Задача A. Середина игры

Заметим, что за любой результат в игре суммарно даётся 2 очка, значит и сумма очков в любой момент должна быть чётной. Если $A + B$ нечётное, то ответ «Error».

Если $A \geqslant 2$ и $B \geqslant 2$, то определить однозначно невозможно: пусть $M = min(A, B)$, невозможно понять, выиграл ли каждый по M раз или же было $2 \cdot M$ ничьих.

Для остальных случаев ответ $\lfloor\frac{A}{2}\rfloor, \lfloor\frac{B}{2}\rfloor, A \mod 2$.

Пример решения на Python:

```
a = int(input())
b = int(input())

if (a+b) % 2 == 1:
    print("Error")
    exit(0)
if a > 1 and b > 1:
    print("Undefined")
    exit(0)

print(a // 2, b // 2, a % 2)
```

Задача B. Поп-ит

Для решения задачи нужно посчитать количество нулей в заданной матрице, количество единиц в ней же, и вывести минимум из этих двух чисел.

Пример решения на Python:

```
n, m = map(int, input().split())
s = ""
for _ in range(n):
    s += input()
print(min(s.count("0"), s.count("1")))
```

Задача C. Ужин для интровертов

Оценка. Пусть L_i, R_i — количество свободных мест слева и справа для i-го интроверта. Тогда по условию $L_i+R_i \geqslant K$. Пусть людей за столом X, просуммируем по всем людям. $\sum L_i = \sum R_i = N-X$. Тогда $2 \cdot N - 2 \cdot X \geqslant X \cdot K, 2 \cdot N \geqslant X \cdot (K + 2), X \leqslant \frac{2 \cdot N}{K+2}$.

Пример. Будем для заданного X строить пример на минимальное N, полученное из оценки сверху. Заметим, что если мы добьемся того, что в примере сумма каждого $L_i + R_i = K$, то n в этом примере само собой станет минимальным подходящим под оценку, так как все неравенства станут равенствами.

Если $K = 2 \cdot y$, то между каждыми двумя соседями оставляем по y мест. Если $K = 2 \cdot y + 1$, то чередуются y и $y + 1$ пустых мест, в случае нечетности X получится рядом два отрезка по $y + 1$ мест, что даст дополнительную единичку в сумму всех L_i и R_i, что учтется формулой сверху, так как это единственный случай, когда $X \cdot (K + 2)$ будет нечетным, а с другой стороны неравенства стоит $2 \cdot N$ — четное число.

Пример решения на Python:

```
n = int(input())
k = int(input())
print(2 * n // (k + 2))
```

Задача D. Плохие ставки

Если игрок ставит на сумму $S < N$ или $S > N \cdot K$, то шансов победить у него нет. Разберём остальные случаи. Если $N = 1$, тогда любые две ставки равновероятны. Если $N > 1$, тогда нужно посчитать матожидание $\frac{N \cdot (K+1)}{2}$, чем дальше S от него (то есть значение $|S - \frac{N \cdot (K+1)}{2}|$ больше), тем меньше шансов на победу. Достаточно сравнить посчитанные значения.

Пример решения на Python:

```
n = int(input())
k = int(input())
s1 = int(input())
s2 = int(input())

def evaluate(s):
    if s < n or s > n * k:
        return int(1e100)
    if n == 1:
        return 0
    return abs(2 * s - n * (k + 1))

c1, c2 = evaluate(s1), evaluate(s2)

if c1 == c2:
    print("Equal")
elif c1 > c2:
    print("Second")
else:
    print("First")
```

Задача E. Клюкало

Если менять i-ю деталь, то можно уменьшить отклонение на $\frac{1}{s_i}$. Поэтому вначале лучше всего изменять детали, которые в стандарте весят меньше всего. Тогда можно изменять детали в порядке неубывания. Если текущее отклонение $K_c - \frac{|a_i - s_i|}{s_i} > K$, то деталь можно привести к стандарту напрямую, затем уменьшить K_c. Если же $K_c - \frac{|a_i - s_i|}{s_i} \leqslant K$, то нужно решить систему уравнений $K_c - \frac{x}{s_i} \leqslant K$ и $K_c - \frac{x-1}{s_i} > K$. Решением этой системы является $x = \lceil \frac{K_c - K}{s_i} \rceil$.

Чтобы избежать проблем с точностью, все вычисления следует проводить в дробном виде. Или же, например, так как знаменатели всех дробей не превосходят 10, можно домножить все вещественные числа на 2520 (это наименьшее общее кратное целых чисел от 1 до 10) и решать задачу в целых числах.

Пример решения на Python:

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

sraw = list(map(int, input().split()))
araw = list(map(int, input().split()))

s = []
c, z = 0, 2520
for i in range(n):
    s.append([sraw[i], araw[i]])
    c += (max(s[i]) - min(s[i])) * (z // s[i][0])

s.sort()

ans = 0
for i in range(n):
    if c <= k * z:
        break
    if c - (max(s[i]) - min(s[i])) * z // s[i][0] > k * z:
        ans += max(s[i]) - min(s[i])
        c -= (max(s[i]) - min(s[i])) * z // s[i][0]
```

```
    else:
        ans += ((c - k * z) * s[i][0]) // z
        if ((c - k * z) * s[i][0]) % z > 0:
            ans += 1
        break

print(ans)
```

Задача F. Маска для монстров

Очевидно, что кратчайшая маска должна идти по границе многоугольника, причём от одной вершины по кругу до другой. Тогда ответом будет разность периметра и длины наибольшей стороны.

Пример решения на Python:

```
import math

n = int(input())
a = []
for i in range(n):
    a.append([int(s) for s in input().split()])

d = []
for i in range(n):
    dx = a[i][0] - a[i - 1][0]
    dy = a[i][1] - a[i - 1][1]
    d.append(math.sqrt(dx ** 2 + dy ** 2))

print(max(d) - max(d))
```

Задача G. Наибольший наибольший общий делитель

Пусть G — искомый НОД. Поскольку на отрезке есть хотя бы два числа, делящихся на G, то $G \leqslant R - L$. Переберём все G от $R - L$ до 1 и для каждого смотрим, есть ли хотя бы два числа делящихся на него. Это можно сделать простой проверкой $L \leqslant \lfloor \frac{R}{G} \rfloor \cdot G - G$. Если это условие выполняется, то два искомых числа равны $\lfloor \frac{R}{G} \rfloor \cdot G - G$ и $\lfloor \frac{R}{G} \rfloor \cdot G$.

Пример решения на Python:

```
l, r = map(int, input().split())

mx = 0
for i in range(1, r - l + 1):
    if i * (r // i) - i >= l:
        mx = i

print(mx * (r // mx) - mx, mx * (r // mx))
```

Задача H. Прогрессивный NoSQL

Заведём множество уже выданных имён и словарь, в котором для каждого введённого имени будет хранится предыдущий дописанный числовой суффикс. Когда приходит новый запрос, если его нет в множестве, добавляем это имя в множество, а в словаре записываем 0. Если же запрос есть в множестве, то пытаемся увеличить предыдущий дописанный числовой суффикс до тех пор, пока не найдётся имя, которого нет в множестве, после этого обновляем этот числовой суффикс.

Нетрудно заметить, что этот алгоритм будет работать за $O(Q \cdot \log_{10} Q)$.

Пример решения на Python:

```
previous_counter = dict()
has = set()
```

```
q = int(input())
for _ in range(q):
    name = input()
    counter = previous_counter.get(name, 0)
    while True:
        try_add = name
        if counter != 0:
            try_add += str(counter)
        counter += 1
        if not try_add in has:
            has.add(try_add)
            print(try_add)
            previous_counter[name] = counter
            break
```

Задача I. Марго покидает Мегабайтбург

Создадим граф, в котором вершинами будут пустые клетки, рёбрами с весом 0 — ходы из пустой клетки (i, j) в пустые клетки из $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, $(i, j+1)$, а также рёбра с весом 1 — ходы из пустой клетки (i, j) в пустые клетки из $(i-2, j)$, $(i+2, j)$, $(i, j-2)$, $(i, j+2)$. В таком графе можно запустить 0-1 BFS, который посчитает кратчайшее расстояние от общежития до аэропорта, это расстояние означает минимальное количество прыжков, которое понадобится, чтобы добраться от общежития до аэропорта. Это расстояние надо сравнить с K, и вывести соответствующий ответ.

Задача J. Snakes&Snakes

Вначале посчитаем, куда ведёт каждый телепорт (чтобы избегать цепочек телепортов). Для этого заведём массив e_i, в котором если $p_i = 0$, то $e_i = i$, иначе $e_i = e_{i-p_i}$. Если заполнять этот массив в порядке ввода, то все предыдущие значения будут посчитаны.

Заведём массив d_i — минимальное количество ходов, чтобы добраться до i-й клетки. Изначально $d_0 = 0$, остальные $d_i = +\infty$. Будем обрабатывать клетки без телепорта в порядке ввода, если клетка была обработана раньше, то мы её пропускаем, после этого обновляем $d_{e_{i+x}} = \min(d_{e_{i+x}}, d_i + 1)$, где x — числа от 1 до 6, помечаем клетку, как обработанную. Затем обрабатываем клетку с номером e_{i+6}, но только если она не была уже обработана, причём $d_{e_{e_{i+6}+x}}$ будет обновляться также значением $d_i + 1$. Будем продолжать обрабатывать такие клетки до тех пор, пока не встретим уже обработанную, после этого переместимся в следующую клетку в порядке ввода.

Можно легко понять, что поскольку телепорты всегда отправляют назад, то на каждом шаге расстояния будут посчитаны только для префикса, причём если $i < j$, то $d_i \leqslant d_j$. Это означает корректность алгоритма, который будет работать за $O(N)$.

Задача K. Бинарные деревья

Заметим, что операция переподвешивания обратимая, поэтому сведём оба дерева к бамбуку. Докажем, что одно дерево можно свести к бамбуку за не более чем $\frac{N-1}{2}$ операций. Покажем это по индукции — для дерева из одной вершины это можно сделать за $0 \leqslant \frac{1-1}{2}$ операций. Пусть теперь дерево размера k, если у корня один сын, то решаем для его поддерева за $\frac{k-2}{2} < \frac{k-1}{2}$. Если два сына — пусть размеры их поддеревьев a и b, тогда $k = a + b + 1$. Сводим каждое поддерево к бамбуку за $\frac{a-1}{2}$ и $\frac{b-1}{2}$ операций соответственно, затем переподвешиваем второе поддерево к самой нижней вершине первого. Всего выйдет операций $\frac{a-1+b-1}{2} + 1 = \frac{a+b}{2} = \frac{k-1}{2}$.

Задача L. Апокалипсис

Заметим, что заражение распространяется на одинаковое расстояние от изначального многоугольника, и его площадь в i-й день равна $S_i = 2^i \cdot S_0$. Пусть расстояние, на которое распространилось заражение в i-й день, равно d_i. Тогда $S_i = S_0 + P_0 \cdot d_i + \pi d_i^2$, где P_0 — периметр изначального многоугольника. Это квадратное уравнение, найдя решения для которого, можно найти d_i для каждого из дней. Нетрудно заметить, что при данных ограничениях на входные данные $d_{70} > 10^{10}$, это означает, что любое из поселений будет заражено в первые 70 дней, поэтому достаточно быстро посчитать расстояние D_j от каждого поселения до изначального многоугольника и найти первый день i, когда $d_i \geqslant D_j$.

Чтобы быстро считать расстояние от точки до выпуклого многоугольника достаточно сделать несколько шагов:

1. Проверить, лежит ли точка внутри многоугольника за $O(\log N)$, если лежит внутри, то расстояние равно 0, иначе выполнить следующие шаги;

2. Построить касательные из точки к многоугольнику за $O(\log N)$, тем самым найдутся две вершины многоугольника с индексами l и r;

3. Многоугольник разбивается на две части найденными вершинами. Поскольку многоугольник выпуклый, то в одной из частей расстояния до точки будут представлять из себя выпуклую функцию, по которой можно запустить тернарный поиск минимального значения. На самом деле, достаточно найти ближайшую вершину, а затем выбрать минимальное расстояние до смежных с этой вершиной отрезков-сторон. Этот шаг алгоритма работает также за $O(\log N)$.

Таким образом, алгоритм нахождения подходящего дня для каждого поселения работает за $O(\log N)$, а всё решение будет работать за $O(Q \cdot \log N)$.