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
Ящики	A	A
Маркеры	B	
Не золотое сечение	C	
Камешки		B
Команда	D	C
Диапазон	E	D
Чаевые		E

Автор задач — Виктор Соловьёв.

«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:

Ящики	10
Маркеры	2
Не золотое сечение	2
Камешки	2
Команда	3
Диапазон	3
Чаевые	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 Диапазон можно попробовать посылать всегда ответ -1 (или, ещё лучше, не всегда, а для всех данных, кроме данных из условия). На этом тоже можно 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».

Ящики

Ответ вычисляется по формуле $23(n - m)$. После всех действий Вити в каждом оставшемся ящике будет $24 - 1$ яблок, а самих ящиков будет $n - m$.

Сложность алгоритма — *O(1).*

Маркеры

Для решения задачи достаточно пройтись циклом по символам входной строки и посчитать количество раз, когда соседние символы различаются.

Можно решить задачу немного иначе: если сначала заменить каждую группу из идущих подряд одинаковых символов всего одним символом, то ответ от этого не поменяется. При этом, в полученной строке все стоящие рядом символы будут различными, а значит при длине строки m ответом будет $m - 1$. Большинство языков идут в комплекте с готовыми алгоритмами для этого, например в C++ можно использовать std::unique:

```
#include <algorithm>
#include <iostream>
#include <string>

int main() {
    std::string s;
    std::cin >> s;

    s.erase(std::unique(s.begin(), s.end()), s.end());
    std::cout << s.length() - 1;

    return 0;
}
```

Сложность приведённых выше решений — *O(n).*

Не золотое сечение

Нетрудно заметить, что действия, совершаемые над сторонами прямоугольника, повторяют **алгоритм Евклида** для нахождения **наибольшего общего делителя**. Стороной полученного квадрата будет НОД(a, b), а количество отрезанных квадратов совпадает с количеством шагов в алгоритме Евклида.

Сложность решения зависит от того, как был реализован алгоритм. Классическая версия алгоритма (когда на каждом шагу большее из чисел a и b заменяется их разностью) не позволяет набрать максимальный балл на тестах третьей группы. Вместо этого требуется использовать модификацию алгоритма Евклида, использующую взятие остатка от деления вместо многократных вычитаний, и дополнить её так, чтобы следить за числом вычитаний.

Сложность оптимизированного алгоритма Евклида оценивается как $O(\log(\min(a, b))$.

Камешки

При решении этой задачи важно понимать, что соперники не обязаны играть оптимально. Условие требует найти минимальное и максимальное количество камешков у Вити среди всех возможных вариантов сыграть партию. Можно с тем же успехом считать, что Саша подыгрывает Вите, помогая тому достичь минимума или максимума камешков к концу партии.

Наивное жадное решение (30 баллов)

Когда Вите нужно набрать максимальное число камешков к концу игры, будем считать, что Витя берёт максимальное количество камешков на каждом своём ходу, а Саша — минимальное (чтобы больше камешков досталось Вите). Аналогично, когда мы ищем минимум, пусть Витя всегда берёт минимальное число камешков на каждом своём ходу, а Саша — максимальное.

Это решение даёт правильный ответ в случае, когда соперники могут брать любое количество камешков от 1 до *b* (или до *c*, соответственно), что позволяет набрать по крайней мере 30 баллов за все тесты первой группы.

Почему наивное решение работает не всегда

Рассмотрим такой пример входных данных:

```
110
15 20
5 5
```

В этом примере Витя может брать от 15 до 20 камешков, а Саша всегда берёт 5. Если Витя всегда будет брать 20 камешков на своём ходу, то соперники совершат 4 полных круга по (20 + 5) = 25 камешка, после чего камешков останется 10, и Витя не сможет их взять:

110 = 4 * (20 + 5) + 10

Витя возьмёт суммарно 4 * 20 = 80 камешков, что не является максимальным значением для таких входных данных. Если вместо этого Витя возьмёт 15 камешков на одном из своих ходов, то и в конце камешков останется 15, и он сможет их забрать:

110 = 3 * (20 + 5) + 1 * (15 + 5) + 15

В этом варианте Витя берёт 3 * 20 + 15 + 15 = 90 камешков

Неочевидная проблема с минимумом

Для минимума работают похожие рассуждения, но оптимальное решение может быть сложнее заметить.

Попробуем найти минимум для таких входных данных:

```
60080
80 90
10 10
```

Жадное решение разделит камешки таким образом:

60080 = 667 * (80 + 10) + 50

Витя возьмёт 667 * 80 = 53360 камешков, 50 камешков останутся лежать в куче к концу игры.

Саша не может брать больше камешков, а если Витя берёт больше камешков, то кажется, что результат ухудшается (больше камешков достаётся Вите, меньше остаётся в куче). Например, если в десяти своих ходах Витя возьмёт 81 камешек вместо 80, получим:

60080 = 657 * (80 + 10) + 10 * (81 + 10) + 40, Витя берёт 53370

продолжая, можем свести число остающихся камешков до 0:

60080 = 617 * (80 + 10) + 50 * (81 + 10) + 0, Витя берёт 53410

однако, если мы продолжим, то обнаружим конфигурацию, при которой игра кончается на ходе Вити, и при этом в кучке остаётся 79 камешков:

60080 = 605 * (80 + 10) + 61 * (81 + 10) + 79, Витя берёт 605 * 80 + 61 * 81 = 53341

Это и есть оптимальный ответ. Обратите внимание, что в нём Витя совершает на 1 ход меньше, чем в жадном решении (605 + 61 < 667).

Решение динамическим программированием (правильное, но медленное), 70 баллов

Мы можем построить динамику, где состояние описывается двумя параметрами — сколько камешков остаётся в куче и чей сейчас ход. Хранить будем или минимальное или максимальное количество камешков у Вити для такой конфигурации (задачу нужно отдельно решить для минимума и для максимума). Состояний будет 2*n, переходов из одного состояния — по числу вариантов хода игрока, то есть не более max(b, d). Ответом будет минимальное (или максимальное) количество камешков Вити из всех состояний, в которых игра закончена (не забывайте при этом, что она может закончиться как на ходе Вити, так и на ходе Саши).

Это решение всегда даёт верный ответ, но не успевает отработать на тестах третьей группы (при больших значениях *n*).

Общая идея полных решений

Когда мы набираем максимальное количество камешков, то либо лучший ответ получается жадно, либо нужно дать Вите совершить ещё один ход (внеся изменения в предыдущие так, что Саша брал больше камешков или Витя брал меньше) —

возможно, что это выгоднее. При этом никогда нет смысла давать Вите совершить на два хода больше, чем в жадном решении — так просто больше камешков от общего количества достанется Саше (который тоже совершит ещё один ход).

Аналогично, при наборе минимума, иногда нужно дать Вите совершить на один ход меньше, чем в жадном решении.

Полное решение 1, жадность + динамика, $O(max(b,\ d)^3)$

Идея этого решения в следующем: когда камешков в куче много, большинство ходов в оптимальном решении игроки должны будут совершить жадно. Совершим какую-то часть ходов жадно, оставив *такую часть камней*, что их достаточно для оптимизации ответа. Для этой части камней применим описанную выше динамику.

Как понять, какое количество камней нужно оставить динамике, чтобы точно не пропустить оптимальный ответ? Количество должно быть таким, чтобы на нём можно было изменить количество ходов на единицу даже тогда, когда только один из соперников может варьировать число взятых на своём ходу камешков, и при том только на единицу. Решение жюри использует оценку сверху $2 \cdot max(b,\ d)^2$. Жадностью нужно забрать максимальное количество камешков, при котором остаётся их достаточно для динамики (это можно сделать с помощью формул целочисленного деления за O(1)). Сложность решения определяется сложностью динамики, где состояний будет пропорционально $max(b,\ d)^2$, а переходов из каждого состояния будет ещё $max(b,\ d)$, что даёт суммарную оценку $O(max(b,\ d)^3)$.

Полное решение 2, формулы, $O(1)$

Можно решить всю задачу при помощи формул, рассмотрев несколько принципиальных случаев (жадные решения можно получить с помощью операций целочисленного деления, после чего нужно попробовать добавить/убрать один ход, но только если это возможно в данных ограничениях; не забывайте также, что игра может кончится как на ходе Вити, так и на ходе Саши).

Команда

Частичное решение за $O(n^3)$, 30 баллов

Используем три вложенных цикла (цикл в цикле в цикле), чтобы перебрать все способы выбрать первого, второго и третьего игроков команды. Для каждой тройки найдём минимальный и максимальный рейтинг и проверим, что они отличаются не более чем в *k* раз. Это решение позволяет пройти все тесты первой группы, но уже для второй группы будет слишком медленным.

В этом решении важно не забыть, что если участники выбирались произвольно, то каждая команда была посчитана 6 раз (потому что есть 6 способов переставить 3 членов команды местами), так что полученный ответ нужно поделить на 6. Вместо этого можно второго участника всегда брать с номером большим, чем у первого, а третьего — с номером больше, чем у второго (тогда порядок участников в команде будет фиксированным, и каждая команда будет посчитана единожды).

Частичное решение за $O(n^2)$, 70 баллов

Оптимизируем решение выше: не будем перебирать третьего игрока в команде, вместо этого научимся за *O(1)* определять, сколько разных игроков могли бы быть в одной команде с уже выбранными двумя. Для этого возьмём минимальный и максимальный из рейтингов двух уже выбранных участников — рейтинг третьего участника не может быть меньше, чем уменьшенный в *k* раз максимальный, и не может быть больше, чем увеличенный в *k* раз минимальный.

Чтобы за *O(1)* получать количество участников в диапазоне от *a* до *b*, воспользуемся префикс-суммами. Сначала посчитаем для каждого возможного рейтинга *x* от 0 до 100000 число *cnt(x)* — количество участников с таким рейтингом (это можно сделать за число операций, равное числу участников, то есть за *O(n)*). Затем посчитаем для каждого рейтинга *x* количество участников *sum(x)*, рейтинг которых не превосходит *x*. Нетрудно заметить, что *sum(1) = cnt(1)*, а *sum(k + 1) = sum(k) + cnt(k + 1)*, что позволяет нам вычислить все sum за число операций, равное количеству возможных значений рейтинга (то есть 100000). Теперь, если мы хотим знать количество участников с рейтингом не менее *a* и не более *b*, мы можем мгновенно вычислить его как *sum(b) - sum(a - 1)*.

В этом решении важно не забыть учесть, что первый или второй участник могут оказаться повторно посчитаны при оценки количества потенциальных третьих участников (если их рейтинг лежит в рассматриваемом диапазоне). Также, если первый и второй участник выбираются произвольно, то каждая команда будет посчитана шесть раз (потому что есть 6 способов переставить 3 членов команды местами), так что полученный ответ нужно поделить на 6.

Полное решение за $O(n \log n)$

Оптимизируем предыдущее решение ещё раз: отсортируем сначала всех участников по рейтингу (быстрая сортировка потребует $O(n \log n)$ операций). Теперь будем второго участника в команде не перебирать, а всегда выбирать так, чтобы у него был максимально большой рейтинг при данном первом участнике (то есть наибольший рейтинг, ещё не превосходящий увеличенного в k раз рейтинга первого участника команды). Количество команд при фиксированных двух участниках будем оценивать как в предыдущем решении.

Можно каждый раз находить второго участника при помощи бинарного поиска (это потребует $O(\log n)$ операций для каждого первого участника, то есть $O(n \log n)$ операций суммарно, что не хуже сложности сортировки) — это уже будет полным решением.

Можно немного упростить алгоритм, если вы не хотите выполнять бинарный поиск. Обратим внимание, что если мы для какого-то первого участника уже нашли максимально удалённого от него по рейтингу второго, то для следующего первого участника нам нет смысла рассматривать на место второго участника кого-то с меньшим рейтингом, чем в прошлый раз. Таким образом, номер второго участника монотонно не убывает при росте номера первого, и мы мы сдвинем его не более n раз за всё решение. Сложность алгоритма тогда определяется сложностью сортировки и остаётся равной $O(n \log n)$, но код получается проще.

Диапазон

Автор задачи: Виктор Соловьёв

Простейшие решения при малом n, 10 баллов

Пусть нам нужно предложить диапазон, в котором 7 раз используется число 5. Мы можем использовать диапазон из одного числа, в записи которого семь пятёрок:

5555555 5555555

Если нам нужно вывести десять двоек, может использовать подряд идущие числа:

222223 222224

Такими способами мы можем пройти тесты первой группы.

Решение скользящим окном, *O(max b * log(max b))*, 40+ баллов

Будем увеличивать значение *b* (строя в качестве ответа диапазон [0, b]) до тех пор, пока не получим не менее *n* использований цифры d на этом диапазоне (когда мы сдвигаем значение *b* на единицу, нам нужно в новом числе посчитать количество цифр *d*, в простейшей реализации будем делать это за количество цифр в числе, то есть *O(log(d))*). Если использований цифры получится ровно n, то мы нашли ответ, если больше — будем сдвигать границу a, пока число использований не уменьшится. После этого нам может понадобиться снова сдвигать b, и так далее.

Интуитивно кажется, что такое решение найдёт ответ, если он вообще существует (если *n* достижимо в ограничениях на *b*): гораздо чаще цифра встречается в числе всего один раз, чем несколько, поэтому кажется, что окно сможет сойтись. В полном решении мы докажем это предположение строго, но на олимпиаде можно просто понадеяться на его справедливость.

Оба числа a и b изменяется не более max b раз, то есть у алгоритма будет не более двух миллиардов итераций, на каждой из которых нужно ещё подсчитывать количество цифр в числе.

Можно оптимизировать подсчёт, если хранить уже посчитанные для других чисел значения: пусть *cnt*(*x*) это число вхождений цифры *d* в запись *x*. Пусть теперь мы приписываем цифру *a* в конец числа *x*, получая число 10*x* + *a*, тогда:

cnt(10*x* + *a*) = *cnt*(*x*) + 1, если a = d,

cnt(10*x* + *a*) = cnt(*x*), в противном случае

Это позволяет нам получать число цифр для нового значения *b* за *O(1)* вместо *O(log b)*, но ценой использования оперативной памяти пропорционально log(max b), что для высоких значений *n* невозможно.

Полное решение за $O(log^2 n)$

Для числа *x* можно за количество его цифр (то есть *O(log x)*) посчитать, сколько раз цифра *d* использовалась на всём диапазоне от 0 до *x* включительно (предлагаем вам вывести алгоритм для этого самостоятельно). Будем далее это количество использований на префиксе называть *F(b)*.

Вооружившись алгоритмом нахождения *F(b)*, применим теперь бинарный поиск, чтобы найти первое приближение *b* — минимальное значение *b*, при котором количество использований цифры *d* не меньше *n*. У бинарного поиска будет не более *log n* шагов, на каждом из которых нам нужно применять алгоритм выше, что даёт нам сложность $O(log^2 n)$ для нахождения первого приближения *b*.

Далее будем действовать по уже описанному в предыдущем решении алгоритму скользящего окна.

Утверждение: до получения ответа граница *a* будет сдвинута менее 200 раз, а граница *b* — не более одного.

Докажем это.

Для ***d* = 0 или *d* = 9** верно следующее:

Количество нулей или девяток в числе не более 9 (потому что *b* ограничено миллиардом). Значит, после вычисления приближенного значения *b*, *F(b)* может превышать *n* не более чем на 9 (если допустить обратное, то мы можем выкинуть какое-то число с не более чем девятью нулями/девятками сверху диапазона, и *F(b)* всё ещё будет не меньше *n*, а значит *b* было не минимально возможным).

У нас будет возможность «избавиться» от 9 нулей или девяток по одной за раз, прежде чем они начнут встречаться в числах сразу по 2 (то есть прежде, чем мы дойдём до 99 или до 100):

0, 10, 20, 30, 40, 50, 60, 70, 80 — уже 9 нулей, до 100 не дошли

9, 19, 29, 39, 49, 59, 69, 79, 89 — уже 9 девяток, до 99 не дошли

То есть мы точно найдём подходящее для данного *b* значение *a* в пределах первой сотни чисел, нам не придётся ни разу менять *b*.

Для **прочих значений *d*** верно следующее:

Возможно, когда граница *a* проходит через число *dd* (две цифры *d* подряд), мы делаем суммарное количество использований слишком маленьким (оно было слишком большим, изменилось сразу на 2, и теперь недостаёт одного использования).

В этом случае увеличим *b* на единицу. Теперь количество использований *d* опять не может превосходить *n* более чем на 9. В то же время, между числами *dd* и 1*dd* есть хотя бы 9 чисел, где *d* используется всего 1 раз.

Покажем это на примере d = 2. После числа 22, цифра 2 встречается в следующих числах:

23, 24, 25, 26, 27, 28, 29, 32, 42 — уже 9 двоек

То же самое справедливо и для прочих значений *d*. Мы приведём только пример для 8, где меньше всего использований 8 в разряде десятков:

89, 98, 108, 118, 128, 138, 148, 158, 168

Таким образом, даже если мы увеличили значение *b* на единицу, подходящее *a* будет найдено среди первых 200 чисел.

Чаевые

Полное решение

Авторское решение использует динамическое программирование со следующими состояниями:

параметр 1 — число уже рассмотренных для взятия купюр

параметр 2 — набранная сумма

хранимое значение — минимальный номинал купюры в наборе при данных параметрах

Для получения ответа нужно из всех состояний, где набранная сумма не меньше *s* + *t*, выбрать состояние с наименьшей суммой, где после вычитания номинала минимальной купюры сумма не становится меньше *s*.

Обратите внимание, что на максимальных ограничениях невозможно хранить все состояния. Нужно использовать тот факт, что достаточно хранить промежуточные ответы только для текущего и предыдущего количества рассмотренных купюр, а не всю историю вычислений. Без этой оптимизации вы не сможете получить все баллы третьей группы тестов.

Полный перебор, 25 баллов

Баллы первой группы тестов можно получить, сделав полный перебор (каждую купюру мы или берём, или нет, что даёт нам 2^n вариантов для рассмотрения).