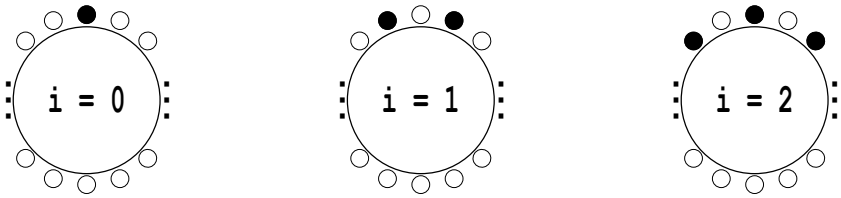
Разбор задачи «Большой круглый стол»

Автор задачи:	Григорий Шовкопляс
Подготовка условия, решения и тестов:	Григорий Шовкопляс
Автор разбора:	Григорий Шовкопляс

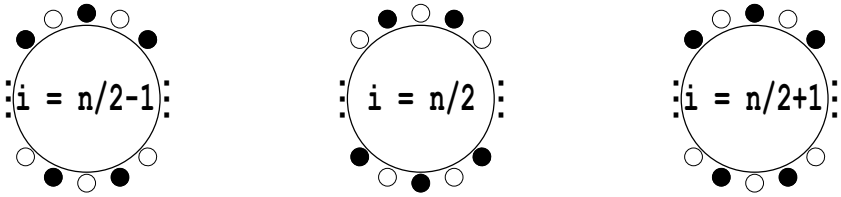
Пусть i существенно меньше n. Рассмотрим возможные перемещения Маши, на картинке черным отмечены позиции, где она могла оказаться после того, как она поменялась местами с соседом i раз.



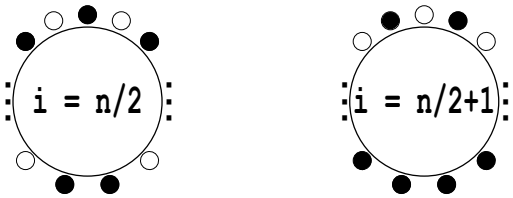
Легко видеть, что число различных мест равно $i + 1$.

Однако когда i достигнет значения $n/2$, возможные позиции Маши, если она перемещалась вокруг стола по часовой стрелке и против часовой стрелки, соответственно, «встретятся», и нужно изучить, что же произойдет, после значения $i = n/2$.

Пусть, для начала, n четно. Тогда после после того, как Маша поменяется местами с соседом $n/2$ раз, количество различных мест перестанет увеличиваться.



Если n нечетно, то количество различных мест будет возрастать вплоть до n.



Таким образом искомое число мест равно $\min(n/2, k + 1)$, если n четно, и $\min(n, k + 1)$, если n нечетно.

Разбор задачи «Помеха справа»

Автор задачи:	Юрий Петров
Подготовка условия, решения и тестов:	Михаил Ютман, Антон Гардер
Автор разбора:	Михаил Ютман

Будем рассматривать машины в порядке возрастания времени, параллельно поддерживая четыре очереди из машин, едущих соответственно вверх, вправо, влево и вниз. Поддерживаем также счетчик, который отвечает за то, какой сейчас момент времени.

Сначала добавляем все машины, которые подъезжают к перекрестку в текущий момент времени, в соответствующие очереди. Теперь для каждой очереди проверяем, может ли оттуда уехать первая машина. Из очереди может уехать машина, если очередь справа от нее пустая. После того, как мы узнали, из каких очередей могли уехать машины, достанем из этих очередей машины и запомним для них ответ. Заметим, что как видно из второго примера в условии, иногда перекресток могут одновременно проехать две машины во встречных направлениях.

Если после этого есть хотя бы одна непустая очередь, то перейдем к моменту времени, большему на 1. Иначе переходим к моменту времени, в который подъезжает следующая машина, если еще остались машины, либо завершаем цикл по интересным моментам и выводим ответ.

Если в текущий момент времени из очереди никто не уехал, но при этом есть непустая очередь, то это значит, что в каждой очереди есть хотя бы одна машина. Значит дальше ни одна из машин, которые сейчас стоят в очереди, и те, которые еще не были добавлены, никогда не проедут перекресток, поэтому запомним для этих машин, что они не смогут проехать, выйдем из цикла и выведем ответ.

Итоговая сложность решения $O(n)$, так как на каждой итерации цикла мы либо добавляем машину в очередь, либо удаляем машину из очереди.

Разбор задачи «Древнее заклинание»

Автор задачи:	Николай Будин
Подготовка условия, решения и тестов:	Николай Будин
Автор разбора:	Николай Будин

Давайте рассмотрим граф с $n \cdot m \cdot l$ вершинами. Будем обозначать вершину тройкой (x, y, z) $(1 \leqslant x \leqslant n;\ 1 \leqslant y \leqslant m;\ 1 \leqslant z \leqslant l)$, она будет соответствовать одновременно ячейке таблицы (x, y) и z-му символу заклинания.

Рассмотрим теперь, как построить ребра. Обозначим как $f[x][y]$ символ в ячейке (x, y) таблицы, а $s[z]$ — символ заклинания на позиции z. Пусть ячейки (x_1, y_1) и (x_2, y_2) являются соседними по стороне в таблице. Тогда проведем ориентированное ребро из (x_1, y_1, z) в $(x_2, y_2, (z \bmod l) + 1)$, если $f[x_1][y_1] = s[z]$ и $f[x_2][y_2] = s[(z \bmod l) + 1]$.

Несложно заметить, что любой цикл в таком графе соответствует искомой последовательности. Осталось найти цикл или выяснить, что его нет. Это можно сделать с помощью обхода в глубину за время, пропорциональное сумме числа вершин и ребер нашего графа, то есть $O(nml)$.

Сделаем напоследок несколько технических замечаний. Поскольку в нашем графе в худщем случае около 8 миллионов вершин и 32 миллионов ребер, то явно строить граф и хранить списками смежности все ребра может не получиться из-за ограничений по памяти. Даже если аккуратно реализовать хранение ребер, размер графа будет существенно превышать размер кеша процессора. Работа с таким графом будет происходит медленно из-за постоянных промахов кеша, поэтому возможно превышение ограничения по времени. Эффективнее во время выполнения обхода в глубину, находясь в вершине (x, y, z), перебирать 4 соседние клетки и проверять, можно ли в них перейти.

Разбор задачи «Эскалатор»

Автор задачи:	Демид Кучеренко
Подготовка условия, решения и тестов:	Михаил Путилин
Автор разбора:	Михаил Путилин

Сначала посчитаем, сколько суммарно цифр в числах от 1 до n, кратных десяти.

Переберём количество цифр в числе от 2 до длины числа n. Посчитаем cnt_k — сколько k-значных чисел принадлежат отрезку от 1 до n, k-значные числа лежат на отрезке $[10^{k-1}; 10^k - 1]$. Если $n \geqslant 10^k$, то все они они не больше n, и $cnt_k = 9 \cdot 10^{k-1}$. В противном случае нас интересуют только числа на отрезке $[10^{k-1}; n]$, и $cnt_k = n + 1 - 10^{k-1}$.

При $k \geqslant 2$ среди k-значных чисел на десять делится каждое десятое число, начиная с наименьшего. Поэтому таких чисел $\lfloor cnt_k/10 \rfloor$, и суммарно в них $\lfloor cnt_k/10 \rfloor \cdot k$ цифр. Здесь $\lfloor x \rfloor$ означает округленное вниз число x.

К суммарному числу цифр в числах, кратных десяти, нужно добавить единицу —длину числа 1, и, если n не кратно десяти, длину числа n. Кроме того, нужно не забыть про случай $n = 1$ — тогда ответ равен 1.

Наконец, необходимо использовать 64-битный тип данных, причем как для подсчета ответа, так и для чтения из ввода значения x.

Разбор задачи «Сериал»

Автор задачи:	Михаил Путилин
Подготовка условия, решения и тестов:	Михаил Путилин
Автор разбора:	Михаил Путилин

Отсортируем размеры файлов по неубыванию, пусть $s_i \leqslant s_{i+1}$. Размеры файлов — $s_1, \cdots, s_n$, для удобства дальнейших рассуждений положим $s_0 = 0$.

Пусть запросы — $x_1, \ldots, x_t$, обозначим как y_i их частичные суммы: $y_i = x_1 + \cdots + x_i$ $(0 \leqslant i \leqslant t)$. При этом $y_0 = 0$, так как мы должны скачать все файлы целиком, а скачивать больше, чем размер наибольшего файла, бессмысленно, то $y_t = s_n$.

Заметим, что имеет смысл делать запросы только так, чтобы после каждого запроса суммарное число загруженных байт было равно размеру какого-то файла. Иными словами, если запросы — $x_1, \ldots, x_t$, то для любого i от 0 до t должно выполняться $y_i = s_w$ для некоторого w.

Доказательство: предположим обратное. Пусть $s_w < y_i < s_{w+1}$, при этом возьмём наименьшее такое i. Тогда можно присвоить y_i значение s_w, уменьшив соответствующим образом x_i и увеличив x_{i+1}. Ответ от этого не ухудшится, так как теперь при загрузке файлов размера s_w в последнем пакете не будет лишних байт, а для остальных файлов ничего не изменится. Если в результате окажется, что y_i совпало с y_{i-1}, то y_i нужно будет вообще убрать.

Будем теперь решать задачу методом динамического программирования. Пусть есть последовательность запросов суммарного размера s_i. Тогда dp_i — это наименьший возможный суммарный размер пакетов, которые будет загружены, если скачивать при помощи этой последовательности все серии. При этом файлы размера не больше s_i будут скачаны целиком, а от остальных — только первые s_i байт.

Начальное значение: $dp_0 = 0$, так как $s_0 = 0$. Ответом будет dp_n.

Как вычислить dp_i: переберём последний запрос x_t. Как следует из изложенного ранее, он должен быть таким, что $s_i - x_t = s_w$ для некоторого w. Поэтому будем перебирать w от 0 до $i - 1$. Первые $t - 1$ запросов скачают s_w байт, потратив на это dp_w. Последний запрос будет выполнен только для файлов размера больше s_w, и для каждого из них будет скачано $x_t + k$. Значит, всего будет скачано $dp_w + cnt_w \cdot (s_i - s_w + k)$. dp_i — это минимум этих значений по всем w.

Время работы решения — $O(n^2)$. Заметим, что его можно далее соптимизировать до времени $O(n \log n)$ с использованием Convex Hull Trick, но при заданных ограничениях этого не требуется.

Разбор задачи «Парадокс с дробями»

Автор задачи:	Никита Михайлов
Автор условия:	Андрей Станкевич
Подготовка решения и тестов:	Никита Михайлов
Автор разбора:	Никита Михайлов

Для начала упорядочим дроби в порядке неубывания. Это можно сделать любым алгоритмом сортировки за время $O(k^2)$ или $O(k \log k)$. Будем называть медианой дробей a/b и c/d дробь $\frac{a+c}{b+d}$.

Заведём двумерный массив B, где $B_{i,j}$ — медиана дробей a_i и a_j. Теперь, для того чтобы найти ответ на задачу, нужно найти две такие ячейки этого массива $B_{a,c}$, $B_{b,d}$, для которых верны следующие условия:

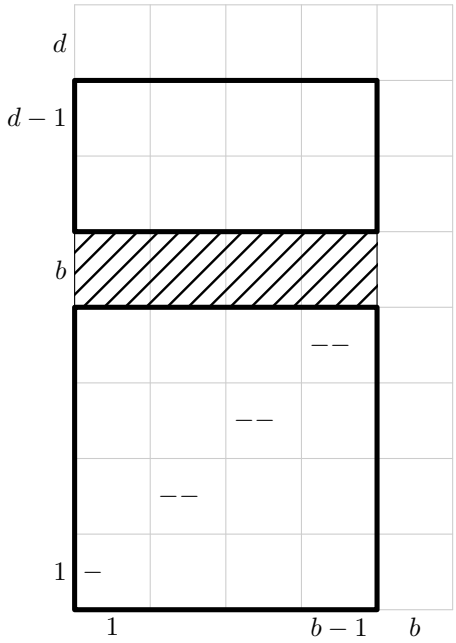
- $a < b$
- $c < d$
- a, b, c и d — различные числа
- $B_{a,c} - B_{b,d}$ максимально

Переберём ячейки массива (b, d), для каждой из них найдём такую подходящую клетку (a, c), в которой $B_{a,c}$ максимально. Будем, не ограничивая общности, считать, что $b < d$, так как всегда можно одновременно поменять местами b и d, а также a и c.

На первый взгляд для клетки (b, d) искомой клеткой будет минимум на прямоугольнике с угловыми клетками $(1, 1)$ и $(b - 1, d - 1)$. Однако эта клетка не всегда подходит, поскольку a, b, c и d должны быть различными числами, поэтому ячейки, для которых $a = c$ или $c = b$ не следует учитывать при поиске минимума.

Избавиться от ячеек, где $a = c$, легко, достаточно положить для них $B_{a,c} = -\infty$.

Чтобы избавиться от ячеек, для которых $b = c$ необходимо искать минимум не на одном, а на двух прямоугольниках: с угловыми клетками $(1, 1)$ и $(b - 1, b - 1)$, а также $(1, b + 1)$ и $(b - 1, d - 1)$ (второй из них может отсутствовать, если $b = d$ или $b = d - 1$).



Будем перебирать ячейки по возрастанию b, а при равных b по возрастанию d. В каждой строке будем хранить максимум m_i среди значений $B_{1,i}, B_{2,i}, \ldots, B_{b-1,i}$. При переходе от d к $d' = d + 1$ максимум на двух описанных прямоугольниках необходимо обновить значением m_d. При переходе от b к следующему значению $b' = b + 1$ все значения m_i следует обновить значением $B_{b,i}$, а искомый максимум сделать равным максимуму значений $m_1, m_2, \ldots, m_b$.

Итоговая сложность решения по времени: $O(k^2)$, по памяти: $O(k^2)$.

Разбор задачи «Головоломка»

Автор задачи:	Антон Гардер
Подготовка условия, решения и тестов:	Дмитрий Филиппов
Автор разбора:	Дмитрий Филиппов

Переберем перемещение фигуры над препятствием, а затем рассмотрим процесс опускания фигуры. Возможны два подхода, оба из которых приводят к решению за время порядка $O(s^3)$, где s — верхняя граница на размеры фигуры и препятствия.

Первый подход: будем опускать фигуру вниз по одной клетке. Если какая-то клетка фигуры совпала с какой-то клеткой препятствия, или если все клетки фигуры стали ниже всех клеток препятствия, опускание надо прекратить. При проверке, что некоторая клетка фигуры совпала с клеткой препятствия, достаточно проверять только нижнюю клетку каждого столбца. Когда процесс опускания завершен, поднимем фигуру обратно на одну клетку и посчитаем количество очков, набранных Димой, обновим ответ.

Оценим время работы при этом подходе. Возможных перемещений — $O(m_f + m_o)$, количество итераций опускания — $O(n_f + n_o)$. Проверка пересечений с препятствием на каждой итерации —

$O(m_f)$, так как мы проверяем только самые нижние клетки в каждом столбце. Подсчет очков после опускания — $O(n_f \cdot m_f)$. Итого: $O((m_f + m_o) \cdot ((n_f + n_o) \cdot m_f + n_f \cdot m_f))$, что соответствует $O(s^3)$.

Второй подход: для каждого столбца j найдем самую высокую клетку препятствия в нем — $upper_j$. Теперь посчитаем сразу количество итераций опускания фигуры как минимум по всем клеткам фигуры (i, j) значений $(i - upper_j - 1)$. Опустим все клетки фигуры на эту величину, затем посчитаем полученной Димой количество очков и обновим ответ.

Время работы второго подхода — $O((m_f + m_o) \cdot (n_o \cdot m_o + n_f \cdot m_f)$, что также является $O(s^3)$.

Разбор задачи «Кто хочет стать миллионером?»

Автор задачи и условия	Андрей Станкевич
Подготовка решения и тестов:	Никита Михайлов
Автор разбора:	Артем Васильев

Будем составлять искомый список чисел последовательно. Предположим, что мы уже сгенерировали n чисел, поймем, чему будет равно a_{n+1}. С одной стороны, $a_{n+1} \geqslant 2 \cdot a_n$. С другой стороны, a_{n+1} должно заканчиваться на x нулей, где x — половина длины десятичной записи a_{n+1}, округленная вверх. Это условие равносильно тому, что a_{n+1} делится на 10^x.

Заметим, что длина числа $2 \cdot a_n$ не более, чем на один больше, чем длина числа a_n. Если длина не изменилась, или же длина изменилась с нечетного числа на четное, то, поскольку a_n делилось на 10^x, то и $2 \cdot a_n$ будет делиться на 10^x, и минимальным числом, подходящим под условие задачи, будет $a_{n+1} = 2 \cdot a_n$.

В противном случае, необходимо увеличить число до того, пока оно не станет делиться на 10^x. С предыдущего шага мы знаем, что a_n делится на 10^{x-1}. Поэтому, для того, чтобы получить число, делящееся на 10^x, можно прибавлять 10^{x-1}, пока текущее число не делится на 10^x. Таких прибавлений будет не больше десяти. В конце мы получим число, которое больше, чем два предыдущих, и делится на 10^x. Таким образом, начиная с первого числа $a_1 = 100$, мы можем получить все нужные n чисел.

Разбор задачи «Операция «Перестановка»»

Автор задачи:	Антон Гардер
Подготовка условия, решения и тестов:	Василий Алферов
Автор разбора:	Василий Алферов

Сделаем двоичный поиск по ответу — количеству утверждений, после которых можно однозначно восстанавливать исходную перестановку. Двоичный поиск можно применить, так как если перестановку можно восстановить после x утверждений, то тем более её можно однозначно восстановить после $x + 1$ утверждения.

Посмотрим, как проверить, что после k утверждений можно восстановить требуемую перестановку. Построим граф с n вершинами, где i-я вершина — это позиция солдата в перестановке. Пусть известно, что номер солдата, стоявшего в a-й позиции больше номера солдата, стоявшего в b-й позиции. Тогда проведём ориентированное ребро из вершины b в вершину a. Сделаем так для первых k утверждений.

Заметим, что граф получится ациклическим даже для $k = m$, так как по условию гарантируется, что существует хотя бы одна перестановка, для которой выполняются все условия, которые помнит адъютант. У ациклического графа есть топологическая сортировка — такой порядок вершин, что ребра идут только из вершин с меньшими номерами в вершины с большими. Топологическую сортировку графа с V вершинами и E ребрами можно построить с помощью обхода в глубину за время $O(V + E)$.

Заметим, что есть взаимно-однозначное соответствие между топологическими сотрировками получившегося графа и подходящими построениями содлат. Действительно, пусть задана топологическая сортировка, тогда для обратной перестановки выполняются все условия: если на a-й позиции должен стоять солдат с большим номером, чем на b-й, то из b в a проведено ребро, значит, b в

обратной перестановке находится раньше, а значит, номер солдата, стоящего на b-м месте меньше. Обратно, если у нас есть какой-нибудь ответ, обратная ему перестановка соответствует какой-то топологической сортировке графа. Итак, для того, чтобы проверить единственность возможного построения, необходимо и достаточно проверить единственность топологической сортировки в этом графе.

Это можно сделать следующим образом. Построим какую-нибудь топологическую сортировку и проверим, что между каждой парой соседних вершин в ней есть ребро. Если это условие выполняется, то никакой другой топологической сортировки быть не может. Обратно, если топологическая сортировка единственна, то между каждой парой соседних вершин есть ребро, иначе можно поменять местами пару соседних вершин, между которыми ребра нет, и получить другую топологическую сортировку.

Сложность такого решения $O((n+m)\log m)$, оно укладывается в ограничения по времени.

Однако можно усовершенствовать это решение и получить решение за $O(n+m)$. Заметим, что если ответ не равен -1, то в графе, построенном на всех запросах, топологическая сортировка единственна. Заметим также, что если она в какой-то момент стала единственной, то при добавлении ребер уже не поменяется. Значит, достаточно топологически отсортировать итоговый граф и взять в качестве ответа максимальный номер ребра между соседними вершинами.

Разбор задачи «Трудности переписки»

Автор задачи и условия:	Демид Кучеренко
Подготовка решения и тестов:	Станислав Наумов
Автор разбора:	Станислав Наумов

Рассмотрим, как может измениться строка при нажатие кнопки «Home». Давайте представим s как сумму строк $c_1 + c_2 + \cdots + c_{k-1} + c_k$, где операция «+» — это запись одной строки в конце другой. Тогда если курсор сдвигался на начало строки как раз после написания каждой из строк $c_1, c_2, \ldots, c_k$, мы получим строку t, равную $c_k + c_{k-1} + \cdots + c_2 + c_1$.

Для решения задачи воспользуемся динамическим программированием. Пусть dp_i — логическое значение: верно ли, что из первых i символов строки s можно получить последние i символов строки t в нужном порядке. Тогда $dp_0 = True$. А ответ находится в dp_n: если $dp_n = True$, то из s можно получить t, иначе нельзя. Научимся пересчитывать значения в массиве dp. Переберем i от 0 до n и будем обновлять значение «вперед». Пусть мы можем набрать первые i символов в нужном порядке, значит, значение $dp_i = True$. Попробуем добавить еще j символов, это возможно, когда соответствующие подстроки $s[i+1..i+j]$ и $t[n-i-j+2..n-i+1]$ равны. В этом случае устанавливаем значение dp_{i+j} в $True$.

Если сравнивать строки, проверяя на равенство каждый соответствующий символ двух строк, то решение будет работать за время порядка n^3 и не уложится в ограничения по времени. Для оптимизации сравнения строк можно воспользоваться хешами, z-функцией или прекфикс-функцией, тогда две подстроки будут сравниваться за $O(1)$ и итоговое решение будет работать за $O(n^2)$.

Разбор задачи «Таня, мячи и «исключающее или»»

Авторы задачи:	Владислав Подтёлкин, Сергей Копелиович
Подготовка условия, решения и тестов:	Станислав Наумов
Автор разбора:	Станислав Наумов

В задаче необходимо посчитать сумму «исключающих или» нескольких пар чисел, эта функция считается по битам независимо. Посчитаем для каждого бита, какой вклад он вносит в итоговую сумму.

Пусть c_i — это количество чисел от 1 до n, где i-й бит равен единице, соответственно $n - c_i$ — это количество чисел, где этот бит равен нулю. Функция $\oplus$ от двух битов принимает значение 1, если эти биты различны. Поэтому количество слагаемых в итоговой сумме, где i-й бит равен единице,

ровно $c_i \cdot (n - c_i)$. Тогда, если k — количество битов в двоичной записи числа n, то ответ на задачу — это $\sum_{i=0}^{k} 2^i \cdot c_i \cdot (n - c_i)$

Научимся быстро считать c_i. Рассмотрим числа в порядке возрастания от 0 до n и посмотрим только на i-й бит. В первых 2^i числах этот бит принимает значение, равное 0, в следующих 2^i — значение, равное 1, и так далее. Нетрудно заметить, что период равен 2^{i+1}. Теперь легко посчитать c_i, для этого рассмотрим, сколько раз пройдет полный период, и учтем остаток.