Общие замечания по всем задачам

Наборы задач для 7-8 и 9-11 классов пересекаются. Всего было 7 задач, ниже разобраны они все. Задачи распределены по классам следующим образом:

	7-8 классы	9-11 классы
Акции	A	A
Точное деление	B	
Бочонок и ведро	C	B
Порядок	D	C
Остатки	E	
Сброс		D
Волшебный квадрат		E

«Бесплатные» баллы за задачи

На олимпиадах такого формата нужно стремиться к тому, чтобы набрать как можно больше баллов на каждой из задач. Начисление баллов за отдельные тесты позволяет писать частичные решения задач и зарабатывать баллы на них.

Если у вас совсем нет идеи решения (даже частичного), за задачу всё равно можно получить ненулевое количество баллов, обрабатывая только данные из примеров в условии (то есть написав программу, которая умеет распознавать данные из примеров и выводить ответы из примеров). Вот какое количество баллов можно было набрать для каждой из задач таким способом, совсем не решая саму задачу:

Акции	4
Точное деление	2
Бочонок и ведро	8
Порядок	2
Остатки	2
Сброс	2
Волшебный квадрат	2

Кроме того, этим трюком можно дополнить существующее у вас частичное решение. Отметим, однако, что на соревнованиях более высокого уровня тесты из условия могут не входить в тестовый набор (если это так, то об этом написано в правилах олимпиады или условиях задач).

Также обращайте внимание на особенные ответы, которые могут повторяться во множестве тестов к задаче. Например в задаче Остатки можно попробовать посылать всегда ответ 0, а в задаче Волшебный квадрат — ответ -1 (или, ещё лучше, не всегда, а для всех данных, кроме данных из условия), и на этом тоже получить какие-то баллы.

O-нотация

В разборах ниже используется O-нотация, то есть записи вроде «сложность алгоритма составляет *O(n)*». Если вы ещё не знакомы с O-нотацией, то можно считать это эквивалентным записи «количество операций в алгоритме пропорционально n».

Акции

Автор задачи: Виктор Соловьёв

Нужно удвоить полученное на вход число. При $n \leqslant 10^5$ и входное число и ответ помещаются в знаковый 16-битный тип. Для решения задачи на полный балл нужно использовать по крайней мере 32-битный тип целых чисел.

Сложность алгоритма — *O(1).*

Точное деление

Автор задачи: Данил Мячин

Ответ составляет половину от числа во входных данных. Иными словами, в знаменатели дроби до её сокращения всегда находится число 2, а сократима она только в том случае, если числитель чётен.

Таким образом, для нечётных чисел ответ имеет вид

$$n/2,$$

а для чётных

$$m/1,$$

где m — половина n.

Нужно уметь выводить запись ответа в одну строку без лишних пробелов. В частности в python для этого нужно или отменять перевод строки при использовании нескольких вызовов print, или использовать подстановку значения в форматированную строку.

Для решения задачи на максимальный балл необходимо использовать по крайней мере 64-битный тип целых чисел. 32-битные типы (которые в большинстве языков являются стандартными для целых чисел, как int в C++) гарантируют набор только 80 баллов.

Сложность алгоритма — *O(1).*

Бочонок и ведро

Автор задачи: Виктор Соловьёв

Крайние значения отношения объёма ведра к объёму бочонка достигаются в следующих случаях:

1. когда n вёдер едва хватает для заполнения резервуара, а m-1 бочонка почти хватает (а из последнего, m-го бочонка, используется одна бесконечно малая капля);
2. когда n-1 вёдер почти хватает, а m бочонков хватает впритык.

Рассмотрим первый случай. Обозначим объём ведра как $V_{в}$, а объём бочонка как $V_{б}$. Тогда

$$nV_{в} = (m - 1)V_{б},$$

$$\frac{V_{в}}{V_{б}} = \frac{m-1}{n}.$$

Аналогично, для второго случая получим

$$\frac{V_{в}}{V_{б}} = \frac{m}{n-1}.$$

Нетрудно заметить, что первое из этих значений является минимальным, а второе — максимальным.

Для решения задачи на максимальный балл необходимо уметь выводить вещественные числа с точностью не менее чем до шести знаков после десятичной точки (чтобы выполнить условие на погрешность в ответе).

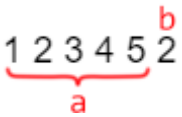
Сложность алгоритма — *O(1).*

Порядок

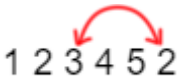
Автор задачи: Виктор Соловьёв

Решение для строго возрастающей последовательности

Нетрудно заметить, что для строго возрастающей последовательности ответ равен количеству чисел в последовательности *a*, больших, чем *b*. Рассмотрим это на первом примере:



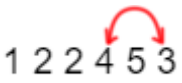
Поставим *b* на правильное место в последовательности *a*. Для этого поменяем его с первым числом, которое больше *b*:

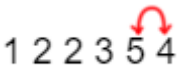


Получим:

1 2 2 4 5 3

Теперь уже число 3 нужно поставить на правильное место в последовательности, причём это место на 1 позицию ближе к концу последовательности, чем предыдущее. Продолжим обмены, пока не отсортируем последовательность:





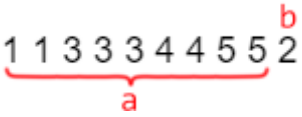
1 2 2 3 4 5

Таким образом, достаточно посчитать количество чисел в последовательности *a*, значение которых превосходит *b*.

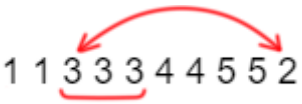
Сложность этого алгоритма — *O(n)*.

Полное решение

Рассмотрим такой пример:



Поставим число *b* на своё место. Обратите внимание, что при этом мы меняем его с первым из нескольких чисел с минимальным значением, большим чем b:



1 1 2 3 3 4 4 5 5 3

Теперь нет смысла менять тройку с другими тройками, можно поменять её сразу с первой четвёркой в группе четвёрок:

Ответом будет не количество чисел, больших чем *b*, а количество уникальных значений, больших чем *b*. Посчитать их всё ещё можно за *O(n)*.

Встроенные алгоритмы удаления дубликатов из последовательности, такие как std::unique в C++, позволяют свести общее решение к решению для строго возрастающей последовательности.

Остатки

Автор задачи: Виктор Соловьёв

Заметим, что для всех чисел *a* с одинаковым остатком от деления на *m* будет одинаковым и остаток от деления a^n на *m*. Иначе говоря, достаточно рассмотреть только числа от 0 до $m - 1$, потому что все последующие числа будут повторять их поведение при возведении в степень *n* и взятии остатка.

Частичное решение при n, m ⩽ 10

В первой группе тестов (при n, m ⩽ 10) можно провести вычисление максимально просто: возвести каждое число от 0 до $m - 1$ в степень *n* (используя умножение в цикле) и потом уже взять остаток от деления на *m*. Все вычисления поместятся в 64-битные целочисленные типы или в 32-битный беззнаковый (максимальное значение, которое нужно будет вычислить, это 9^{10}. Сложность такого решения — *O(nm)*.

Частичное решение при n, m ⩽ 1000

Нетрудно доказать, что остаток произведения равен остатку произведения остатков, то есть

$$(a \cdot b)\%m = (a\%m \cdot b\%m)\%m,$$

где % — операция взятия остатка. Иными словами если в конце нам нужен только остаток от деления a^n на m, то и все промежуточные вычисления мы можем делать с остатками.

Улучшим предыдущее решение следующим образом: вычисляя a^n многократным умножением на *a* в цикле, будем после каждого умножения подменять результат его остатком от деления на *m.* Результат от этого не изменится, но все промежуточные вычисления теперь гарантированно помещаются в 64-битные целочисленные типы.

Сложность такого решения — всё ещё *O(nm)*.

Полное решение быстрым возведением в степень

Воспользуемся алгоритмом быстрого возведения в степень. Используя соотношения

$a^n = (a^{n/2})^2$, если n — чётное,

$a^n = a \cdot a^{n-1}$, иначе,

можно вычислять степень числа за *O(log n)* вместо *O(n)*.

Продолжим заменять промежуточные вычисления на каждом шагу их остатками от деления на *m*, чтобы избежать переполнений.

Сложность такого решения — *O(m log n)*.

В языке python встроенной функции pow можно передать три аргумента, чтобы вычислить остаток от деления a^n на *m* за *O(log n)*. В языках, где нет такой встроенной функции, её придётся написать самостоятельно.

Сброс

Автор задачи: Виктор Соловьёв

Задача решается методом динамического программирования, и решения будут описаны в его терминах («состояния», «переходы», «мемоизация»).

Частичное решение при $k = n$

Если количество повторяющихся ходов не ограничено, то на каждом ходу можно использовать любой вариант.

В этом случае задачу можно решать одномерной динамикой. Единственным параметром состояния будет количество карт на руках, а хранимым значением — минимальное количество ходов для достижения этого состояния. Изначально нам известно, что *n* карт на руках можно иметь, сделав *0* ходов, а про любое другое количество карт мы не знаем ничего (можем пока считать, что требуется бесконечное число ходов для достижения этих состояний).

Из каждого состояния можно сделать до *m* различных переходов. Если у нас на руках *x* карт, вариант хода *i* может быть использован (то есть $a_i \leq x$) и

$$d_x + 1 < d_{x-a_i},$$

где d_x — минимальное число ходов для достижения состояния с *x* картами на руках, то ответ для состояния с числом карт $x - a_i$ может быть улучшен до меньшего значения:

$$d_{x-a_i} = d_x + 1.$$

Обратим внимание на то, что переходы можно делать только из достижимых состояний (то есть таких, в которые мы уже умеем попадать за конечное число ходов).

Всего потребуется рассмотреть $n + 1$ состояний (0 карт, одна карта,..., n карт) и из каждого попытаться сделать *m* переходов (если решение написано не рекурсивно, то можно рассматривать состояния в порядке убывания числа карт). После рассмотрения всех состояний, ответ на вопрос задачи будет находиться в d_0.

Сложность этого алгоритма — *O(nm),* затраты по памяти — *O(n).*

Частичное решение при $k = 1$

Если $k = 1$, то каждый ход кроме первого должен отличаться от предыдущего. Значит, для совершения переходов из состояния, нам теперь нужно знать, каким был предыдущий ход. Сделаем номер последнего использованного варианта вторым параметром состояния. При совершении перехода теперь нельзя использовать вариант, номер которого равен номеру в текущем состоянии. Выбирая вариант *i* для

перехода мы обновляем состояние, в котором последний использованный вариант это *i*.

Количество различных состояний (и затраты по памяти) при этом увеличатся в *m* раз, количество различных переходов из одного состояния всё ещё *m*.

Сложность этого алгоритма — $O(nm^2)$, затраты по памяти — *O(nm).*

Полное решение

Пусть *k* может отличаться и от 1 и от *n.* Тогда, для совершения перехода, нам необходимо знать не только какой вариант был использован последним, но и сколько раз подряд он уже был использован. Это число (меняющееся в пределах от 0 до *k*) станет третьим параметром состояний.

Нельзя использовать *i*-й вариант для перехода, если он был последним использованным и уже использован *k* раз. При переходе в состояние с тем же вариантом, счётчик его использований увеличивается на 1, в противном случае — сбрасывается до значения 1 (последний вариант использован в последнем переходе, но не в переходе перед ним).

Количество состояний увеличилось в $k + 1$ раз, количество переходов из одного состояния не поменялось.

Сложность этого алгоритма — $O(nm^2k)$, затраты по памяти — *O(nmk).*

Это решение укладывается в ограничения по времени и памяти и проходит все тесты.

Оптимизация памяти в полном решении до *O(nm)*

Не будем хранить количество использований последнего варианта. Вместо этого, делая переходы из какого-то состояния, попробуем сразу использовать каждый вариант от *1* до *k* раз подряд, то есть поменять значение карт на руке не только на a_i, но и на $2a_i$, $3a_i$,...,ka_i.

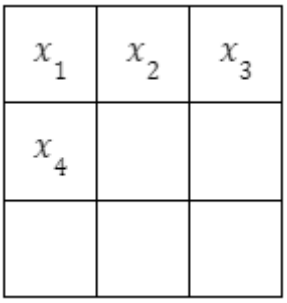
Состояний в этом случае будет (n+1)m, как во втором неполном решении, но количество различных переходов из одного состояния теперь mk.

Таким образом, сложность алгоритма всё равно составит $O(nm^2k)$, но памяти потребуется всего *O(nm)*,

Волшебный квадрат

Авторы задачи: Виктор Соловьёв, Александр Пак

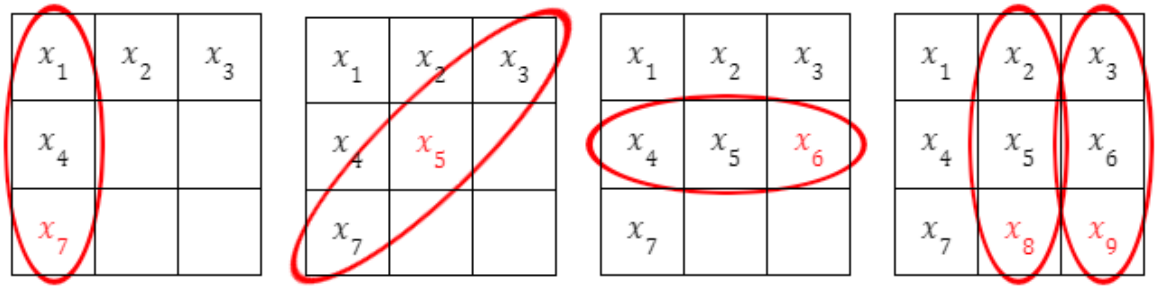
Пусть мы уже поместили какие-то числа в первые 4 ячейки квадрата:



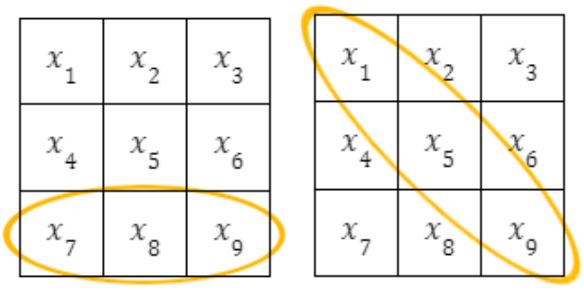
Заметим, что по этим числам мы можем восстановить весь квадрат. Для этого сначала получим сумму *S*, которой должны равняться суммы во всех строках, столбцах и на диагоналях, как

$$S = x_1 + x_2 + x_3$$

Зная эту сумму, восстановим числа в остальных ячейках:



При этом останется две суммы, которые мы ещё не использовали в ходе вычислений. Нужно проверить, что и они тоже равны *S*:



Если суммы сходятся, а в нашем наборе есть все числа получившегося квадрата (причём в нужном количестве, если какое-то из чисел используется несколько раз), то мы нашли ответ. Для быстрой проверки можно использовать хэш-таблицы.

Переберём теперь все способы упорядоченно взять 4 числа из *n* данных и для каждой комбинации применим алгоритм выше. Заметим, что первое число можно выбрать *n* способами, второе придётся выбирать из оставшихся $n - 1$, третье — из $n - 2$, четвёртое — из $n - 3$. Всего придётся рассмотреть не более $13 \cdot 12 \cdot 11 \cdot 10 = 17160$ вариантов, и для каждого выполнить описанные ранее вычисления, которые можно делать достаточно быстро.

Частичные решения

В задачах такого рода хорошо работают решения с перебором случайных вариантов. Можно перебрать некоторое количество случайных вариантов расставить числа, умещающееся в ограничения по времени, и считать, что если ответ не найден, то его нет вовсе. Чем больше при этом используется эвристик для того, чтобы не рассматривать заведомо нерабочие комбинации, тем меньше не рассмотренных вариантов остаётся и тем выше шанс найти ответ, если он есть. Против таких решений сложно подбирать тесты, и они могут даже набрать 100 баллов.