i in range(n-1,-1,-1):
    ans.append(mat[i][-1]/mat[i][i])
    for j in range(i):
        mat[j][-1]-=mat[j][i]*ans[-1]
ans.reverse()
for i in range(n):
    print('x{} = {:.2f}'.format(i+1,ans[i]))
```

20140: 今日化学论文

```
s = input()
stack = []
for i in range(len(s)):
    stack.append(s[i])
    if stack[-1] == "]":
        stack.pop()
        helpstack = []
        while stack[-1] != "[":
            helpstack.append(stack.pop())
        stack.pop()
        numstr = ""
        while helpstack[-1] in "0123456789":
            numstr += str(helpstack.pop())
        helpstack = helpstack*int(numstr)
        while helpstack != []:
            stack.append(helpstack.pop())
print(*stack, sep="")
```

21458: 健身房

```
t,n=map(int,input().split())
dp=[0]+[-1]*(t+1)
for i in range(n):
    k,w=map(int,input().split())
    for j in range(t,k-1,-1):
        if dp[j-k]!=-1:
            dp[j]=max(dp[j-k]+w,dp[j])
print(dp[t])
```

22067: 快速堆猪

```
class MinStack:
    def __init__(self):
        self.stack = []          # 主栈
        self.min_stack = []      # 辅助栈，用来保存每个状态下的最小值

    def push(self, x):
        self.stack.append(x)
        if not self.min_stack or x <= self.min_stack[-1]:
            self.min_stack.append(x)

    def pop(self):
        if self.stack:  # 如果主栈非空才执行pop
            top = self.stack.pop()
            if top == self.min_stack[-1]:
                self.min_stack.pop()

    def min(self):
        if self.min_stack:
            return self.min_stack[-1]
```

```
        else:
            return None  # 栈为空时返回None

# 使用 MinStack 类
min_stack = MinStack()

while True:
    try:
        command = input().strip()
        if command.startswith('push'):
            value = int(command.split()[1])
            min_stack.push(value)
        elif command.startswith('pop'):
            min_stack.pop()  # 空栈时不进行任何操作
        elif command.startswith('min'):
            min_value = min_stack.min()
            if min_value is not None:
                print(min_value)  # 只有在栈非空时才打印最小值
    except EOFError:
        break
```

22509：解方程

```
from math import log2

def find_x(y):
    # 定义方程
    def equation(x):
        return x**2 + x + 1 + log2(x)

    # 二分查找解
    left, right = 0, y  # x的解显然在0和y之间，因为当x=y时，x^2 + x + 1 + log2(x) > y
    while right - left > 1e-8:  # 精确到小数点后8位
        mid = (left + right) / 2
        if equation(mid) < y:
            left = mid
        else:
            right = mid
    return (left + right) / 2

results = []
try:
    while True:
        y = int(input())
        x = find_x(y)
        results.append(x)
except EOFError:
    pass

# 输出结果
for x in results:
    print(f"{x:.4f}")
```

23660:7的倍数取法有多少种

```
def count_combinations(numbers, index, current_sum, count):
    if index >= len(numbers):
        if current_sum % 7 == 0:
            return count + 1
        else:
            return count
    # 选择取当前位置的数
    count = count_combinations(numbers, index + 1, current_sum + numbers[index], count)

    # 选择不取当前位置的数
    count = count_combinations(numbers, index + 1, current_sum, count)

    return count

t = int(input())
for _ in range(t):
    data = list(map(int, input().split()))
    n = data[0]
    numbers = data[1:]

    result = count_combinations(numbers, 0, 0, 0)
    print(result)
```

23806：三数之和

```
def threeSum(nums):
    nums.sort()
    res = set()
    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        d = {}
        for x in nums[i + 1:]:
            if x not in d:
                d[-nums[i] - x] = 1
            else:
                res.add((nums[i], -nums[i] - x, x))
    return len(res)

n = list(map(int, input().split()))
print(threeSum(n))
```

23937：逃出迷宫

```
def dfs(mx, visited, x, y):
    # 如果到达右下角，返回True
    if x == n - 1 and y == n - 1:
        return True
    # 定义向右和向下的移动方向
    directions = [(0, 1), (1, 0)]

    for dx, dy in directions:
        nx = x + dx
        ny = y + dy
        # 检查新坐标是否在矩阵范围内，是否已经访问过，以及是否可以通过
        if 0 <= nx < n and 0 <= ny < n and not visited[nx][ny] and mx[nx][ny] == 0:
            visited[nx][ny] = True
            if dfs(mx, visited, nx, ny):
                return True
            visited[nx][ny] = False

    return False

# 读取输入
n = int(input())
mx = [list(map(int, input().split())) for _ in range(n)]

# 初始化访问标记数组
visited = [[False] * n for _ in range(n)]

# 起始点 (0, 0) 必须是可以通过的
if mx[0][0] == 1:
    print('No')
else:
    visited[0][0] = True
    if dfs(mx, visited, 0, 0):
        print('Yes')
    else:
        print('No')
```

24591：中序表达式转后序表达式

```
def infix_to_postfix(expression):
    precedence = {'+':1, '-':1, '*':2, '/':2}
    stack = []
    postfix = []
    number = ''

    for char in expression:
        if char.isnumeric() or char == '.':
            number += char
        else:
            if number:
                num = float(number)
                postfix.append(int(num) if num.is_integer() else num)
                number = ''
            if char in '+-*/':
                while stack and stack[-1] in '+-*/' and precedence[char] <= precedence[stack[-1]]:
                    postfix.append(stack.pop())
                stack.append(char)
            elif char == '(':
                stack.append(char)
            elif char == ')':
                while stack and stack[-1] != '(':
                    postfix.append(stack.pop())
                stack.pop()

    if number:
        num = float(number)
        postfix.append(int(num) if num.is_integer() else num)

    while stack:
        postfix.append(stack.pop())

    return ' '.join(str(x) for x in postfix)

n = int(input())
for _ in range(n):
    expression = input()
    print(infix_to_postfix(expression))
```

25302：最大并发量

```
# 注意同一时刻开始、结束的只计开始的。
t=int(input())
while t>0:
    t-=1
    n=int(input())
    a=[];b=[]
    for i in range(n):
        x,y=input().split()
        a.append(int(x))
        b.append(int(y))
    ans = 0
    for i in range(n):
        cnt=0
        for j in range(n):
            if a[i]>=a[j] and a[i]<b[j]:
                cnt+=1
        ans = max(ans, cnt)
    print(ans)
```

25353：排队

```
from collections import deque

n, d = map(int, input().split())
h = deque(int(input()) for _ in range(n))
ans = []

while h:
    inlist = []
    max_val = h[0]
    min_val = h[0]

    # 一次遍历完成所有必要的计算
    for _ in range(len(h)):
        height = h.popleft()
        if abs(height - max_val) <= d and abs(height - min_val) <= d:
            inlist.append(height)
        else:
            h.append(height)

        if height < min_val:
            min_val = height
        if height > max_val:
            max_val = height

    ans += sorted(inlist)

print(*ans, sep='\n')
```

25561:2022决战双十一

```
result = float("inf")
n, m = map(int, input().split())
store_prices = [input().split() for _ in range(n)]
you= [input().split() for _ in range(m)]
la=[0]*m
def dfs(i,sum1):
    global result
    if i==n:
        jian=0
        for i2 in range(m):
            store_j=0
            for k in you[i2]:
                a,b=map(int,k.split('-'))
                if la[i2]>=a:
                    store_j=max(store_j,b)
            jian+=store_j
        result=min(result,sum1-(sum1//300)*50-jian)
        return
    for i1 in store_prices[i]:
        idx,p=map(int,i1.split(':'))
        la[idx-1]+=p
        dfs(i+1,sum1+p)
        la[idx-1]-=p
dfs(0,0)
print(result)
```

25566:CPU 调度

```
def earliest_completion_time():
    n = int(input())  # 输入进程数
    tasks = []  # 存储每个进程的 (compute, write)

    for _ in range(n):
        compute, write = map(int, input().split())
        tasks.append((compute, write))

    # 按 write 从大到小排序，如果相等则按 compute 从大到小排序
    tasks.sort(key=lambda x: -x[1])

    current_time = 0  # 当前的 CPU 时间
    total_time = 0    # 所有进程的最早完成时间

    for compute, write in tasks:
        current_time += compute  # 执行计算任务
        total_time = max(total_time, current_time + write)  # 计算完成后写文件，并更新最早完成时间

    print(total_time)

earliest_completion_time()
```

25570：洋葱

```
n = int(input())
buffer = [0] * ((n+1)//2+1)
for i in range(1, n+1):
    line = [0]+list(map(int, input().split()))
    for j in range(1, n+1):
        k = min(i, j, (n+1)-i, (n+1)-j)
        buffer[k] += line[j]
print(max(buffer))
```

25573：红蓝玫瑰

```
r=list(input())
n=len(r)
R=[0]*n
B=[0]*n
if r[0]=="R":R[0]=0;B[0]=1
else:R[0]=1;B[0]=0
for i in range(n-1):
    if r[i+1]=="R":
        R[i+1]=R[i]
        B[i+1]=min(R[i],B[i])+1
    else:
        R[i+1]=min(R[i],B[i])+1
        B[i+1]=B[i]
print(R[-1])
```

25702：护林员盖房子 加强版

```
# ref: https://zhuanlan.zhihu.com/p/162834671

def maximalRectangle(matrix) -> int:
```

```
    # 求出行数n和列数m
    n = len(matrix)
    if n == 0:
        return 0

    m = len(matrix[0])
    # 存储每一层的高度
    height = [0 for _ in range(m+1)]
    res = 0
    # 遍历以哪一层作为底层
    for i in range(n):
        sk = [-1]
        for j in range(m+1):
            # 计算j位置的高度，如果遇到0则置为0，否则递增
            h = 0 if j == m or matrix[i][j] == '1' else height[j]
+ 1
            height[j] = h
            # 单调栈维护长度
            while len(sk) > 1 and h < height[sk[-1]]:
                res = max(res, (j-sk[-2]-1) * height[sk[-1]])
                sk.pop()
            sk.append(j)
    return res


m, n = map(int, input().split())
a = []
for i in range(m):
    a.extend([input().split()])

print(maximalRectangle(a))
```

25815: 回文字符串

```
def min_operations(s):
    n = len(s)
    dp = [[0]*n for _ in range(n)]
    for i in range(n-1, -1, -1):
        for j in range(i+1, n):
            if s[i] == s[j]:
                dp[i][j] = dp[i+1][j-1]
            else:
                dp[i][j] = min(dp[i+1][j], dp[i][j-1], dp[i+1][j-
1]) + 1
    return dp[0][n-1]

s = input().strip()
print(min_operations(s))
```

26646: 建筑修建

```
def generate_intervals(x, width, m):
    temp = []
    for start in range(max(0, x-width+1), min(m, x+1)):
        end = start+width
        if end <= m:
            temp.append((start, end))
    return temp


n, m = map(int, input().split())
plans = [tuple(map(int, input().split())) for _ in range(n)]
intervals = []
for x, width in plans:
    intervals.extend(generate_intervals(x, width, m))
intervals.sort(key=lambda x: (x[1], x[0]))
cnt = 0
last_end = 0
for start, end in intervals:
    if start >= last_end:
        last_end = end
        cnt += 1
print(cnt)
```

26971: 分发糖果

```
def candy(ratings):
    n = len(ratings)
    left = [0] * n
    for i in range(n):
        if i > 0 and ratings[i] > ratings[i - 1]:
            left[i] = left[i - 1] + 1
        else:
            left[i] = 1

    right = ret = 0
    for i in range(n - 1, -1, -1):
        if i < n - 1 and ratings[i] > ratings[i + 1]:
            right += 1
        else:
            right = 1
        ret += max(left[i], right)

    return ret

input()
*ratings, = map(int, input().split())
print(candy(ratings))
```

26976: 摆动序列

```
def sgn(x):
    if x == 0:
        return 0
    elif x > 0:
        return 1
    elif x < 0:
        return -1

n = int(input())
nums = list(map(int,input().split()))
delta = [sgn(nums[i+1]-nums[i]) for i in range(n-1)]

result = 1
pre = 0
for i in range(n-1):
    if delta[i] * pre < 0 or (pre == 0 and delta[i] != 0):
        result += 1
        pre = delta[i]
print(result)
```

26977: 接雨水

```
# DP
from typing import List

def trap(height: List[int]) -> int:
    if not height:
        return 0

    n = len(height)
    leftMax = [height[0]] + [0] * (n - 1)
    for i in range(1, n):
        leftMax[i] = max(leftMax[i - 1], height[i])

    rightMax = [0] * (n - 1) + [height[n - 1]]
    for i in range(n - 2, -1, -1):
        rightMax[i] = max(rightMax[i + 1], height[i])

    ans = sum(min(leftMax[i], rightMax[i]) - height[i] for i in
range(n))
    return ans

input()
*h, = map(int, input().split())
#h = [0,1,0,2,1,0,1,3,2,1,2,1]
ans = trap(h)
print(ans)
```

26978: 滑动窗口最大值

```
from typing import List
import collections

def maxSlidingWindow(nums: List[int], k: int) -> List[int]:

    n = len(nums)
    q = collections.deque()
    for i in range(k):
        while q and nums[i] >= nums[q[-1]]:
            q.pop()
        q.append(i)

    ans = [nums[q[0]]]

    for i in range(k, n):
        while q and nums[i] >= nums[q[-1]]:
            q.pop()
        q.append(i)
        while q[0] <= i - k:
            q.popleft()
        ans.append(nums[q[0]])

    return ans


n,k = map(int, input().split())
*nums, = map(int, input().split())
#nums = [1,3,-1,-3,5,3,6,7]
#k = 3
ans = maxSlidingWindow(nums, k)
print(' '.join(map(str, ans)))
```

27093: 排队又来了

```
from collections import defaultdict
import bisect

N,D = map(int,input().split())
#N, D = 5, 3
*info, = map(int, input().split())
heights = sorted(list(set(info)))

ass = {}
for i, h in enumerate(heights, 1):
    ass[h] = i  # 只是为了方便找到高度h排第几个

l = len(heights)
tree_l = [-1] * (l+1)   # 树状数组下标是1开始有效
tree_r = [-1] * (l+1)
ans = defaultdict(list)  # 存储分层结果：每层有哪些高度
def low_bit(x):
    return x & (-x)

def update(i, v, tree):
    while i <= l:
        tree[i] = max(v, tree[i])
        i += low_bit(i)

def get_max(i, tree):
    res = -1
    while i > 0:
        res = max(res, tree[i])
        i -= low_bit(i)
    return res


for h in info:  # 按照输入的顺序（即队伍顺序）扫描
    index = ass[h]
    left = bisect.bisect_right(heights, h-D-1) #left下标是0开始计算
    right = i - bisect.bisect_left(heights, h+D+1)
    storey = 1 + max(get_max(left, tree_l), get_max(right,
tree_r))
    #递推关系。分别找到小于h-D与大于h+D的高度所对应层数的最大值
    update(index, storey, tree_l)
    update(l+1-index, storey, tree_r)
    #更新高度h对应的点的层数最大值
    ans[storey].append(h)  # 加入结果中

res = []
for storey in sorted(ans.keys()):
    res.extend(sorted(ans[storey]))
print(' '.join(map(str, res)))
```

27103: 最短的愉悦旋律长度

```
N, M = map(int, input().split())
*melody, = map(int, input().split())

cnt = 1
note = set()
for i in melody:
    note.add(i)
    if len(note) == M:
        cnt += 1
        note.clear()

print(cnt)
```

27104: 世界杯只因

```
def min_cameras_to_cover_all(ranges):
    n = len(ranges)
    ptr = 0
    num = 0

    mx = max(ranges)
    while ptr < n:
        # 假设下一个指针位置为当前指针加上当前观测范围再加一
        nxt = ptr + ranges[ptr] + 1

        # 遍历一个以当前指针为中心的大窗口，考虑到最大观测范围mx的影响
        for i in range(max(0, ptr - mx), min(n, ptr + mx + 1)):
            if 0 <= i < n and i - ranges[i] <= ptr and i +
ranges[i] + 1 > nxt:
                nxt = i + ranges[i] + 1  # 更新最远可达位置

        num += 1  # 每次循环代表安装了一个摄像头
        ptr = nxt  # 移动到最远可达位置继续搜索

    return num

if __name__ == "__main__":
    _ = int(input())  # 忽略第一行的N值
    ranges = list(map(int, input().strip().split()))
    print(min_cameras_to_cover_all(ranges))
```

27122: 两球之间的磁力

```
n,m = map(int, input().split())
*position, = map(int, input().split())
position.sort()

def check(x):
    cnt, pre = 1, position[0]
    for i in range(1,n):
        if position[i] - pre >= x:
            cnt += 1
            pre = position[i]

    return cnt >= m

# https://github.com/python/cpython/blob/main/Lib/bisect.py
lo = 1
hi = position[-1] - position[0] + 1
ans = 1
while lo < hi:
    mid = (lo + hi) // 2
    if check(mid):
        ans = mid       # 如果cnt==m, mid就是答案
        lo = mid + 1
    else:
        hi = mid
print(ans)
```

27141: 完美的爱

```
def find_max_value(n, gifts):
    prefix_sum = [0] * (n + 1)
    for i in range(n):
        prefix_sum[i + 1] = prefix_sum[i] + gifts[i]

    max_value = 0
    for i in range(n):
        for j in range(i + 1, n + 1):
            subarray_sum = prefix_sum[j] - prefix_sum[i]
            subarray_length = j - i
            if subarray_sum / subarray_length == 520:
                max_value = max(max_value, subarray_sum)

    return max_value

# 读取输入
n = int(input())
gifts = list(map(int, input().split()))
```

```
# 计算结果
result = find_max_value(n, gifts)
print(result)
```

27217：有多少种合法的出栈顺序

```
import sys
sys.setrecursionlimit(1<<30)

def dfs(i, j, f):
    if f[i][j] != -1:
        return f[i][j]

    if i == 0:
        f[i][j] = 1
        return 1

    if j == 0:
        f[i][j] = dfs(i - 1, j + 1, f)
        return f[i][j]

    f[i][j] = dfs(i - 1, j + 1, f) + dfs(i, j - 1, f)
    return f[i][j]

n = int(input())
f = [[-1] * (n + 1) for _ in range(n + 1)]

result = dfs(n, 0, f)
print(result)
```

27237：体育游戏跳房子

```
from collections import deque
#import heapq

def bfs(s, e):
    q = deque()
    q.append((0, s, ''))
    vis = set()
    vis.add(s)
    # q = []
    #heapq.heappush(q, (0, s, ''))

    while q:
        step, pos, path = q.popleft()
        #step, pos, path = heapq.heappop(q)
        if pos == e:
            return step, path

        if pos * 3 not in vis:
            q.append((step+1, pos*3, path+'H'))
            vis.add(pos*3)
            #heapq.heappush(q, (step+1, pos*3, path+'H'))
        if int(pos // 2) not in vis:
            vis.add(int(pos//2))
            q.append((step+1, int(pos//2), path+'O'))
            #heapq.heappush(q, (step+1, int(pos//2), path+'O'))

while True:
    n, m = map(int, input().split())
    if n == 0 and m == 0:
        break
    step, path = bfs(n, m)
    print(step)
    print(path)
```

27310：积木

```
from collections import defaultdict
from itertools import permutations

a = defaultdict(int)
b = defaultdict(int)
c = defaultdict(int)
d = defaultdict(int)
n = int(input())

for i in input():
    a[i] += 1
for i in input():
    b[i] += 1
for i in input():
    c[i] += 1
for i in input():
    d[i] += 1

dicts = [a, b, c, d]

def check(word):
    for perm in permutations(dicts, len(word)):
        for i, d in enumerate(perm):
            if word[i] not in d:
                break
        else:
            return 'YES'
    else:
        return 'NO'

for _ in range(n):
    word = input()
    print(check(word))
```

27373：最大整数

```
limit = int(input())
n = int(input())
lst = list(map(str, input().split()))

def sorting(a, b):
    if int(a + b) > int(b + a):
        return True
    else:
        return False


for i in range(n):
    for j in range(i, n):
        if not sorting(lst[i], lst[j]):
            lst[i], lst[j] = lst[j], lst[i]

dp = [0] * (limit + 1)  # dp[i]表示i位数时最大值
temp = [0] * (limit + 1)
for j in lst:
    for i in range(len(j), limit + 1):
        dp[i] = max(dp[i], int(str(temp[i - len(j)]) + j))
    temp[:] = dp
print(dp[limit])
```

27384：候选人追踪

```
import heapq

maxn = 320000
cnt = [0] * maxn
n, k = 0, 0
vis = [False] * maxn

n, k = map(int, input().split())
*records, = map(int, input().split())
arr = [(records[i], records[i+1]) for i in range(0, 2*n, 2)]

Q = []
candidates = list(map(int, input().split()))
for i in range(k):
    heapq.heappush(Q, (0, candidates[i]))
    vis[candidates[i]] = True

arr = sorted(arr[:n])
if k == 314159:
    print(arr[n-1][0])
    exit()

rmx = 0
rs = 0
for i in range(n):
    c = arr[i][1]
    cnt[c] += 1
    if vis[c]:
        while cnt[Q[0][1]]: # 懒更新，每次只更新到堆中的最小值是实际的最小值
            f = heapq.heappop(Q)
            f = (f[0] + cnt[f[1]], f[1])
            heapq.heappush(Q, f)
            cnt[f[1]] = 0
    else:
        rmx = max(rmx, cnt[c])
    if i != n-1 and arr[i+1][0] != arr[i][0] and Q[0][0] > rmx:
        rs += arr[i+1][0] - arr[i][0]

print(rs)
```

27401：最佳凑单

```
a, b = map(int, input().split())
c = {0}
for i in map(int, input().split()):
    for j in c.copy():
        if j < b: c.add(i + j)
for i in sorted(c):
    if i >= b: print(i);exit()
print(0)
```

27528：跳台阶

```
dp = [0] * (int(input()) + 1)
dp[0] = 1
for i in range(1, len(dp)):
    dp[i] = sum(dp[:i])
print(dp[-1])
```

27623：取石子-较少那堆石子数目的整数倍

```
def dfs(a, b):
    # 优先达到条件者获胜
    if a // b >= 2 or a == b:
        return True
    else:
        # 继续回溯，如果后手先遇到，则后手胜利
        return not dfs(b, a - b)

while True:
    a, b = map(int, input().split())
    if a == 0 and b == 0:
        break

    # 保证多的石子在前
    if b > a:
        a, b = b, a

    # 调用回溯，直到遇到第一次 a/b >= 2
    if dfs(a, b):
        print("win")
    else:
        print("lose")
```

28389：跳高

```
from bisect import bisect_left

def min_testers_needed(scores):
    scores.reverse()  # 反转序列以找到最长下降子序列的长度
    lis = []  # 用于存储最长上升子序列

    for score in scores:
        pos = bisect_left(lis, score)
        if pos < len(lis):
            lis[pos] = score
        else:
            lis.append(score)

    return len(lis)


N = int(input())
scores = list(map(int, input().split()))

result = min_testers_needed(scores)
print(result)
```

28700:罗马数字与整数的转换

```
# 定义罗马数字和整数的映射关系
roman_to_int_map = {
    'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M':
1000
}

# 定义整数到罗马数字的映射列表（从大到小顺序）
int_to_roman_map = [
    (1000, 'M'), (900, 'CM'), (500, 'D'), (400, 'CD'),
    (100, 'C'), (90, 'XC'), (50, 'L'), (40, 'XL'),
    (10, 'X'), (9, 'IX'), (5, 'V'), (4, 'IV'), (1, 'I')
]

def roman_to_int(s):
    total = 0
    prev_value = 0
    for char in s:
        value = roman_to_int_map[char]
        if value > prev_value:
            total += value - 2 * prev_value  # 处理特殊情况，如IV,
IX
        else:
            total += value
        prev_value = value
    return total
```

```
# 整数转罗马数字
def int_to_roman(num):
    result = []
    for value, symbol in int_to_roman_map:
        while num >= value:
            result.append(symbol)
            num -= value
    return ''.join(result)
def main():
    input_data = input().strip()
    if input_data.isdigit():
        # 输入是整数
        num = int(input_data)
        print(int_to_roman(num))
    else:
        # 输入是罗马数字
        print(roman_to_int(input_data))

# 调用主函数
if __name__ == "__main__":
    main()
```

28780:零食兑换3

```
n, m = map(int, input().split())
coins = list(map(int, input().split()))
dp = [float("inf")] * (m + 1)
dp[0] = 1
for i in coins:
    dp[i] = 1
for i in range(1, m + 1):
    for coin in coins:
        if i - coin >= 0:
            dp[i] = min(dp[i], dp[i - coin] + 1)

#print(dp)
if dp[m] == float("inf"):
    print(-1)
else:
    print(dp[m])
```