Разбор задачи «Сколько нулей?»

Автор задачи:	Михаил Иванов
Подготовка тестов и решений:	Никита Гаевой
Автор разбора:	Никита Гаевой

Отсортируем числа на входе по возрастанию b_i и просуммируем, поддерживая сумму в таком же формате. Легко видеть, что если в какой-то момент времени b_i стало строго больше, чем количество нулей в конце суммы, то ответ на задачу уже не изменится, и мы можем сразу же его вывести, не суммируя оставшиеся числа. С другой стороны, также легко видеть, что если этого не случилось, то количество ненулевых цифр в сумме не бывает больше пятнадцати (так как всего чисел не более $2 \cdot 10^5$, и каждое из них не превосходит 10^9), а значит, складывать числа можно за константное время.

Разбор задачи «Бочка и песочные часы»

Автор задачи:	Михаил Иванов
Подготовка тестов и решений:	Михаил Иванов
Автор разбора:	Михаил Иванов

Для удобства будем мерить песок в количестве секунд, которое уйдёт на то, чтобы этот песок полностью пересыпался в другую половину часов. Например, фраза «в нижней половине хотя бы t песка» значит, что если часы перевернуть, песок будет сыпаться из верхней половины (которая до переворачивания была нижней) в нижнюю не менее t секунд.

Сначала допустим, что в бочке двое песочных часов, $t_1 \leqslant t_2$. Как отмерить $t_1 + t_2$ секунд? Для этого нужно перевернуть часы, прождать t_2 секунд (то есть дождаться второго писка динамика), ещё раз перевернуть часы, прождать t_1 секунд. Почему этот способ сработает? Потому что, если мы прождали t_2 секунд, то в первых часах гарантированно за это время весь песок высыплется в нижнюю половину.

Аналогично давайте отмерим любую линейную комбинацию с целыми неотрицательными коэффициентами. Пусть надо отмерить $\sum\limits_{i=1}^{n} k_i t_i$ секунд. Решение — сначала k_n раз отмерить t_n секунд, потом k_{n-1} раз отмерить t_{n-1} секунд, и так далее. Как это сделать, и почему это будет работать?

Пусть самый большой k_i, не равный нулю — $k_m \geqslant 1$ (то есть $k_i = 0$ при $m < i \leqslant n$). Будем мерить количество песка. Будем поддерживать следующий инвариант: для каждого i в нижней половине i-х песочных часов должно быть не менее $\min\{t_i, t_m\} = t_{\min\{i,m\}}$ песка. Другими словами, в первых m часах в нижней половине весь песок, а в остальных часах в нижней половине не менее t_m песка. Заметим, что если инвариант выполняется для m, то он выполняется и для всех меньших индексов.

Как этот инвариант поможет нам? Допустим, он выполнен в текущий момент C. Перевернём часы и дождёмся m-го писка динамика (который мог произойти одновременно с несколькими другими — в частности, возможно, мы в итоге услышим более m писков). Докажем, что прошло ровно t_m секунд.

- Докажем, что к этому моменту прошло не меньше t_m секунд. Действительно, поскольку мы услышали m писков, мы услышали писк хотя бы одних часов с номером $i \geqslant m$. Но из инварианта в момент C следует, что после переворачивания в их верхней половине было не менее t_m песка, а, значит, на то, чтобы весь песок высыпался, ушло не менее t_m секунд.

- Докажем, что к этому моменту прошло не больше t_m секунд. Действительно, если мы прождём ровно t_m секунд, то в первых m часах непременно высыплется вниз весь песок (так как в каждых из них не более t_m песка), они все пропищат, и дальше точно будет бессмысленно ждать.

Нетрудно понять, что теперь инвариант для числа m выполнен: в первых m часах весь песок пересыпался, а в остальных пересыпалось вниз хотя бы t_m песка. Значит, для нового m' (которое равно m, если $k_m > 1$, и меньше m, если $k_m = 1$) инвариант также верен. Таким образом, в момент $C + t_m$ мы снова можем действовать и снова выполнен инвариант. Продолжая так делать, мы отмерим все нужные промежутки t_i в порядке убывания.

Разбор задачи «Половина информации»

Автор задачи:	Иван Казменко
Подготовка тестов и решений:	Иван Казменко
Автор разбора:	Иван Казменко

Эта задача про кодирование и декодирование отличается от других тем, что возможно не любое сопоставление исходной и закодированной строки: символы, которые не меняются на знаки вопроса, должны остаться теми же.

Решение 1: Давайте заменять на вопросики ту цифру, которая встречается чаще.

Например, в строке «`01001111`» три нуля и пять единиц, поэтому закодируем её как «`0?00????`». При декодировании вопросики — это та цифра, которой не осталось: если мы видим «`0?00????`», то понимаем, что, раз остались нули, вопросики стоят на месте единиц.

Единственная проблема в таком решении — то, что строка из всех нулей и строка из всех единиц заменяются на одну и ту же строку. Действительно, увидев код «`?????`», мы не можем понять, какой была исходная строка: «`00000`» или «`11111`».

Для нечётных n можно, например, сделать так: «`00000`» заменять на «`?????`», а «`11111`» на «`111??`»: $\lceil n/2 \rceil$ единиц и $\lfloor n/2 \rfloor$ вопросиков. Действительно, при кодировании всех других строк вопросиков будет строго больше половины, значит, такая строка не получится.

Для чётных n будем решать так же, но сначала определим, что делать при равном количестве нулей и единиц: например, будем заменять на вопросики единицы, а нули — оставлять. Тогда из «`1100`» получится «`??00`», а не «`11??`». Теперь наше правило с превращением «`1111`» в «`11??`» сработает и для чётного n, так как «`11??`» не могло получиться заменой нулей.

Решение 2: Давайте разделим задачу на маленькие части.

Сначала придумаем какое-нибудь решение при $n = 2$: например, `00` $\leftrightarrow$ `0?`, `01` $\leftrightarrow$ `??`, `10` $\leftrightarrow$ `1?`, `11` $\leftrightarrow$ `?1`.

Теперь для чётных n просто разобьём строку на пары символов, и в каждой паре решим подзадачу с $n = 2$. Например, «`01 00 11 11`» превратится в «`?? 0? ?1 ?1`». Получится, что вопросиков не меньше половины, так как их не меньше половины в каждой паре.

Для нечётных n сделаем то же самое, а последний символ оставим без изменений. Поскольку требуемое количество вопросиков округляется вниз, вопросиков в парах будет достаточно.

Разбор задачи «Красивые множества»

Автор задачи:	Никита Гаевой
Подготовка тестов и решений:	Никита Гаевой
Автор разбора:	Никита Гаевой

Заметим, что сумма чисел в условии равна $(n+1)!$. Этот факт легко доказывается по индукции, а также это можно было получить, вычислив несколько значений суммы при маленьких значениях n и угадав формулу. Наивного решения, вычисляющего факториал по модулю, было достаточно, чтобы пройти все группы тестов, кроме последних двух. Для того, чтобы пройти предпоследнюю группу, достаточно было выполнить предподсчёт факториалов с шагом в 10^7. Для того, чтобы пройти и последнюю группу, нужно было дополнительно применить теорему Вильсона, из которой следует, что $998\,244\,352!$ сравним с -1 по модулю $998\,244\,353$. Теорема Вильсона позволяет легко учесть все множители, не кратные $998\,244\,353$.

К счастью, для того, чтобы учесть все остальные, достаточно решить такую же задачу вычисления количества конечных нулей и последней ненулевой цифры числа $\lfloor \frac{n+1}{998\,244\,353} \rfloor !$, что можно сделать рекурсивно.

Разбор задачи «n станков»

Автор задачи:	Михаил Иванов
Подготовка тестов и решений:	Михаил Иванов
Автор разбора:	Михаил Иванов

Чтобы решить первые две группы, достаточно было исполнять каждый запрос за линейное время. Чтобы сделать d_i преобразований из деталей a_i в детали b_i, нужно для каждого j от $a_i + 1$ до b_i совершить не менее d_i преобразований из детали $j - 1$ в деталь j. Хотелось бы сказать, что для этого надо j-й станок арендовать на $\frac{d_i}{v_j}$ дней, но в общем случае d_i не делится на v_j, поэтому на самом деле надо арендовать станок на $\left\lceil \frac{d_i}{v_j} \right\rceil$ дней. Итак, надо просто посчитать

$$\sum_{j=a_i+1}^{b_i} \left\lceil \frac{d_i}{v_j} \right\rceil.$$

Чтобы получить полный балл, предстояло это ускорить. Рассмотрим каждое слагаемое $\left\lceil \frac{d_i}{v_j} \right\rceil$ как функцию от d_i. Как она себя ведёт при замене $d_i - 1$ на d_i? Нетрудно понять, что следующим образом: если d_i сравнимо с единицей по модулю v_j, то возрастает на единицу, а если не сравнимо, то не изменяется. При $d_i = 0$ функция нулевая. Таким образом, можно рассмотреть строку, в которой на позициях, сравнимых с единицей по модулю v_j, стоит единица, а на остальных стоит ноль, и требуемое число — количество единичек на позициях с нулевой по d_i-ю. Назовём эту строку *аддитивным массивом*.

Воспользуемся корневой декомпозицией. Обозначим максимальное возможное d_i буквой D. Зафиксируем некоторое n_0 и рассмотрим два вида v_j — не превосходящие n_0 и превосходящие. Станки первого вида назовём *частыми*, второго — *редкими*. Сначала разберёмся с частыми станками. Для каждого $i \leqslant n_0$ построим структуру данных, позволяющую за $\mathcal{O}(1)$ на отрезке массива $\{v_j\}_{i \in \{1,\ldots,n\}}$ находить количество элементов, равных i (префиксные суммы). Теперь в каждом запросе те станки, у которых $v_j \leqslant n_0$, мы можем обработать за $\mathcal{O}(n_0)$: переберём i, найдём от $a + 1$ до b число s_i станков с $v_j = i$ и добавим к ответу $\left\lceil \frac{d}{i} \right\rceil$. Суммарно эти операции займут $\mathcal{O}(qn_0)$ времени.

Теперь разберёмся с редкими станками. Они хороши тем, что в у каждого из них в аддитивном массиве не более $\left\lceil \frac{D}{n_0} \right\rceil$ единичек. Будем постепенно увеличивать b и для каждого его значения строить дерево Фенвика, поддерживающее массив, равный поэлементной сумме аддитивных массивов всех редких станков с первого по b-й, и позволяющее находить префиксную сумму этого массива. Чтобы перейти от b к $b+1$, надо либо ничего не сделать (если станок частый), либо сделать $\left\lceil \frac{D}{n_0} \right\rceil$ запросов к дереву Фенвика, в каждом из которых добавляется единица в некоторый элемент. Такой проход займёт $\mathcal{O}(\frac{nD \log D}{n_0})$ времени (логарифм на операции с деревом Фенвика).

Как тогда отвечать на запросы? Воспользуемся тем, что все запросы известны заранее (то есть поступают в offline). Тогда для ответа на запрос (a_i, b_i, d_i) мы должны в момент, когда мы храним дерево Фенвика в состоянии a_i, вычесть из ответа ans_i d_i-ю префиксную сумму, а когда мы храним дерево Фенвика в состоянии b_i, добавить d_i-ю префиксную сумму. Тогда мы в итоге $2q$ раз обратимся к дереву Фенвика и потратим на это $\mathcal{O}(q \log D)$ времени.

Таким образом, получилась асимптотика $\mathcal{O}\left(qn_0 + q \log D + \frac{nD \log D}{n_0}\right)$. Оптимально подобрать такое n_0, чтобы первое и третье слагаемое были равны с точностью до константы. Для этого возьмём $n_0 = \Theta\left(\sqrt{\frac{nD \log D}{q}}\right)$ и получим асимптотику $\mathcal{O}\left(q \log D + \sqrt{nqD \log D}\right)$.

Разбор задачи «Сколько равных?»

Автор задачи:	Владислав Макаров
Подготовка тестов и решений:	Владислав Макаров, Никита Гаевой
Автор разбора:	Владислав Макаров

В разборе будет описано решение жюри на полный балл (возможно, есть какие-то существенно другие решения). Для того, чтобы прийти к этому решению, нужно было сделать несколько последовательных наблюдений, каждое из которых приводит к решению на большее количество баллов.

1 Решение на 31 балл

Каждый хороший массив $x = [x_0, x_1, \ldots, x_n]$ задаётся своим массивом разностей $d = [d_1, d_2, \ldots, d_n]$, где $d_i = x_i - x_{i-1} \in [\ell_i, r_i]$ для каждого i от 1 до n. Более того, для $0 \leqslant i < j \leqslant n$, условие $x_i = x_j$ эквивалентно тому, что $0 = x_j - x_i = (x_j - x_{j-1}) + (x_{j-1} - x_{j-2}) + \ldots + (x_{i+1} - x_i) = d_j + d_{j-1} + \ldots + d_{i+1}$.

Заметим, что это условие зависит только от d-шек с номерами от $i+1$ до j. Пусть есть хороший массив, в котором элементы на позициях $0 \leqslant i_1 < i_2 < \ldots < i_k \leqslant n$ равны (то есть $x_{i_1} = x_{i_2} = \ldots = x_{i_k}$). То, равны ли x_{i_1} и x_{i_2} или нет, зависит только от того, как мы выбрали d-шки с номерами от $i_1 + 1$ до i_2. Аналогично, равны ли x_{i_2} и x_{i_3} или нет, зависит только от d-шек с номерами от $i_2 + 1$ до i_3. И так далее. Важно, что все эти отрезки d-шек попарно не пересекаются.

Получается, что условия $x_{i_1} = x_{i_2}$, $x_{i_2} = x_{i_3}$, …, $x_{i_{k-1}} = x_{i_k}$ в каком-то смысле «независимы»: если можно выполнить каждое из них по отдельности, то можно выполнить и все вместе.

Давайте построим такой граф: вершинами будут числа от 0 до n, а ориентированное ребро из вершины i в вершину j будет тогда и только тогда, когда существует хороший массив, в котором $x_i = x_j$. Несложно видеть, что нас интересует длина самого длинного пути в таком графе.

Когда может выполниться условие $x_i = x_j$? В точности когда $d_{i+1} + d_{i+2} + \ldots + d_j$ может равняться 0. С другой стороны, $d_{i+1} + d_{i+2} + \ldots + d_j$ — произвольное целое число из отрезка $[\ell_{i+1} + \ell_{i+2} + \ldots + \ell_j, r_{i+1} + r_{i+2} + \ldots + r_j]$.

Следовательно, нужно проверить, содержит ли этот отрезок число 0. Если зафиксировать j и рассматривать все $i < j$ в порядке убывания, то границы этого отрезка поддерживать легко. Пусть dp_i — длина наибольшего пути, заканчивающегося в вершине i. Она пересчитывается вот так: dp_j — это максимум $\mathrm{dp}_i + 1$ по всем таким $i < j$, что в графе есть ребро $i \to j$; если же таких i вообще нет, то 1. Получили решение за $O(n^2)$ времени и $O(n)$ памяти.

2 Решение на 68 баллов

Давайте немного упростим предыдущее решение. А именно, пусть $L_i := \ell_1 + \ell_2 + \ldots + \ell_i$, $R_i := r_1 + r_2 + \ldots + r_i$ для всех $0 \leqslant i \leqslant n$. Тогда наличие ребра $i \to j$ в графе эквивалентно тому, что отрезок $[\ell_{i+1} + \ell_{i+2} + \ldots + \ell_j, r_{i+1} + r_{i+2} + \ldots + r_j] = [L_j - L_i, R_j - R_i]$ содержит число 0. Теперь нам уже не важен порядок, в котором мы перебираем i: всё равно условие проверяется за $O(1)$.

Но давайте пойдём ещё чуть дальше. Что означает, что отрезок $[L_j - L_i, R_j - R_i]$ содержит 0? Это означает, что $L_j - L_i \leqslant 0 \leqslant R_j - R_i$, то есть $L_j \leqslant L_i$ и $R_i \leqslant R_j$. Иными словами, отрезок $[L_i, R_i]$ содержится в отрезке $[L_j, R_j]$!

Таким образом, dp_j — это максимум $\mathrm{dp}_i + 1$ по таким i, что отрезок $[L_i, R_i]$ содержится в $[L_j, R_j]$. Отсюда получается решение за $O(n \log^2 n)$: идём по j в порядке возрастания и храним двумерную структуру данных на максимум, делаем запрос максимума на прямоугольнике

$[L_j, +\infty) \times (-\infty, R_j]$, с помощью него понимаем значение dp_j, после чего обновляем структуру данных в точке (L_j, R_j) значением dp_j.

Поскольку все отрезки $[L_j, R_j]$ известны заранее, то можно заранее сжать все координаты (это не самая простая, но стандартная техника; её объяснение выходит за рамки данного разбора). В зависимости от конкретных используемых структур и аккуратности реализации такое решение должно получать от 68 до 100 баллов.

3 Решение на 100 баллов

Предыдущее решение использует двумерные структуры данных и работает за $O(n \log^2 n)$. Авторское решение не использует никаких структур данных, работает за $O(n \log n)$ и реализуется очень просто. Чтобы к нему прийти, нужно сделать ещё одно наблюдение.

На самом деле мы хотим найти самую длинную последовательность отрезков $[L_{i_k}, R_{i_k}]$, в которой каждый отрезок вложен в следующий, а индексы i_k возрастают. Оказывается, что от второго условия можно избавиться. Действительно, длины отрезков $[L_i, R_i]$ возрастают с увеличением i, при этом даже строго (из-за условия $\ell_i < r_i$). Поэтому, если один отрезок вложен в другой, то у него автоматически меньший индекс.

Давайте поэтому отсортируем отрезки по возрастанию R (а при равенстве R — по убыванию L). Тогда, если один отрезок вложен в другой, то больший отрезок идёт в этом порядке позже. На набор вложенных отрезков должно выполняться два условия: нестрогое убывание L-ок и нестрогое возрастание R-ок. Второе условие выполняется автоматически (мы так отсортировали). Поэтому, в таком порядке сортировки, наборы вложенных отрезков соответствуют в точности нестрого убывающим последовательностям L-ок.

Итак, мы свели нашу к задачу к известной задаче о наибольшей возрастающей подпоследовательности (с точностью до знака и строгости/нестрогости сравнений). Она решается либо с помощью сжатия координат и одномерного дерева отрезков, либо с помощью двоичного поиска без каких-либо структур данных вообще. Второе решение особенно приятно в реализации и заходит в ограничения по времени с почти шестикратным запасом.

Интуитивно можно сказать, что произошло следующее: у нас была «трёхмерная» задача про отрезки $[L_i, R_i]$ (измерениями были левые границы L, правые границы R и сами индексы i). В таких задачах обычно можно «избавиться» от одного из измерений с помощью сканирующей прямой. В предыдущем решении мы так избавлялись от i. Однако, оказалось, что это третье измерение не независимо от двух других, а очень строго с ними связано. Поэтому от него можно избавиться «бесплатно», а ещё от одного измерения (в данном случае, R-ок) — с помощью сканирующей прямой. Здесь использовалась специфика задачи (длины отрезков возрастают с увеличением i); для произвольных отрезков $[L_i, R_i]$, скорее всего, нет никакого «одномерного» решения.