Задача 1. Мешок (7-8 классы)

У нас есть неизвестный диапазон целых чисел $[x; x+n-1]$, в котором ровно n различных чисел.

Если тролль доставал карточки только с числами от min (минимального значения из всех что показал тролль) до max (максимального значения из всех что показал тролль), значит, все эти числа точно лежат внутри диапазона $[x; x+n-1]$. Следовательно, диапазон тролля должен, как минимум, вмещать отрезок $[min; max]$. Отсюда следует несколько выводов:

1. Нижняя граница диапазона не может быть больше минимального числа: $x \leqslant min$.
2. Верхняя граница диапазона не может быть меньше максимального числа: $max \leqslant x+n-1$ или же $max - n + 1 \leqslant x$.

Таким образом, получаем: $max - n + 1 \leqslant x \leqslant min$.

Задача 2. Брачный договор. Один? ДВА! (7-11 классы)

Сначала нужно посчитать, сколько пар мы можем создать. Их количество равно: $\min(N, M)$. Для каждой пары мы можем заключать брак и расторгать его, что даст нам три документа за $K + L$ дней.

Обозначим за $[x]$ — целую часть числа x. За T дней мы можем заключить и расторгнуть брак для одной пары: $P = \left[\dfrac{T}{K+L}\right]$ раз. После этого остается еще $Rem = T - P \cdot (K + L)$ дней. Если $K \leqslant Rem$, то мы можем еще раз заключить брак, а для его расторжения уже дней не хватит.

Таким образом, итоговая формула выглядит так: $\min(N, M) \cdot \left(3 \cdot P + \left[\dfrac{Rem}{K}\right]\right)$

Задача 3. Урок физкультуры (7-11 классы)

Представим данные как граф, где вершины — это дети, каждое ориентированное ребро — требование, что один ребенок должен стоять раньше другого.

Задача свелась к ответу на вопрос, можно ли упорядочить вершины так, что все ориентированные ребра будут идти от вершины с меньшим порядком к вершине с большим порядком.

Для решения первой подгруппы можно было перебрать все возможные $N!$ порядков вершин, для каждого порядка вершин проверить каждое из M ребер. Если мы нашли порядок вершин, при котором все ребра идут от вершины с меньшим порядком к вершине с большим порядком, то ответ `YES`. Если ни для какого порядка вершин это условие не соблюдается, то ответ `NO`. В итоге решение работает за $O(T \cdot M \cdot N!)$

Для решения второй подгруппы можно было заметить, что мы можем действовать жадным алгоритмом: на каждом шаге выбирать произвольную вершину, в которую не входит ни одно ребро, и добавлять такую вершину следующей по порядку.

Это можно было реализовать следующим образом. Будем для каждой вершины, еще не выбранной ранее, считать количество ребер, которое входит в эту вершину. Учитываются только те ребра, оба конца которых не были выбраны ранее.

Если на каждом из N шагов была вершина, в которую не входит ни одно из рассматриваемых ребер, то ответ `YES`.

Если же на каком-то шаге еще остались вершины, и не можем выбрать ни одну из них для добавления в порядок, то подходящего порядка вершин не существует и ответ `NO`.

Выбор каждой вершины работает за $O(N + M)$, значит, один тестовый набор работает за $O(N \cdot (N + M))$, и все решение работает за $O(T \cdot N \cdot (N + M))$.

Для нахождения полного решения, заметим, что можно оптимизировать предыдущее решение.

Для этого вначале для каждой вершины подсчитаем, сколько ребер в нее входят.

Сохраним отдельно все вершины, в которые не входит ни одно ребро, например, в очереди.

Если на каком-то шаге очередь пуста, подходящего порядка вершин не существует и ответ будет `NO`.

Иначе берем вершину из очереди, и заметим, что при добавлении ее в порядок перестанут учитываться только те ребра, которые исходят из этой вершины. Значит, достаточно обработать только

такие ребра, и если для рассматриваемого ребра в вершину на конце не входит только одно ребро, то добавляем такую вершину в очередь.

Каждый тестовый случай будет работать за $O(N + M)$, и все решение будет работать за $O(T \cdot (N + M))$.

Такое решение **не пройдет по времени**, так как в условии **нет дополнительного ограничения на сумму N**.

Поэтому заметим, что на ответ никак не влияют вершины, в которые не входит и из которых не выходит ни одно ребро. Для этого достаточно перебирать не все вершины, а только те вершины, которые являются началом или концом какого-либо ребра. Таких вершин не более $2 \cdot M$, поэтому решение будет работать за сумму количества ребер по всем подтестам, которое уже ограничено числом $5 \cdot 10^5$

Альтернативное полное решение

Можно было заметить, что такой порядок вершин существует тогда и только тогда, когда нет циклов в ориентированном графе.

Проверять наличие цикла в графе можем с помощью поиска в глубину.

Вспомним, что ребра перехода в поиске в глубину образуют дерево. Для каждой вершины будем хранить одно из трех значений: 0, если вершина еще не была посещена, 1, если вершина была посещена, но еще не обошли все ребра из нее, и 2 — если для вершины обошли все ребра и вернулись в ее предка при обходе.

Если в какой-то момент в процессе обхода рассматриваемое ребро ведет в вершину со значением 1, то это ребро идет от вершины к какому-то его дальнему предку, тем самым образовав цикл, и ответ `NO`.

Если такого ни разу не произошло, то цикла нет, и ответ `YES`.

Задача 4. ШИМ калибровка (7-8 классы)

Для решения первой подзадачи можно воспользоваться простым перебором. Вариантов написания перебора при таких ограничениях очень много, а потому их реализацию оставим в качестве упражнения.

Для решения второй подзадачи можно написать перебор по времени, проверив все возможные значения T от 1 до 1000. Первое подходящее (делящее импульсы на нужное число пакетов) и будет решением.

Для решения третьей и четвёртой подзадач заметим, что количество пакетов зависит только от «разрывов» (разниц $t_{i+1} - t_i$) между импульсами.

Не сложно догадаться, что перебирать все возможные значения T — бессмысленно, ведь гораздо логичнее и правильнее перебрать только значения разрывов между импульсами. Почему? Если мы введём функцию числа пакетов от времени $p(T)$, то очевидно, что она будет менять своё значение только в тех точках, где она будет пересекать одну из разностей $t_{i+1} - t_i$, а значит, рассматривать остальные точки смысла не имеет. Более того, не сложно понять, что данная функция будет являться монотонно убывающей (чем больше значение T — тем меньше значение функции $p(T)$).

Отсюда следует два решения, которые мы можем написать:

РЕШЕНИЕ 1 (Бинарный поиск)

Так как число пакетов можно представить как монотонно-убывающую функцию $p(T)$, то найти время T можно с помощью бинарного поиска для T, двигаясь влево или вправо по числу пакетов.

Сложность решения $O(n \cdot \log(t_n))$.

РЕШЕНИЕ 2 (Сортировка)

Зная, что число пакетов уменьшается с увеличением T, мы можем вычислить разности между импульсами, записать их в массив и затем отсортировать.

Нам нужно получить k пакетов, то есть нам нужны $k - 1$ самых больших разрывов (разниц $t_{i+1} - t_i$). Остальные должны быть $\leqslant T$. То есть $T = d[k - 1]$, где d — отсортированный массив разностей между импульсами.

Однако это верно тогда и только тогда, когда $d[k - 2] \geqslant d[k - 1]$. То есть последний разрыв, который должен быть «разделителем пакета», должен быть строго больше порогового T.

Если $d[k-2] = d[k-1]$, то при этом T они оба станут либо $\geqslant T$, либо $\leqslant T$. То есть невозможно получить ровно k пакетов.

Итоговая сложность решения: $O(n \cdot \log(n))$.

Подзадача четыре отличается от подзадачи три большими требованиями к качеству реализации решения.

Задача 5. Диджей Иван (7-11 классы)

Для решения первой подзадачи было достаточно перебрать и проверить все возможные расстановки «медляков» за $O(2^N \cdot N)$.

Для решения второй подзадачи обозначим за `dp[i]` ответ на задачу для префикса из треков от 1 до i включительно. Тогда `dp[N]` будет являться ответом на задачу.

`dp[0]` обозначим за 1, все остальные — за 0.

Теперь для каждого i от 1 до N включительно пройдемся по j от $i-1$ до 0 включительно. Если сумма длин треков на отрезке $[j+1; i]$ лежит в промежутке $[x; y]$, это значит, что нам подходят ответы, где стоят «медляки» после трека j и после трека i, и между ними нет других «медляков». (Для простоты можно считать, что плейлист начинается и заканчивается «медляком», это не влияет на ответ).

Для каждого подходящего j будем делать `dp[i] = (dp[i] + dp[j]) % MOD`.

Сумму на отрезке $[j+1; i]$ можно поддерживать и обновлять при обходе j.

Такое решение работает за $O(N^2)$.

Для полного решения оптимизируем решение второй подзадачи.

Для этого осчитаем префиксные суммы длин треков. Для каждого i при помощи бинарного поиска найдем l, начиная с которого сумма на отрезке $[l+1; i] \leqslant y$, и r, начиная с которого сумма на отрезке $[r+1; i] < x$. (Суммы на отрезке будем считать при помощи префиксных сумм). Тогда для текущего i подходят все j от l до $r-1$ включительно.

Таким образом, нам нужно к `dp[i]` прибавить все значения `dp[l] ... dp[r - 1]`. Чтобы делать это быстро, будем поддерживать еще один массив префиксных сумм — для массива `dp`.

Такое решение работает за $O(N \log N)$.

Задача 6. Цепная Реакция (9-11 классы)

Для решения 2 подзадачи можно найти все пути импульса рекурсивно, вызывая рекурсию после каждого перехода импульса.

Для решения 3 и 4 подзадач воспользуемся динамическим программированием.

Пусть $dp[i][w]$ — количество способов дойти импульсу до дома i с пропускной способностью последнего кабеля, равным w. Значение $dp[i][w]$ будем пересчитывать через все кабели с пропускной способностью w, одним из концом которых является дом i.

Однако для подзадачи 3 значения w могут быть слишком большими, поэтому «сожмем» пропускные способности, но необходимо учесть, что «сжатые» пропускные способности могут нарушить условие разницы не более чем в 2 раза, поэтому при переходе в динамике учитываем «несжатые» значения.

Для полного решения также воспользуемся «сжатием» пропускных способностей. Для каждого дома найдем список пропускных способностей всех кабелей, одним из концом которого является рассматриваемый дом. Упорядочим по возрастанию и удалим все «копии».

Обозначим через $\text{weight}[i][j]$ пропускную способность j-го кабеля в полученном списке для дома i.

Пусть теперь $\text{dp}[i][w]$ — количество способов дойти импульсу до дома i, где последняя пропускная способность кабеля меньше или равна $\text{weight}[i][w]$.

Чтобы перейти от $\text{dp}[i][w-1]$ к $dp[i][w]$, переберем все кабели с пропускной способностью $\text{weight}[i][w]$, одним из концом которых является дом i. Пусть $from$ — второй конец рассматриваемого кабеля. Количество путей через такое ребро будет равно $\text{dp}[from][x] - \text{dp}[from][y] + \text{dp}[from][0]$, где $\text{dp}[from][x]$ — количество путей импульса до дома $from$ с последней пропускной способностью,

меньшей weight$[i][w]$, а dp$[from][y]$ — количество путей импульса до дома $from$ с последней пропускной способностью, меньшей $\frac{\text{weight}[i][w]}{2}$. Значения x и y можно найти бинарным поиском, так как weight$[from]$ — отсортирован.

Таким образом, мы перебрали все варианты с меньшей последней пропускной способностью, вычли неподходящие варианты, когда последняя пропускная способность меньше более чем в 2 раза и добавили случай, когда этот кабель — первый.

Почему это будет работать быстро?

Заметим, что у нас для каждого dp$[i]$ количество элементов будет не более количества кабелей у дома i. Тогда суммарное количество элементов в dp будет не более суммарного количества кабелей у всех домов, то есть $2 \cdot M$. Таким образом, мы максимум $2 \cdot M$ раз будем вызывать по два бинарных поиска.

Задача 7. Перестановка карт (9-11 классы)

Для решения первой подгруппы можно было явно предпосчитать ответ любым способом и сохранить его в коде.

Введём обозначение p_i — позиция i-й карты.

Для решения задачи заметим, что ответ на задачу равен $\frac{\sum_{i=1}^{n} p_i - i}{2}$, так как стоимость переместить карту на своё место — $p_i - i$, и при этом перемещаются 2 карты одновременно, поэтому в сумме стоимость перемещений посчитана дважды.

Одним из способов добиться такого ответа является следующий.

Будем перебирать карты от 1 до N и будем перемещать их на свои места. Из-за порядка перебора очередная выбранная карта i никогда не должна будет перемещаться на большие позиции (так как все карты от 1 до $i-1$ находятся на своих местах на позициях от 1 до $i-1$, а значит карта i может находиться на позициях не меньше i).

Далее будем перебирать все позиции от p_i до i: если мы встретили карту с номером $j \geqslant p_i$, значит её нужно переместить правее, а значит, мы поменяем местами карты i и j. Это не ухудшит ответ, так как карта с номером j переходит в позицию с большим номером и при этом не превосходит j. Это значит, что в конечном итоге карта переместится в позицию j, и что весь её путь разобьётся на части, в сумме дающие $|q_j - j|$, где q_j — исходная позиция j, так как карта j не оказывалась левее q_j и правее j.

Таким образом, мы получили порядок перестановки, который гарантирует минимальную стоимость.

Задача 8. Загадка Алисы (9-11 классы)

Для решения первой подзадачи будем восстанавливать числа a, b в двоичной системе, начиная с младших битов. Поймем, как для каждого k (начиная с 0) найти k-ый бит чисел a и b за 3 запроса.

Пусть a_1 — значение числа, полученного из числа a взятием последних k битов. Аналогично, b_1 — значение числа, полученного из числа b взятием последних k битов.

Эти значения уже известны, так как знаем значения битов на позициях от 0 до $k-1$ чисел a, b.

Заметим, что если взять $x_1 = 2^k - a_1$, $x_2 = 2 \cdot 2^k - a_1$, $y_1 = 2^k - b_1$, $y_2 = 2 \cdot 2^k - b_1$, то все четыре числа $a + x_1, a + x_2, b + y_1, b + y_2$ будут делиться без остатка на число 2^k.

При этом заметим, что так как $x_2 - x_1 = y_2 - y_1 = 2^k$, то только одно из чисел $a + x_1, a + x_2$ будет делиться без остатка на 2^{k+1}. Аналогично, только одно из чисел $b + y_1, b + y_2$ будет делиться без остатка на 2^{k+1}.

Получается, для восстановления k-го бита остается вычислить, какое из чисел $a + x_1, a + x_2$ и какое из чисел $b + y_1, b + y_2$ делятся на 2^{k+1} без остатка.

Заметим, что НОД будет делиться на 2^{k+1} тогда и только тогда, когда оба аргумента делятся без остатка на 2^{k+1}.

При этом только среди четырех пар $(a + x_1, b + y_1), (a + x_1, b + y_2), (a + x_2, b + y_1), (a + x_2, b + y_2)$ только в одной паре оба числа делятся на 2^{k+1} без остатка.

Значит, среди четырех запросов $(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)$ только для одного запроса возвращенный результат будет нацело делиться на 2^{k+1}.

Заметим, что не нужно спрашивать все 4 пары, так как если для первых трех возвращаемый результат не делится нацело на 2^{k+1}, то по методу исключения для оставшейся пары результат будет делиться на 2^{k+1} без остатка, и четвертый запрос делать не надо.

В результате получили восстановление k-го бита чисел a, b за не более чем 3 запроса. Так как $a, b \leqslant 10^9$, то для однозначности восстановления числа достаточно восстановить младшие 30 битов (так как $2^{30} > 10^9$), в результате получим не более $30 \cdot 3 = 90$ запросов, что проходит первую подзадачу на 30 баллов.

Для полного решения воспользуемся идеей из частичного решения. Заметим, что вместо основания системы счисления 2 можно взять любое другое основание.

Пусть у нас есть основание p.

Рассуждая аналогично, получим формулы для k-го бита: $x_1 = p^k - a_1, x_2 = 2 \cdot p^k - a_1, ..., x_p = p \cdot p^k - a_1$, аналогично $y_1 = p^k - b_1, y_2 = 2 \cdot p_k - b_1, ..., y_p = p \cdot p^k - b_1$, числа a_1, b_1 определены также.

Только одно число из набора $x_1, x_2, ...x_p$ и только одно число из набора $y_1, y_2, ..., y_p$ нацело делятся на p^{k+1}.

Теперь можем составить p^2 пар, где только для одной оба числа в паре делятся на p^{k+1}, и теперь найти k-ый бит можем за не более чем $(p^2 - 1)$ запросов с использованием метода исключения.

Понятно, что вычисление k битов числа при основании p — и есть нахождение остатка при делении числа на p^k.

Поэтому можем одновременно получать информацию из запроса для разных p.

А именно: возьмем $p_1 = 2$, и для него вычислим $d_1 = 16$ младших бит, $p_2 = 3$, и для него $d_2 = 6$ младших бит и $p_3 = 5$, и для него $d_3 = 2$ младших бита.

При формировании запроса нам понадобятся числа, которые при делении на числа $2^{k_1}, 3^{k_2}, 5^{k_3}$ дают нужные нам остатки. Это можно сделать благодаря Китайской теореме об остатках, так как числа $2^{k_1}, 3^{k_2}, 5^{k_3}$ попарно взаимно просты.

Посчитаем количества запросов для каждого p:

Для $p_1 = 2$: достаточно будет $(p_1^2 - 1) \cdot d_1 = 48$ запросов.

Для $p_2 = 3$: достаточно будет $(p_2^2 - 1) \cdot d_2 = 48$ запросов.

Для $p_3 = 5$: достаточно будет $(p_3^2 - 1) \cdot d_3 = 48$ запросов.

В итоге потратили 48 запросов (так как для разных p находили остатки одновременно), что проходит ограничение в 50 запросов, и далее по Китайской теореме об остатках можно вычислить остатки при делении чисел a, b на $2^{16} \cdot 3^6 \cdot 5^2$.

Но так как $2^{16} \cdot 3^6 \cdot 5^2 > 10^9$, то числа a, b восстанавливаются однозначно, и было потрачено не более 50 запросов, что дает полное решение.

Задача 9. Начинающий астроном (7-11 классы)

Для решения подгруппы с $N = 1$ нужно найти все возможные значения a для прямоугольника. Первое, что хочется сделать — посчитать значение a для вершин многоугольника. Однако к данному решению легко подобрать контрпример. Например, взять $b = 4$ и квадрат $(1, 4)$, $(3, 4)$, $(3, 6)$, $(1, 6)$. Тогда экстремальное значение $a = 1$, в этом случае парабола касается прямоугольника в точке $(2, 4)$, а остальные значения дают пересечения с прямоугольником. Поэтому будем искать максимальное или минимальное значение a для всей стороны.

Для начала получим, что парабола проходит через точку (x, y) при $a = \dfrac{bx - y}{x^2}$ или же $a = \dfrac{b}{x} - \dfrac{y}{x^2}$.

Рассмотрим случай вертикальной стороны (когда $x_1 = x_2$), тогда заметим, что в $a = \dfrac{b}{x} - \dfrac{y}{x^2}$ значение $\frac{b}{x}$ не изменяется, а во второй значение $-\dfrac{y}{x^2}$ линейно зависит от y, а значит, минимума и максимума a достигает в вершинах стороны.

Теперь рассмотрим случай, когда сторона не вертикальная. Тогда можно представить прямую, на которой лежит сторона, в виде $\alpha x + \beta$, где $\alpha = \frac{y_2 - y_1}{x_2 - x_1}$, а $\beta = y_0 - \beta \cdot x_1$. Подставим эту формулу в формулу для a: $a = \dfrac{bx - \alpha x - \beta}{x^2} = \dfrac{(b - \alpha)}{x} - \dfrac{\beta}{x^2}$. Найдя производную по x, получаем: $a'(x) = -\dfrac{(b - \alpha)}{x^2} + 2\dfrac{\beta}{x^3}$. Приравняв к 0, получаем: $-\dfrac{(b - \alpha)}{x^2} + 2\dfrac{\beta}{x^3} = 0$, преобразуя: $2\beta - (b - \alpha)x = 0$,

отсюда $x_c = \dfrac{2\beta}{(b-\alpha)}$. Теперь, если x_c лежит между x_1 и x_2, то значение a лежит между $a(x_c)$ и $a(x_1)$ и $a(x_2)$, иначе a лежит между $a(x_1)$ и $a(x_2)$.

Альтернативно, a_c можно найти с помощью тернарных поисков.

Теперь решим задачу для больших N. Мы знаем, что для каждого прямоугольника у нас существует отрезок a, при котором парабола пересекает прямоугольник. Отметим эти отрезки (l_i, r_i). Для каждого отрезка добавим 2 пары $(l_i, 1)$ и $(r_i, -1)$ и отсортируем их по возрастанию первой координаты, а в случае равенства — по убыванию второй. Тогда пройдёмся по этому массиву и будем поддерживать сумму вторых координат — количество пересекаемых прямоугольников, а разница первых координат у соседей — количество пересечений. Таким образом, задача решается за $O(n \log n)$.