Задача A. Яичница

Для решения этой задачи требуется вывести $X - Y$.

Однако в этой задаче есть две вспомогательные группы тестов, позволяющие решать ее поэтапно.

Для прохождения 1-й группы тестов можно не считывать вводимые числа вовсе. Достаточно написать вывод числа 7 — ответа на задачу при $X = 7, Y = 0$.

Во 2-й группе тестов $Y = 0$, поэтому ответ равен $X - 0 = X$. Таким образом, для прохождения второй группы тестов достаточно считать только число X и вывести его.

Критерии оценивания:

№	Баллы	Ограничения		Необх. группы
		X	Y	
1	17	$X = 7$	$Y = 0$	—
2	34	$X \leqslant 50$	$Y = 0$	1
3	49	$X \leqslant 50$	$Y < X$	1, 2

Баллы выставляются автоматически проверяющей системой.

Задача B. Сумма двух

Решение задачи предполагает разбор всех случаев. Для этого требуется сравнить D с тремя числами: $A + B$, $B + C$ и $A + C$. Если D равно хотя бы одному из них, то ответ — «`YES`», иначе — «`NO`».

Для решения на частичный балл заметим, что в первых трёх группах тестов сумма оставшихся чисел не зависит от того, какое число будет выбрано для стирания, так как числа A, B и C равны. Поэтому достаточно единственной проверки на равенство чисел $2A$ и D.

В первой группе тестов эту проверку можно осуществить на листочке: $2A = 10$, а $D = 11$. Поэтому достаточно написать программу, которая выводит «`NO`».

Для прохождения второй группы тестов нужно считать число D. Так как число D вводится последним, до этого необходимо считать и остальные числа. Однако известно, что $A = B = C = 5$, поэтому D достаточно сравнить с константой 10.

Для прохождения третьей группы тестов D нужно сравнить с числом $2A$.

В четвёртой группе тестов сумма двух оставшихся чисел после стирания уже зависит от выбранного числа. Но так как $A = B$, достаточно только разобрать два случая из трёх: если стереть число A и если стереть число C.

Критерии оценивания:

№	Баллы	Ограничения		Необх. группы
		D	Дополнительно	
1	7	$D = 11$	$A = B = C = 5$	—
2	38	$D \leqslant 300$	$A = B = C = 5$	1
3	16	$D \leqslant 300$	$A = B = C$	1–2
4	29	$D \leqslant 300$	$A = B$	1–3
5	10	$D \leqslant 300$	—	1–4

Баллы выставляются автоматически проверяющей системой.

Задача C. Бодрые квартиры

Для прохождения 1-й группы тестов достаточно проверить, включен ли свет в единственном окне.

Во 2-й группе тестов в доме только одна квартира, но ее длина равна n. Поэтому требуется посчитать количество окон с выключенным светом во всем доме.

Такое решение можно реализовать при помощи прохода по строке циклом `for`. Для этого заведём вспомогательную переменную — количество окон, в которых свет не горит. К ней в теле цикла будем

прибавлять единицу за каждое окно с выключенным светом, то есть за каждый элемент строки, равный символу «.». Отметим, что строку с цветом окон нужно считать, но при этом решение ее никак не использует.

Во многих языках программирования эта группа решается при помощи встроенных функций. Такую реализацию можно использовать, в том числе и как подзадачу для решения следующих групп. Однако далее будет рассматриваться именно реализация через цикл.

Отметим, что можно считать во вспомогательной переменной как количество окон, в которых не горит свет, так и число окон, в которых свет горит. В этом решении от этого зависит только условие, определяющее ответ: 0 или 1. В одном случае, нужно проверить, что во вспомогательной переменной после цикла лежит значение 0, а в другом — n.

Переходя к следующей группе, заметим, что дом с несколькими квартирами можно искусственно разбить на несколько домов с одной квартирой и посчитать ответ на задачу как сумму ответов по всем домам, на которые был разбит исходный дом.

Таким образом, для решения 3-й группы тестов следует отдельно рассмотреть случай, когда в доме две квартиры. Обнаружить, что в доме две квартиры (если гарантируется, что их не больше, чем две), можно сравнением цвета первого и последнего окна. Если эти цвета не отличаются, то квартира одна, и можно запустить решение для 2-й группы тестов.

В противном случае, найдем при помощи цикла или встроенной функции место, где начинается вторая квартира. Для этого можно найти самое левое окно, цвет которого отличается от цвета первого окна, то есть можно найти символ в строке цветов окон, не равный первому символу в этой строке — это будет первое окно второй квартиры. Далее можно явно разбить каждую из введенных строк на две по этому индексу, тем самым разбить исходный дом на два новых, а затем вставить решение второй группы тестов для каждого из них, объединив ответы при помощи суммы.

Отметим, что эту группу можно решить и не создавая новых строк, а, например, выделив два диапазона индексов для циклов по первой и второй части дома: от первого окна до места смены цвета окон и от места смены цвета окон до конца дома. Тело цикла при этом остается таким же, как и в решении второй группы тестов.

Идея с разбиением обобщается для любого числа квартир. Однако реализовать явное разбиение строки на несколько подобным образом довольно трудно. Поэтому посмотрим, как можно проводить такое разбиение неявно, то есть не создавая новых строк.

Опытные участники скорее всего знают, что подобное неявное разбиение можно реализовать при помощи обнуления вспомогательных переменных в тех местах, где разбиение происходит. Таким образом, первый вариант полного решения задачи состоит в следующем. Сначала нужно завести переменную, в которую суммируются ответы по каждой квартире (прибавляется 1, если в рассмотренной квартире во всех окнах горит свет), причем обновление ответа осуществляется через вспомогательную переменную, почти как в решении второй группы тестов, небольшие отличия обозначим позже. Однако обновление ответа должно запускаться не только в самом конце, но и при перемещении из одной квартиры в другую, то есть при смене цвета окон. Помимо этого нужно обнулять счётчик числа окон с включенным (или выключенным) светом, тем самым запускать подсчёт заново при перемещении из квартиры в квартиру.

Отличие условия обновления ответа касается только реализации, в которой во вспомогательной переменной считается количество окон с горящим светом — значение в счётчике нужно сравнивать не с n, а с длиной квартиры, которую можно посчитать, например, при помощи другой вспомогательной переменной. При этом условие, если счётчик считает количество окон с выключенным светом, остается прежним.

Если такая идея незнакома, то к похожему решению можно прийти через следующие группы тестов. При этом саму идею, что писать явное разбиение не следует, подсказывает 4-я группа тестов, в которой у всех квартир ровно одно окно. Но тогда количество квартир, в которых горит свет во всех окнах, в точности равно числу окон, в которых горит свет, а это то, что считает вспомогательная переменная в решении для 2-й группы (или то, что можно вывести простой формулой, если вспомогательная переменная считает число окон с выключенным светом). Таким образом, для решения 4-й группы тестов нужно только изменить вывод ответа в решении для 2-й группы.

Однако что в таком случае будет считать вспомогательная переменная? Очевидно, что это сам

ответ на задачу. Тогда кажется, что раз старый счётчик больше не участвует в реализации, то и вообще как таковой подсчет количества окон с включенным светом отсутствует. Но он, конечно, есть, в неявном и очень упрощенном виде — в проверке света на каждой итерации цикла.

Давайте увидим это в 5-й группе тестов. Когда некоторые квартиры могут состоять из двух окон, решение, очевидно, меняют эти самые квартиры из двух окон. Тогда просто рассмотрим их отдельно. Обнаружить их можно при помощи аккуратного условия: если на текущей итерации окно i не последнее и при этом цвет окна с номером $i + 1$ совпадает с i-м, то на текущей итерации рассматривается левое окно квартиры с двумя окнами. В таком случае к переменной, считающей ответ, нужно прибавлять единицу, только если в обоих окнах горит свет. Нужно при этом пропустить следующую итерацию цикла, так как при стандартной реализации программа может посчитать, что она находится в очередной квартире с одним окном. В некоторых языках программирования для этого достаточно увеличить переменную цикла на 1. Там, где это сделать нельзя, можно написать аккуратное условие: если на текущей итерации окно i не первое и при этом цвет окна с номером $i-1$ совпадает с i-м, то на текущей итерации рассматривается правое окно квартиры с двумя окнами, а значит нужно эту итерацию пропустить.

Видно, что подсчёт числа окон с горящим светом спрятан в проверке света у квартир с двумя окнами. При этом такой подсчёт запускается на каждой итерации цикла, кроме итераций с пропуском, заново.

Как тогда можно изменить решение, чтобы получить полный балл? Условие заранее задать нельзя, так как длины квартир могут быть любыми. Но вспомним, что решение 2-й группы тестов работает для любой длины квартиры. Притом в решении 3-й группы тестов обнаружено, что разбиение, то есть точка, где заканчивается подсчёт, происходит при смене цвета окон.

Тогда давайте запускать подсчёт числа окон с включенным (или выключенным) светом по каждой квартире явно — примерно так, как это делалось в решении второй группы. Для этого нужно написать вложенный цикл, который считает число окон с включенным (или с выключенным) светом до перехода в другую квартиру, то есть до смены цвета окон, либо до конца строки, если цикл был запущен в последней квартире. Такой цикл, очень похожий на решение 2-й группы тестов, заменит условные операторы, которые проверяли свет в окнах в решении 5-й группы тестов. Затем напишем условие, по которому будет обновляться ответ — оно либо остается таким же, как в решении 2-й группы, либо в нем меняется сравнение с числом n на сравнение с длиной квартиры (см. первый вариант полного решения). Остается только пропустить следующие окна уже рассмотренной квартиры, что можно сделать аналогично тому, как это делалось в 5-й группе тестов; проще всего оставить проверку цвета предыдущего окна — она работает для квартир любой длины.

В этой задаче $N \leqslant 10^5$, что ограничивало использование функций внутри цикла (чтобы не допустить квадратичного времени работы программы — когда на строки длины N требуется совершить число операций, сравнимое с N^2), и неявно подсказывало, что задача, скорее всего, может быть решена единственным циклом с какими-то условиями внутри. Однако в качестве второго варианта полного решения предлагается написать вложенный цикл, что обычно свидетельствует о квадратичном времени работы программы. Почему тогда это должно работать?

Спасает от квадратичного времени работы именно пропуск окон той квартиры, ответ для которой уже посчитан. Действительно, тогда вложенный цикл будет запускаться нечасто, причем важно то, что за все его запуски он обойдет все введенные символы только один раз, так как он считает включенный или выключенный свет по каждой квартире ровно один раз. Как следствие заметим, что возможный вердикт «`Time limit exceeded`» мог быть вызван именно отсутствием пропуска уже просмотренных квартир, перегоняя по приоритету ожидаемый вердикт «`Wrong answer`».

Критерии оценивания:

№	Баллы	Дополнительные ограничения	Необх. группы
1	12	$N = 1$	—
2	13	В доме ровно одна квартира	1
3	16	В доме не больше двух квартир	1 – 2
4	14	У каждой квартиры ровно одно окно	1
5	18	У каждой квартиры не более двух окон	1, 4
6	27	Без дополнительных ограничений	1 – 5

Баллы выставляются автоматически проверяющей системой.

Задача D. Грузовики

В 1-й группе тестов ответ всегда t_1, так как единственный грузовик может за один заезд перевести весь груз.

Для прохождения групп тестов 2 – 3 можно вывести формулу, по которой можно посчитать, сколько рейсов нужно совершить единственному грузовику.

Другой вариант: сэмулировать передвижение единственного грузовика при помощи цикла. Для прохождения второй группы тестов подойдет практически любая эмуляция, например, цикл по моментам времени и хранение во вспомогательной переменной расстояния, которое грузовик проехал от пункта A. Однако оптимальнее за одну итерацию цикла эмулировать рейс грузовика: если еще остался груз в пункте A, забрать w_1 груза и изменить счётчик времени на t_1, после чего вернуться, если в пункте A остался груз, увеличив счётчик времени еще на t_1. Этот алгоритм работает быстро, так как грузовик гарантированно перевезет весь груз за 50 000 рейсов (этот случай достигается при $W = 50\,000$ и $w_1 = 1$), а значит и цикл совершит не более 50 000 итераций. Для прохождения третьей группы тестов остается только хранить затраченное время в 64-битном целом числе.

4-ю группу тестов можно свести к предыдущей заменой всех грузовиков на один единственный, просуммировав их грузоподъёмности. Это возможно, так как все t_i равны и грузовики выгодно отправлять одновременно, не допуская простой.

В 5-й и 6-й группах тестов грузовики ездят с разной скоростью, однако гарантируется, что самый медленный грузовик приедет в пункт B раньше, чем самый быстрый вернется в пункт A. Кроме того, в этой группе грузоподъёмность такова, что если все грузовики совершат по одному рейсу, то весь груз будет доставлен. Из этого вытекает, что каждый грузовик нужно отправить не более, чем в один рейс. Но остается вопрос, какие именно грузовики следует отправить.

Заметим, что нужно отправлять грузовики в порядке сортировки по времени перевозки, начиная с самых быстрых. Это обосновывается тем, что ответ, то есть затраченное время на всю перевозку, — это максимум из времени, затраченного на перевозку, по всем грузовикам. Тогда необходимость в рейсах некоторого грузовика за время T есть только тогда, когда груз невозможно перевезти за время T', что $T' < T$. Чтобы гарантировать этот факт при отправке очередного грузовика, нужно отправить сначала все те грузовики, которые доставят очередную часть груза до пункта B раньше него. Но так как в этой подзадаче все грузовики выполняют только один рейс, такое условие можно гарантировать именно обработкой в порядке сортировки по времени перевозки.

Тогда отсортируем грузовики по времени t_i и отправим их ровно столько, сколько нужно для перевозки всего груза. Для этого при помощи цикла найдем наименьшее k, что $w_1 + w_2 + \ldots + w_k \geqslant W$, причем обратите внимание, что подразумеваются индексы уже после сортировки грузовиков по времени перевозки. Тогда нужно отправить именно грузовики до k-го по быстроте, и тем самым ответ на задачу равен t_k. Если сортировку реализовать при помощи квадратичных сортировок, таких как пузырьковая или вставками, то решение пройдет только 5-ю группу. Если же использовать более быстрые сортировки (обычно они встроены в стандартные библиотеки языков программирования) и записывать ответ в 64-битное целое число, то решение пройдет и тесты из 6-й группы.

В следующих группах грузовики могут совершить несколько рейсов. Однако идея остается той же, но немного меняется способ реализации: нужно сортировать не сами грузовики, а моменты прибытия грузовиков в пункт B на их возможных рейсах.

Когда n мало такие моменты можно сгенерировать заранее, так как их потребуется не больше, чем 50 000 на каждый грузовик. Таким образом, создадим массив моментов прибытия, в который

для каждой машины с характеристиками (w_i, t_i) положим соответствующие прибытия, характеризующиеся временем и грузом: (w_i, t_i), $(w_i, 3 \cdot t_i)$, …, $(w_i, 99\,999 \cdot t_i)$. Ответ будет считаться так же, как и в решении 5-й группы тестов. Однако массив, который нужно отсортировать, получается довольно большим, из-за чего даже быстрая сортировка может работать довольно долго.

Ключевая идея: генерировать не все окончания рейсов сразу, а добавлять их только после того, как грузовик совершит очередной рейс. Тогда изначально массив окончаний рейсов выглядит как в решении 5-й группы тестов — массив грузовиков, отсортированный по времени перевозки. Однако после первых отправлений в него добавляются новые окончания рейсов, которые нужно протолкнуть на нужное место, чтобы сохранить сортировку по времени.

Существует множество способов реализовать такое проталкивание и получить баллы за какие-то из групп 7, 8, 9 и 10. Например, когда $N \leqslant 100$, можно удалять уже выполненные перевозки, и тем самым поддерживать длину массива моментов до 100, что позволяет выполнять вставку и удаление довольно быстро. Или если $t_i \leqslant 5$, то можно объединять рейсы с равным временем прибытия в один большой, как это делалось в 4-й группе тестов, и тем самым рассмотреть не более $2 \cdot \max(t_1, t_2, \ldots, t_n) \cdot W \leqslant 500\,000$ моментов.

Полное решение требует эффективной реализации добавления, что реализует структура данных «куча», реализованная в C++ как `set`, а в Python как `heapq`. Эта структура данных позволяет быстро находить минимальный элемент и добавлять новый — это именно то, что требуется для решения этой задачи.

Однако в основном замечании в качестве обоснования фигурирует T — ответ на задачу. Это наталкивает на другой способ решения задачи — при помощи перебора ответа.

Предположим, что мы располагаем временем T, в течение которого можем отправить каждый грузовик в некоторое количество рейсов (возможно, ни разу). Обозначим за k_i количество рейсов, которое успеет выполнить i-й грузовик за время T. Тогда всего будет совершено $2 \cdot k_i$ перемещений между пунктами, а значит выполняется неравенство: $(2k_i - 1) \cdot t_i \leqslant T$, что равносильно $k_i \leqslant \frac{T+t_i}{2t_i}$. То есть максимальное возможное количество рейсов i-го грузовика $k_i = \lfloor \frac{T+t_i}{2t_i} \rfloor$, за эти рейсы он перевезёт $\lfloor \frac{T+t_i}{2t_i} \rfloor \cdot w_i$ груза.

Весь груз возможно перевести за время T, если $\sum\limits_i \lfloor \frac{T+t_i}{2t_i} \rfloor \cdot w_i \geqslant W$. Ответом задачи является наименьшее целое значение T, для которого выполняется данное неравенство.

Если $t_i \leqslant 5$, то ответ не больше 500 000, а значит для прохождения 7-й группы значение T можно просто перебрать, однако может и потребоваться оптимизация — не пересчитывать сумму перевезённого груза каждый раз заново, а хранить моменты, когда значение суммы по конкретному грузовику увеличивается — это можно реализовать подобно тому, как написано предыдущее решение.

9-ю группу тестов можно решить при помощи усовершенствования алгоритма, решающего 3-ю подзадачу. Заметим, что эмуляция — это заодно и перебор ответа. Тогда если запустить эмуляцию по обеим машинам последовательно, но внутри каждой эмуляции считать ответ для машины, движение которой не эмулируется, по формуле, то в итоге будут перебраны все моменты времени, в которые какая-либо часть груза доставляется в пункт B, и ответ может быть выбран как минимум из подходящих моментов.

Для полного решения задачи нужно перебирать T при помощи бинарного поиска. Коротко опишем этот алгоритм.

Двоичный поиск отличается от полного перебора тем, что он проверяет не все возможные значения T, а выбирает очередное T для проверки на основе полученных результатов. Например, если для $T = T_1$ груз доставить невозможно, то в дальнейшем алгоритм будет рассматривать только $T > T_1$. Можно гарантировать, что ответ лежит в диапазоне от 1 до $\max(t_1, t_2, \ldots, t_n) \cdot W \leqslant 10^{14}$, из-за чего можно каждый раз выбирать очередное T для проверки как середину текущего отрезка, где может лежать ответ, тем самым сокращая его после каждой проверки в два раза. Так, бинарный поиск найдет искомое T примерно за 50 проверок.

Критерии оценивания:

Обозначим $t_{\min} = \min(t_1, t_2, \ldots, t_n)$, $t_{\max} = \max(t_1, t_2, \ldots, t_n)$, $\sum w = w_1 + w_2 + \ldots + w_n$.

№	Баллы	Ограничения				Необх. группы
		N	t_i	W	Дополнительно	
1	4	$N = 1$	$t_i \leqslant 5$	$W = w_1$	—	—
2	11	$N = 1$	$t_i \leqslant 5$	—	—	1
3	5	$N = 1$	$t_i \leqslant 10^9$	—	—	1–2
4	10	$N \leqslant 50\,000$	$t_i \leqslant 10^9$	—	$t_1 = t_2 = \ldots = t_n$	1–3
5	20	$N \leqslant 100$	$t_i \leqslant 5$	$W \leqslant \sum w$	$2 \cdot t_{\min} > t_{\max}$	1–2
6	5	$N \leqslant 50\,000$	$t_i \leqslant 10^9$	$W \leqslant \sum w$	$2 \cdot t_{\min} > t_{\max}$	1–5
7	12	$N \leqslant 100$	$t_i \leqslant 5$	—	—	1–2, 5
8	10	$N \leqslant 50\,000$	$t_i \leqslant 5$	—	—	1–2, 5, 7
9	8	$N \leqslant 2$	$t_i \leqslant 10^9$	—	—	1–3
10	6	$N \leqslant 100$	$t_i \leqslant 10^9$	—	—	1–3, 5, 7, 9
11	9	$N \leqslant 50\,000$	$t_i \leqslant 10^9$	—	—	1–10

Баллы выставляются автоматически проверяющей системой.

Задача E. Почти сумма цифр

Попробуем сначала понять, что в целом происходит в задаче. Для этого можно решить 1-ю группу тестов, причем для её решения требуются только знания условных операторов. Действительно, все возможные тесты можно перебрать на листочке (их всего 10) и решить каждый из них как математическую задачу, например, полным перебором. Аналогично можно решить и следующие две группы тестов.

Однако как только становится ясным, что происходит в задаче и как написать полный перебор, следует реализовать его любым способом для прохождения групп 1–3. Например, функцию $s(x)$ можно реализовать просто через условный оператор: если $x \geqslant 10$, вычесть 9. Эту процедуру не обязательно реализовывать отдельной функцией.

Решение начнем с перебора чисел l и r при помощи двух вложенных циклов, причем внутренний должен начать перебор числа r с $r = l$. Затем переберем все числа из отрезка $[l; r]$ еще одним вложенным циклом, и перемножим их между собой (для этого потребуется вспомогательная переменная; обратите внимание, что в отличие от суммы ее начальное значение равно 1, а не 0), а затем проверим делимость на 9 при помощи взятия остатка от деления на 9. Если произведение не поделилось, то к ответу нужно прибавить единицу за найденный отрезок. Так как результат не превосходит $s(1) \cdot s(2) \cdot \ldots \cdot s(12) = 2177280$, хранить вычисления можно в стандартном 32-битном целом числе.

Далее необходимо сделать несколько оптимизаций этого решения. Чтобы пройти 4-ю группу, заметим, что текущий полный перебор делает много лишней работы. На итерации с числами (l, r), где $r > l$ число $s(l) \cdot s(l+1) \cdot \ldots \cdot s(r)$ будет посчитано циклом при помощи не менее, чем $r - l$ операций умножения. Однако на прошлой итерации (для чисел $(l, r-1)$) уже было посчитано число $s(l) \cdot s(l+1) \cdot \ldots \cdot s(r-1)$, и для получения требуемого его нужно только умножить на $s(r)$.

Пусть вспомогательная переменная будет сохранять свое значение между итерациями цикла, перебирающего r (для этого ее можно, например, инициализировать в теле внешнего цикла), тогда в его теле можно заменить цикл на единственную операцию — домножение вспомогательной переменной на $s(r)$, и вычисления будут корректными.

Изменим вычисление функции $s(x)$ — поменяем в текущей реализации условный оператор на цикл `while`.

Однако у такого решения есть несколько проблем, которые нужно решить. Основная из них — переполнение вспомогательной переменной, причем от этого не спасает даже 64-битное целое число.

Первый вариант решения этой проблемы работает в группах с $K = 2$. Заметим, что на отрезке последовательных чисел обязательно встретится число, делящееся на 9, а функция $s()$ не меняет делимость на 9 (действительно, $x - 9k$ и x дают одинаковый остаток от деления на 9). Значит, можно заканчивать вложенный перебор на числе $\min(l + 8, B)$. Такая идея позволяет пройти 4-ю и 6-ю группы тестов.

Но ключевая идея заключается в хранении не всего произведения, а только вхождение степени тройки в произведение: пусть $s(l) \cdot s(l+1) \cdot \ldots \cdot s(r) = 3^k \cdot C$, тогда будем хранить только число k. Тогда программа легко обобщается для любого K: перебор следует завершать при $k \geqslant K$. Так, можно решить $4 - -6$ группы тестов.

Решение 7-й группы тестов — вопрос аккуратной реализации идей и небольших оптимизаций. Однако на некоторых языках программирования стоит сразу перейти к 8-й группе тестов. Для ее прохождения заметим, что можно применить метод двух указателей: начинать перебор r не с l, а с момента, где закончился перебор на прошлой итерации перебора l. Это возможно, так как если отрезок $[l; r]$ подходит, то отрезок $[l+1; r]$ тоже гарантированно подходит.

Для решения оставшихся подзадач заметим, что если B очень большое, то количество отрезков на позиции l и на позиции $l+9$ одинаково. Это связано с тем, что $s(x)$ сохраняет вхождения степени тройки до второй, а более большие обрубает тоже до второй. Таким образом, степень вхождения тройки в числа $s(l), s(l+1), \ldots, s(r)$ соответственно совпадает с $s(l+9), s(l+10), \ldots, s(r+9)$.

Так, можно посчитать ответ только для левых границ, равных первым 9 числам, а для остальных посчитать его на основе уже посчитанных. Посчитать для первых 9 чисел в группах тестов 9 и 10 можно реализовать при помощи циклов, а для прохождения 11-й группы тестов эту границу нужно искать при помощи бинарного поиска.

Посчитать остальные отрезки можно при помощи умножения полученных результатов на количество возможных левых границ отрезков с заданным остатком от деления на 9: тем самым мы перенсим все подходящие отрезки $[l; r]$ на отрезок $[l+9z; r+9z]$ для некоторого z. Остается только посчитать, сколько отрезков тем самым вышли за B. Однако для каждого из таких остатков это можно сделать при помощи суммы арифметической прогрессии: если за границы выходят $t > 0$ отрезков с началом в l, то среди отрезков с началом в $l+9$ выходят за пределы $t+9$ отрезков. Так, просуммируем эти значения при помощи формулы для суммы арифметической прогрессии с шагом 9.

Критерии оценивания:

№	Баллы	Ограничения		Необх. группы
		K	B	
1	8	$K = 2$	$B \leqslant 4$	—
2	12	$K = 2$	$B \leqslant 8$	1
3	16	$K = 2$	$B \leqslant 12$	1 – 2
4	12	$K = 2$	$B \leqslant 2000$	1 – 3
5	6	$K \leqslant 100$	$B \leqslant 2000$	1 – 4
6	10	$K = 2$	$B \leqslant 150\,000$	1 – 4
7	6	$K \leqslant 100$	$B \leqslant 150\,000$	1 – 6
8	8	$K \leqslant 10^5$	$B \leqslant 150\,000$	1 – 7
9	10	$K = 2$	$B \leqslant 2 \cdot 10^9$	1 – 4, 6
10	8	$K \leqslant 10^5$	$B \leqslant 2 \cdot 10^9$	1 – 9
11	4	$K \leqslant 2 \cdot 10^9$	$B \leqslant 2 \cdot 10^9$	1 – 10

Баллы выставляются автоматически проверяющей системой.