Задача A. Новый функционал

Решение **эмуляцией**:

- эмулируем каждое нажатие отдельно;
- номер текущей задачи пересчитываем по правилам, описанным в условии;

Примерный псевдокод такой:

```
while (a != c || b != d) {
        ans = ans + 1

        if (b == n) {
                a = a + 1;
                b = 0;
        } else {
                b = b + 1
        }
}
```

Итоговая асимптотика $O((c-a)\cdot n)$. c и n не превосходят 10^3, поэтому суммарно получится не более 10^6 итераций.

Решение **формулой**:

- нумерация задач сквозная и периодическая (с периодом $n+1$);
- перейдем к единому номеру $F(v,k) = v\cdot(n+1)+k$;
- в таком случае ответом на задачу является $F(c,d) - F(a,b)$.

Это классический приём для сквозной нумерации. Например, если $n = 59$, то мы получаем стандартную задачу о количестве времени между (h_1, m_1) и (h_2, m_2).

Задача B. Научные гипотезы

В задаче представлена стандартная задача **минимакса**.

Один из частых способов решения таких задач следующий:

- попробуем зафиксировать ограничение на максимум $pmax$;
- проверим, что существует хотя бы какой-то ответ, не выходящий за рамки ограничения.
- будем уменьшать $pmax$, пока ответ продолжает существовать.

В данном случае «ответ существует» равносильно «существует какой-то массив p, где максимальное значение не превосходит $pmax$, причем все получающиеся реакции верные, а значения f неотрицательны».

Докажем, что такой подход работает. Обозначим функцию проверки существования ответа через $G(pmax)$.

Можно заметить два свойства:

- если $G(pmax)$ истинно, то и $G(pmax{+}1)$ истинно — если мы построили массив p с ограничением сверхху $pmax$, то и с ограничением $(pmax+1)$ подойдет тот же массив.
- если $G(pmax)$ ложно, то и $G(pmax-1)$ ложно — если мы не смогли построить массив p с ограничением $pmax$, то дальнейшее уменьшение ограничения никак не поможет.

Это означает, что для поиска оптимального решения можно применить **двоичный поиск по ответу**.

Поймем, как должна работать функция G изнутри:

- заметим, что из-за требования неотрицательности f нам выгодно поддерживать фантазию профессора как можно выше.
- в то же время мы не хотим сильно поднимать фантастичность гипотез p, так как у нас есть ограничение сверху $pmax$.

Рассмотрим несколько случаев при обработке i-й реакции:

- Если профессор не верит, то мы просто присваиваем $p_i = f_{i-1} + c + 1$.

  Таким образом, и профессор не верит в гипотезу, и фантастичность истории как можно ниже.
- Если профессор верит, то есть два варианта развития событий:
  - Профессор будет верить во все оставшиеся истории.

    В таком случае мы можем присвоить $p_i = \min(pmax, f_{i-1} + c)$.

    Таким образом, мы и не превышаем границу сверху $pmax$, и делаем гипотезу достаточно правдоподобной для профессора.
  - Далее будет хотя бы одна реакция неверия.

    В таком случае нам придётся присвоить $p_i = \min(pmax - c - 1, f_{i-1} + c)$

    Таким образом, в случае будущей реакции неверия мы сможем прибавить $(c+1)$ и остаться в пределах нашей границы $pmax$.

В итоге $pmax$ подойдет, если в процессе эмуляции процесса:

- все значения f_i будут неотрицательны;
- все значения p_i будут не превышать границу сверху $pmax$.

В качестве правой границы двоичного можно использовать, например, $R = f_0 + n \cdot (c + 1)$ — случай, когда профессор все n раз не верил.

Итоговая асимптотика $O(n \cdot \log R)$.

Задача C. Завтра будет лучше, чем вчера

Вычислим $(n - 1)$ вектор — разницу между соседними точками: $D_i = (x_{i+1} - x_i, y_{i+1} - y_i)$.

По условиям задачи нужно, чтобы в новой системе координат для всех векторов выполнялись условия:

- либо $Dx_i > 0$.
- либо $Dx_i = 0$ и $Dy_i > 0$.

Далее будем решать задачу относительно векторов D.

У нас есть n векторов D_i и нам нужно найти вектора Ox и Oy такие, что:

- Ox и Oy перпендикулярны;
- в системе координат (Ox, Oy) все $Dx_i \geqslant 0$.
- если $Dx_i = 0$, то $Dy_i > 0$.

Сформулируем второе правило в терминах векторов: $(Ox \cdot D[i]) \geqslant 0$, где $\cdot$ обозначает **скалярное произведение** векторов.

В самом деле, скалярное произведение связано с косинусом угла, который напрямую связан с x-координатой вектора в системе координат.

Далее заметим, что для существования итогового ответа все векторы D должны лежать в одной полуплоскости.

Мы будем искать вектор Ox, исходя из этого предположения, а в конце просто проверим, что найденная нами система координат удовлетворяет всем представленным условиям.

Если все векторы лежат в одной полуплоскости, то максимальный угол между двумя векторами строго меньше π. Равен π угол быть не может, так как в этом случае вектора могут лежать только строго на оси Oy, причем хотя бы один из них будет иметь $Dy_i < 0$.

Предположим, что мы нашли один из крайних векторов Dm.

В таком случае углы между Dm и всеми векторами D_i будут иметь один и тот же знак (либо всегда $\leqslant 0$, либо всегда $\geqslant 0$).

Для определенности давайте выберем неположительный знак ($\leqslant 0$).

В терминах векторов это можно записать как $(Dm \times D_i) \leqslant 0$, где $\times$ обозначает **векторное произведение** векторов.

В самом деле, векторное произведение связано с синусом угла, который неотрицателен для всех углов от 0 до π.

Таким образом, вектор Dm можно найти в формате «поиска максимума»:

- инициализируем $Dm = D_1$;
- если $(Dm \times D_i) > 0$ — заменим $Dm = D_i$.

В итоге Dm будет содержать «крайний максимальный» вектор из заданных. Осталось понять, как из Dm получить вектор Ox.

Мы знаем, что Dm в худшем случае находится где-то «с краю» полуплоскости — близко к положительному направлению будущей оси Oy.

В таком случае ничего не мешает в качестве Oy взять сам вектор Dm — его координата x будет равна 0, а $y > 0$.

В качестве Ox в таком случае необходимо выбрать вектор, перпендикулярный Dm.

Из двух возможных векторов выберем тот, у которого $(Dm \times Ox) < 0$. При таком выборе угол между Ox и всеми остальными векторами будет не превышать $\frac{\pi}{2}$ по модулю.

Вычислить подходящий Ox легко — он равен $(Dmy, -Dmx)$:

- $(Dm \cdot Ox) = x \cdot y - y \cdot x = 0$.
- $(Dm \times Ox) = x \cdot (-x) - y \cdot y < 0$.

Теперь для каждого вектора D_i необходимо проверить, что все условия выполняются (в ином случае ответ `No solution`):

- $(D_i \cdot Ox) \geqslant 0$.
- Если $(D_i \cdot Ox) = 0$, то $(D_i \cdot Oy) > 0$.

Задача D. Теория заговора

Для начала заметим, что:

- порядок элементов влияет только на формат вывода.
- любой заговор содержит только уникальные a_i.

Первым делом избавимся от совпадающих значений в массиве a. В дальнейшем мы будем исходить из того, что все значения a_i уникальны.

Далее отсортируем массив пар $B_i = (a[i], i)$ в лексикографическом порядке по возрастанию. В последующем под a_i мы будем подразумевать именно первый элемент пары B_i.

Формализуем условие:

- Из a_j можно перейти в a_i, если
  - $j < i$;
  - $\gcd(a_j, a_i) > 1$.
- Найдите последовательность переходов максимальной длины.

Обратите внимание, что условия перехода запрещают наличие циклов по определению. Таким образом мы решаем задачу **максимального пути в ациклическом ориентированном графе**.

Для начала вспомним, как такая задача решается в общем случае:

- вершины обрабатываются в порядке топологической сортировки графа;
- используется динамическое программирование: dp_i — максимальная длина пути, заканчивающегося в вершине i.
- $dp_i = \max(dp_j) + 1$ — вычисляется за $O(deg_i)$ перебором рёбер (j, i).
- Ответ вычисляется как $\max(dp_i)$ по всем i.

Итоговая асимптотика равна $O(\sum deg_i) = O(E)$ действий по обработке рёбер и $O(V)$ действий по обработке вершин.

В данной задаче граф является неявным: вершины — позиции от 1 до n, а рёбра заданы неявно через условие перехода.

Получается, что решение «втупую» отработает за $O(n^2 \cdot \log A)$:

- вершины по умолчанию заданы в порядке топологической сортировки за счёт $j < i$.
- $deg_i = i - 1$, поэтому $E = O(n^2)$;
- проверка наличия ребра (j, i) выполняется за $O(\log A)$ — необходимо вычислить НОД (gcd) алгоритмом Евклида.
- дополнительных обработок вершины (позиции) не требуется.

Как оптимизировать описанное выше решение?

Заметим, что если $\gcd(X, Y) > 1$, то X и Y имеют хотя бы один общий простой множитель. Иначе говоря, нам не важны сами числа a_i — нам важно лишь, какие простые множители в них входят.

Обозначим **множество** простых множителей числа a_i через P_i.

Сразу заметим, что размер P_i в данной задаче не превосходит 7: $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 < 10^6 < 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$.

Обозначим это ограничение через $Pmax$.

Пока что это позволяет лишь видоизменить проверку наличия ребра: вместо вычисления $\gcd(a_i, a_j)$ будем проверять наличие общего элемента в множествах P_i и P_j.

Асимптотику это практически не изменит — $O(n^2 \cdot Pmax)$, — но позволит нам провести последующие оптимизации.

Введём дополнительный массив $dpF_{i,q}$ — максимальная длина пути, заканчивающегося в вершине i, а **следующий** переход можно произвести по простому множителю q.

В таком случае пересчёт основного массива будет следующим:

- вычисляем $dp_i = \max(dpF_{j,q}) + 1$ по $q \in P_i$ и по $j < i$.
- записываем $dpF_{i,q} = dp_i$.

Заметим, что на первом шаге нас интересует максимальный путь, заканчивающийся в любой из вершин $1 \dots (i-1)$ (с учетом простого числа q) независимо от вершины j.

Поэтому давайте введём массив $dpP_{i,q} = \max(dpF_{j,q})$ по $j \leqslant i$ — оптимум на префиксе.

Обновим формулы пересчёта:

- $dp_i = dpP_{i-1,q} + 1$ по $q \in P_i$.
- $dpP_{i,q} = \max(dpP_{i-1,q}, dp_i)$ для всех q.

Это позволило оптимизировать первый шаг, но второй шаг (обновление dpP) остаётся неэффективным.

Заметим, что вычисление dp_i зависит только от dpP_{i-1}, а dpP_i — от dpP_{i-1} и dp_i.

В таком случае можно применить технику **сканирующей прямой**, позволяющей избавиться от индекса i в массиве dpP:

- Сначала вычислим $dp_i = dpP_q + 1$ по $q \in P_i$.
- Затем обновим $dpP_q = \max(dpP_q, dp_i)$ по $q \in P_i$.

В итоге мы получили вычисление динамики за $O(n \cdot Pmax)$.

Остался один вопрос: как быстро вычислять множество P_i для заданного a_i?

Построим массив $minP_x$ — минимальный простой делитель числа x.

В таком случае для построения P_i нам достаточно начать с числа $x = a_i$ и делить на $minP_x$ до тех пор, пока не окажемся в $x = 1$.

Все уникальные простые q, встреченные в процессе, и будут составлять наше множество P_i.

Эффективно вычислить $minP$ можно, используя решето Эратосфена:

- для простых чисел $minP_q = q$;
- при переборе множителей простого числа $x = q \cdot j$ обновим $minP_x = q$, если $q \leqslant j$.

Итоговая асимптотика равна $O(A \cdot \log\log A + n \cdot Pmax)$, где первое слагаемое — вычисление $minP$, второе — вычисление dp и dpP.

Задача E. Тренировочные сборы

Давайте построим формальную модель задачи:

- Дан ориентированный ациклический граф (DAG).
- В графе есть две вершины Tm и Tk такие, что в них не входит ни одно ребро.
- Надо построить два пути (начинающиеся из Tm и Tk) такие, что объединение путей содержит все n вершин графа.

Для начала произведём топологическую сортировку графа, запустив соответствующий обход от вершин Tm и Tk.

Если какая-то из вершин графа **не была посещена** в процессе данных обходов — значит ответа точно не существует.

Иначе мы, как минимум, можем достичь всех вершин (но не гарантируется, что на это хватит двух путей).

Для каждой вершины v введём показатель «уровня» L_v:

- $L_{Tm} = L_{Tk} = 0$;

- для вершины t уровень $L_t = \max(L_f) + 1$ по всем рёбрам (f, t).

Так как граф строго ациклический, то уровни вершин L_v на любом пути будут строго возрастать. Это позволяет строить ответ жадной эмуляцией, перебирая уровни L от 1 до n по возрастанию:

- обозначим текущие «концы» путей через m и k.
- если на уровне L нет вершин — то обход закончен.
- если на уровне L есть больше двух вершин — то ответа нет (так как путей всего два).
- если на уровне L ровно одна вершина v — пробуем перейти туда и из вершины m, и из вершины k.

  Если не получилось ни одного перехода, то ответа не существует.
- если на уровне L две вершины v_1 и v_2, то пробуем перейти:
  - из m в v_1 и из k в v_2 (должны выполниться оба перехода);
  - из k в v_1 и из m в v_2 (должны выполниться оба перехода);

  Если ни один вариант не удался, то ответа не существует.

Задача F. Волонтёрство

Дано n малых отрезков $[b_i; e_i]$ и один большой отрезок $[S; F]$.

Необходимо для каждого $h = 1 \dots n$ вычислить a_h — суммарная длина большого отрезка, покрытая менее чем h малыми отрезками.

Для начала давайте вычислим величину c_h — суммарную длину большого отрезка, покрытую **ровно** h малыми отрезками.

В таком случае $a_h = c_0 + c_1 + \dots + c_{h-1} = a_{h-1} + c_{h-1}$.

Для вычисления c_h можно применить классический приём **сортировки событий**:

- для каждого малого отрезка создадим два события $(b_i, OPEN)$ и $(e_i, CLOSE)$.
  - если $e_i < S$, то события для отрезка не добавляем;
  - если $b_i > F$, то аналогично игнорируем отрезок;
  - событие $OPEN$ имеет координату $\max(b_i, S)$;
  - аналогично событие $CLOSE$ имеет координату $\min(e_i, F)$.
- обработаем события в лексикографическом порядке по возрастанию;

  в случае равенства времён поставим закрывающие события раньше, чем открывающие ($CLOSE < OPEN$).

Обработка i-го события (T_i, D_i) выглядит следующим образом:

- Пусть OpS — количество «открытых» отрезков, а LT — последнее обработанное время.
- Увеличим c_{OpS} на величину $(T_i - LT)$.
- Если $D_i = OPEN$, то OpS увеличивается на 1, иначе — уменьшается на 1.
- Изменяем $LT = T_i$.

Задача G. Изнурительные тренировки

Для начала вычислим N_p и N_m — минимальные количества дней, за которые можно «набрать форму» по первому и второму навыкам.

Рассмотрим вычисление N_p (для N_m рассуждения аналогичны):

- если $E \leqslant S_p$, то $N_p = 0$.
- иначе $N_p = \lceil \frac{E-S_p}{D_p} \rceil$ — округленное вверх количество «увеличений» навыка.

Предположим, что $N_p \leqslant N_m$.

В таком случае навык придумывания (m) мы будем тренировать усиленно всё время, а вот навык печати (p) можно иногда и потренировать «обычным» способом.

Обозначим через U количество дней с обычной тренировкой навыка p.

В таком случае должно выполняться неравенство $N_p \leqslant (N_m - U) + \frac{U}{2}$ или $2 \cdot N_p \leqslant 2 \cdot (N_m - U) + U = 2 \cdot N_m - U$.

Получаем ограничение $U \leqslant 2 \cdot (N_m - N_p)$. Также надо не забыть, что $U \leqslant N_m$ из нашего предположения.

Так как мы хотим как можно больше дней тренироваться «обычно», то оптимально взять верхнюю границу $U = \min(N_m, 2 \cdot (Nm - Np))$.

Зная U, легко вычислить общую величину ответа:

- $N_m \cdot 4 \cdot T_m$ для тренировки придумывания в усиленном режиме.
- $U \cdot T_p$ для обычной тренировки печати.
- $(N_m - U) \cdot 4 \cdot T_p$ для усиленной тренировки печати.

Случай $N_p > N_m$ рассматривается аналогично.

Задача H. Томительное ожидание

Данная задача подразумевает простую (но эффективную) **эмуляцию** процесса ожидания:

- Если Анатолий стоит на остановке B во время T, то найдем ближайший автобус в направлении остановки A.
  - Пусть это автобус b_i, в таком случае $b_{i-1} < T \leqslant b_i$.
  - Анатолий поедет на этом автобусе, если $b_i + d < a_n - d$.
  - Увеличим счётчик автобусов и переместим Анатолия на остановку во время $b_i + d + 1$.
- Аналогично, если Анатолий стоит на остановке A во время T, то найдем ближайший автобус в направлении остановки B.
  - Пусть это автобус a_j, в таком случае $a_{j-1} - d < T \leqslant a_j - d$.
  - Анатолий поедет на этом автобусе, если $a_j < a_n - d$.
  - Увеличим счётчик автобусов и переместим Анатолия на остановку B во время $a_j + 1$.

Эффективный поиск «ближайших» автобусов можно выполнить одним из двух способов:

- **Двоичным поиском** по массивам a и b.

  Языки `C++` / `Java` / `Python` содержат встроенные реализации такого поиска.

  Итоговая асимптотика равна $O(n \cdot \log n + m \cdot \log m)$.
- Техникой «**двух указателей**».
  - Поддерживаем указатели i и j на последние рассмотренные автобусы.
  - С течением времени указатели могут только увеличиваться.

  Итоговая асимптотика будет равна $O(n + m)$.

Задача I. Справедливое разнообразие

Вычислим для каждого заведения величину c_i — количество посещений заведения i за первые D дней.

В таком случае ответ не может быть меньше $M = \max(c_i)$.

От c_i перейдем к $h_i = M - c_i$ — сколько заведению i не хватает посещений до максимума.

Будем эмулировать процесс следующим образом:

- процесс останавливается, когда все $h_i = 0$.
- если есть **ровно одно** $h_v > 0$, то увеличиваем **все** h_i на 1 (после чего увеличиваем).
- выбираем два заведения с наибольшей величиной h_i, после чего посещаем их (уменьшаем h_i на 1).

Почему этот процесс даст верный результат?

Допустим, что в процессе эмуляции остался только один $h_v > 0$.

- Пусть $h_v > 1$.

  В таком случае это заведение было со старта эмуляции «самым непосещенным» и каждый день мы обязательно ходили в него и еще куда-то.

  Так как даже такого «активного» посещения не хватило для ответа, то приходится ещё минимум один раз посетить каждое заведение.

- Теперь пусть $h_v = 1$.

  В таком случае мы знаем, что суммарное необходимое количество для текущего значения получается нечётным, а такого ответа быть не может по условию задачи.

  Так как все прошлые шаги мы совершали оптимально (исходя из случая $h_v > 1$), то нам остаётся лишь ещё раз посетить все заведения, чтобы суммарное количество посещений стало чётным.

Для эффективной эмуляции процесса нам понадобится структура данных, которая умеет выполнять эффективно следующие операции:

- добавить значение;
- вернуть экстремум (минимум или максимум);
- извлечь соответствующий экстремум.

Известно, что структуры данных «бинарная куча» и «дерево поиска» способны выполнять все эти операции за $O(\log N)$ или эффективнее:

- C++: `std::priority_queue` и `std::set`;
- Java: `PriorityQueue` и `TreeSet`;
- Python: пакет `heapq`.

Осталось понять максимальное количество шагов эмуляции:

- величину можно сверху оценить как $2 \cdot D$;

  например, такая оценка достигается при $n = 3$ мы D дней ходим в заведения $(1, 2)$.

- в таком случае наибольшее количество итераций будет равно $2 \cdot D \cdot n = O(D \cdot n)$.

Итоговая асимптотика решения эмуляцией $O(D \cdot n \cdot \log n)$.

Также у этой задачи существовало решение без применения структур данных.

На самом деле это то же самое решение с эмуляцией, но выполняющее некоторые проверки и вычисления за $O(1)$.

Введём обозначения $maxH = \max(h)$ и $S = \sum h$.

Текущий набор заведений можно посетить без повышения M, если

- $S \mod 2 = 0$ — суммарное количество оставшихся посещений чётное;
- $maxH \leqslant S - maxH$ — нам должно хватать парных заведений, чтобы полностью посетить «самое непосещённое» заведение.

Пока данные условия не выполняются — будем увеличивать все H на 1 (а также увеличивать M).