Общие замечания по всем задачам

Наборы задач для 7-8 и 9-11 классов пересекаются. Всего было 7 задач, ниже разобраны они все.

«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:

Полусумма	8
Дробь	6
Два через два	3
Зачарование	2
Корабли	2
Новогодняя ёлка	10
Мораль	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 о количестве способов разместить корабли на поле можно попробовать посылать всегда ответ 0 или ответ 1 (или, ещё лучше, не всегда, а для всех данных, кроме данных из условия), и на этом тоже получить какие-то баллы.

О-нотация

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

Полусумма

Автор задачи: Виктор Соловьёв

В языках, поддерживающих длинную арифметику, можно решить задачу «в лоб»: сложить числа и поделить результат на 2.

Рассмотрим теперь, как решать задачу, используя только 64-битные типы данных (такие как unsigned long long в C++).

Максимальное значение, которое может иметь беззнаковое 64-битное число, составляет

$$2^{64} - 1 \approx 1.8 \cdot 10^{19},$$

в то время как максиммальное значение суммы *a* и *b* в задаче составляет $2 \cdot 10^{19}$. Иначе говоря, существуют такие входные данные, при которых *a* + *b* переполняет 64-битные типы.

Заметим, что если оба числа чётные, то мы можем сначала поделить их на 2, а уже потом складывать. В этом случае значения всех промежуточных вычислений не превосходят 10^{19}, и 64-битный тип не будет переполнен.

Если оба числа нечётные, то к одному прибавим единицу, а из другого вычтем – сумма от этого не изменится. Для двух чётных чисел решать задачу мы уже умеем.

Дробь

Автор задачи: Виктор Соловьёв

Полное решение за O(q)

Переберём все возможные значения знаменателя *b* от 1 до *q*, для каждого из них вычислим соответствующее значение числителя *a*, исходя из такого соотношения:

$$\frac{x}{y} = \frac{a}{b}$$

Выразим *a*:

$$a = \frac{xb}{y}$$

Это значение может быть дробным. Мы ищем минимальную дробь, строго большую чем *x*/*y*, значит нас, на самом деле, интерусует целое значение *a*, строго большее записанного выше. Его можно вычислить как

$$a = \left\lfloor \frac{xb}{y} \right\rfloor + 1$$

Для деления с округлением вниз можно использовать обычный оператор целочисленного деления (в C++ это оператор /, вызванный для двух целых чисел, в Python – оператор //).

Подсказка: *можно было не выводить формулу с округлением вниз так точно, а просто проверять в решении пару значений около xb/y.*

Теперь, когда для каждого знаменателя мы умеем находить соответсвующий числитель максимально близкой к *x*/*y* дроби, осталось выбрать из всех таких дробей минимальную. Для этого надо уметь сравнивать дроби. Это можно делать в целых числах, используя тот факт, что сравнение

$$\frac{a_1}{b_1} < \frac{a_2}{b_2}$$

эквиваленто сравнению

$$a_1 b_2 < a_2 b_1$$

Наконец заметим, что если мы просматриваем возможные знаменатели *b* в порядке их возрастания, то нам не нужно заботиться о сокращении дроби в ответе. Так, например, мы не могли выбрать в качестве ответа 2/4, потому что раньше бы нашли ответ 1/2, и не стали его заменять.

Частичное решение за $O(q^2)$

При x, y, q $\leqslant 1000$ можно решать задачу, перебирая все пары числителя и знаменателя от 1 до 1001. Такое решение наберёт по крайней мере 50 баллов, но на больших тестах (когда q^2 приближается к 10^{10}) будет работать слишком долго.

О сокращении ответа снова можно не беспокоиться, если при равенстве перебираемых дробей предпочитать ту, у которой знаменатель меньше. Альтернативно, можно поделить выбранные числитель a и знаменатель b на НОД(a, b), это не окажет существенного влияния на производительность.

Другие частичные решения

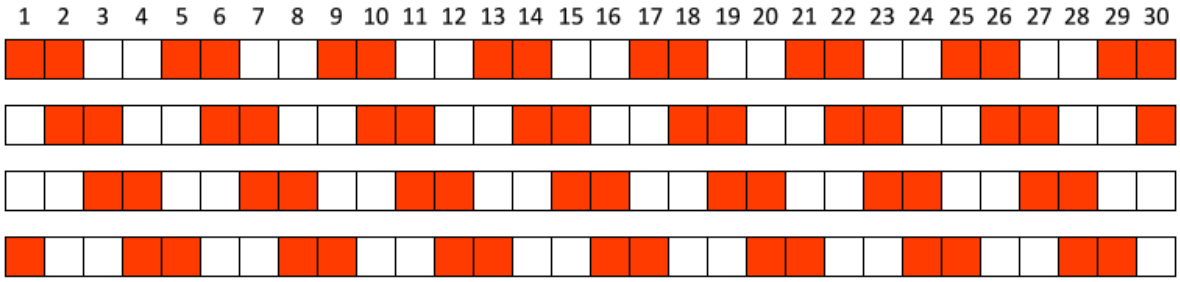
Можно было набрать 28 баллов, всегда используя в качестве знаменателя q.

Два через два

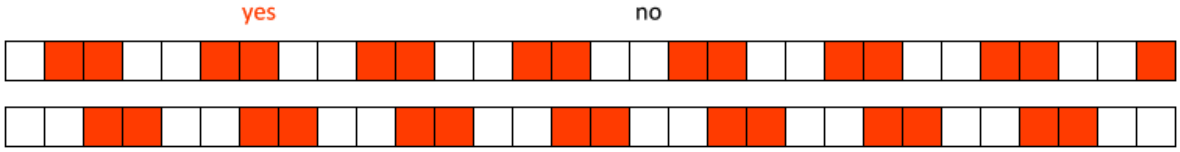
Автор задачи: Виктор Соловьёв

Полное решение перебором

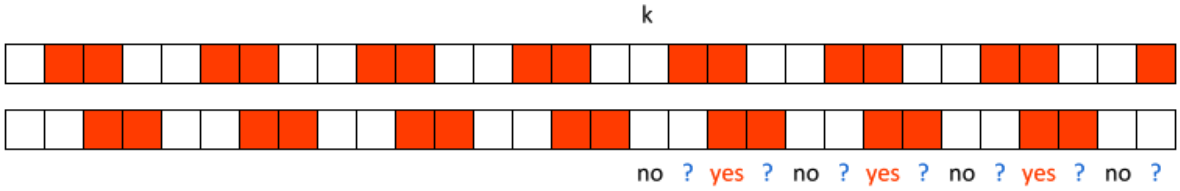
Существует всего 4 различных графика работы два через два:



Выберем те из них, которые удовлетворяют известным ответам Серёжи, например:



Теперь для каждого дня начиная с *k*-о ноября проверим, является ли он рабочим хотя бы в одном из этих графиков, и является ли он нерабочим хотя бы в одном из них. Если возможно и то, и другое, мы увеличим счётчик дней, о которых не можем дать однозначный ответ. Если он может быть только рабочим (или только нерабочим), то увеличим соответствующий счётчик.



Полное решение без перебора

Заметим, что если Серёжа работает в день x, то он работает так же и в дни $x + 4$, $x + 8$, $x + 12$ и так далее. То же самое справедливо и для нерабочих дней.

Кроме того, если в день x серёжа работает, то в дни $x + 2$, $x + 6$, $x + 10$ и так далее – точно не работает (и наоборот).

Мы можем пометить все 30 дней как неопределённые, а потом разметиь рабочие и нерабочие дни, используя два факта выше и имеющиеся два ответа. Может быть неочевидным, но никакой другой информации из входных данных не извлечь, так что остаётся пройтись по календарю и посчитать количество дней различного типа.

Частичное решение для соседних дней

Если нам известна информация о двух соседних днях, то мы можем полностью восстановить весь график Серёжи однозначно. Дней, про которые нам не известен ответ, не будет.

Можем снова использовать тот факт, что если в день x Серёжа работает, то в день $x + 2$ – нет (и наоборот).

Это решение можно дополнительно улучшить, обрабатывая случаи, когда ответы хоть и не даны для двух соседних дней, но эквиваленты ответам для двух соседних дней (за счёт того, что каждые 4 дня картина повторяется). Оно ещё не станет полным, но наберёт больше баллов.

Зачарование

Автор задачи: Виктор Соловьёв

Полное решение за $O(m \log m + n)$

Очевидно, что выгодно сначала накладывать самые дешёвые зачарования, ведь к ним наибольшее число раз применится множитель 2. По этой причине сразу отсортируем цены зачарований по возрастанию.

Заметим также, что на предмет можно наложить сколько угодно зачарований нулевой стоимости, ничего не потратив. Удобно подсчитать количество нулей среди стоимостей зачарования, и просто прибавлять это число для каждого предмета, а из отсортированного списка цен нули убрать.

Может показаться, что для каждого предмета нам нужно просмотреть вплоть до m цен зачарований. В действительности, если бесплатные зачарования мы уже отсеили, то даже для зачарований по цене 1 суммарная стоимость растёт очень быстро, а именно как

$$\left(\ldots\left((1 \cdot 2 + 1) \cdot 2 + 1\right) \cdot 2 + \cdots\right) \cdot 2 + 1 = 2^k - 1,$$

где k – количество эффектов. Таким образом, уже при 50 эффектах единичной стоимости понадобится примерно $1.12 \cdot 10^{15}$ монет, в то время как Витя не закладывает более 10^{15} на предмет. Следовательно, **когда мы исключили нули**, достаточно просматривать не более 50 самых дешёвых зачарований.

Таким образом, на каждый предмет нам нужно не более 50 операций, и обработать их все мы сможем за *O(n)*, с реальным количеством операций не более 50*n*. Чтобы получить окончательную оценку сложности полученного алгоритма, нужно прибавить к этому сложность сортировки массива цен на зачарования. Для полного решения сортировка должна быть быстрой, работающей за $O(m \log m)$, а не за $O(m^2)$. В этом случае, получаем итоговую сложность $O(m \log m + n)$.

В качестве дополнительной оптимизации можно заранее заменить отсортированные цены за зачарования на полные стоимости зачарования всеми эффектами по этот включительно.

Кроме того, если отсортировать по возрастанию заложенных монет предметы, то можно использовать тот факт, что на следующий предмет можно налоложить по крайней мере столько же эффектов, сколько и на предыдущий. С такой модификацией можно и не исключать нули.

Не стоит забывать об использовании достаточно вместительных типов данных, чтобы работать с числами порядка 10^{15}.

Частичные решения

При n, m $\leqslant 10$ можно перебрать для каждого предмета все возможные порядки наложения эффектов, получив сложность $O(m! \cdot n)$. Перебрать все перестановки списка цен зачарований можно при помощи встроенных алгоритмов, например std::next_permutation в C++.

При n, m $\leqslant 1000$ можно использовать медленную сортировку за $O(m^2)$.

Корабли

Автор задачи: Виктор Соловьёв

Полное решение динамикой за $O(mn)$

Подсказка: *если в задаче просят определить количество способов сделать что-то, то с выскокой вероятностью она решается динамическим программированием. Кроме того, всякий раз, когда вы не представляете, как могла бы решаться задача, не лишним будет вспомнить о динамическом программировании.*

Будем использовать динамическое программирование.

Раз порядок выставления кораблей не важен, мы можем для удобства считать, что Витя ставит следующие корабли только правее предыдущих. Тогда, если мы находимся в процессе выставления кораблей, то наше состояние описывается двумя величинами: количеством уже размещённых кораблей и текущей клеткой на поле (начинаем мы с 0 кораблей и нулевой клетки), а хранить для каждого состояния мы будем число способов достичь его. Чтобы хранить все возможные состояния, нам понадобится таблица из $m + 1$ строки по $n + 1$ ячейки.

Чтобы не оперировать отдельно зазорами между кораблями, можем считать, что ставить корабли можно в притык друг к другу, но первый корабль имеет ширину k, а все последующие – $s + k$. Из некоторого состояния мы можем совершить два различных перехода: либо разместив начиная с этой клетки корабль (тогда число кораблей увеличится на 1, а положение – на длину корабля), либо не размещать корабль и просто перейти на следующую клетку поля (число кораблей не поменяется, положение увеличится на 1). Рассматривая какую-то ячейку таблицы, будем прибавлять результат в ней к результатам в тех двух ячеяйках, в которые возможен переход.

Заполнив всю таблицу, мы будем иметь ответ на вопрос задачи в ячейке, соответсвующей m кораблям и последней клетке поля.

Чтобы не возникло переполения целочисленных типов, следует все промежуточные вычисления заменять их остатком от деления на 1000000007. Модификация авторского решения, в которой не производится взятий остатка, набирает 61 балл.

Кроме того, если хранить таблицу полностью, то может происходить переполнение отведённой оперативной памяти. Заметим, что на самом деле достаточно всегда хранить всего две строки таблицы: для текущего и следующего количества кораблей (без этой модификации авторское решение не проходит 2 теста).

Полное решение комбинаторикой за $O(n + m)$

Задачу можно свести к вычислению одного биномиального коэффициента (см метод шаров и перегородок, теорема 2):

$$a = n - mk - (m-1)s,$$

$$C_{a+m}^{m} = \frac{(a+m)!}{a!\,m!}$$

Здесь a – количество клеток, не занятое кораблями и обязательными зазорами между ними. Именно эти клетки мы можем перераспределять. Отдельно нужно обработать ситуацию, когда $a < 0$ (в этом случае ответ равен 0, а $a!$ не определено).

При этом, так как в формуле присутствует деление, придётся либо использовать длинную арифметику (в Python, Java и C# есть встроенные типы для хранени больших чисел), и посчитать числитель целиком, либо промежуточные вычисления сразу производить в арифметике по модулю 1000000007, заменив при этом деление умножением на обратный элемент по модулю. В случае, если была выбрана длинная арифметика, реальная сложность алгоритма будет больше указанной асимптотики (за счёт действий с длинными числами), однако решение всё равно будет достаточно быстрым.

Частичные решения

При $n \leqslant 20$ можно перебирать все варианты помечания клеток как занятых и свободных, и считать, в скольки из этих вариантов получилось m кораблей с достаточно длинными зазорами между ними (решение за $O(2^n \cdot n)$). Перебирать варианты можно рекурсивно или при помощи битовых масок.

При $n \leqslant 1000$ можно не беспокоиться о размере таблицы, и не реализовывать вариант с хранением всего двух строк.

Новогодняя ёлка

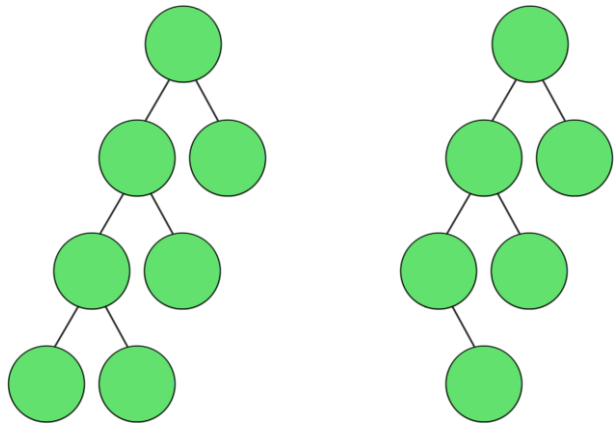
Авторы задачи: Данил Мячин, Виктор Соловьёв

Полное решение за O(n)

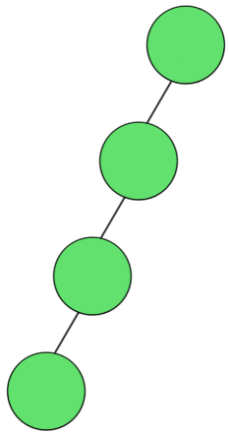
Заметим, во-первых, что в задаче нет никаких бонусов за взаимодействие шариков разного цвета. Это значит, что для каждого цвета мы можем строить отдельное дерево, а затем как угодно соединить их между собой (главное, чтобы при этом ни у какого шарика не было больше двух сыновей).

В этой задаче всегда работает одна из двух оптимальных стратегий построения дерева из шариков одного цвета.

Если $b_2 > 2b_1$, то выгодно у максимально большого числа вершин сделать по два сына. Проще всего при этом объединять их в следующую формацию (для нечётного и чётного числа шариков, соответственно):



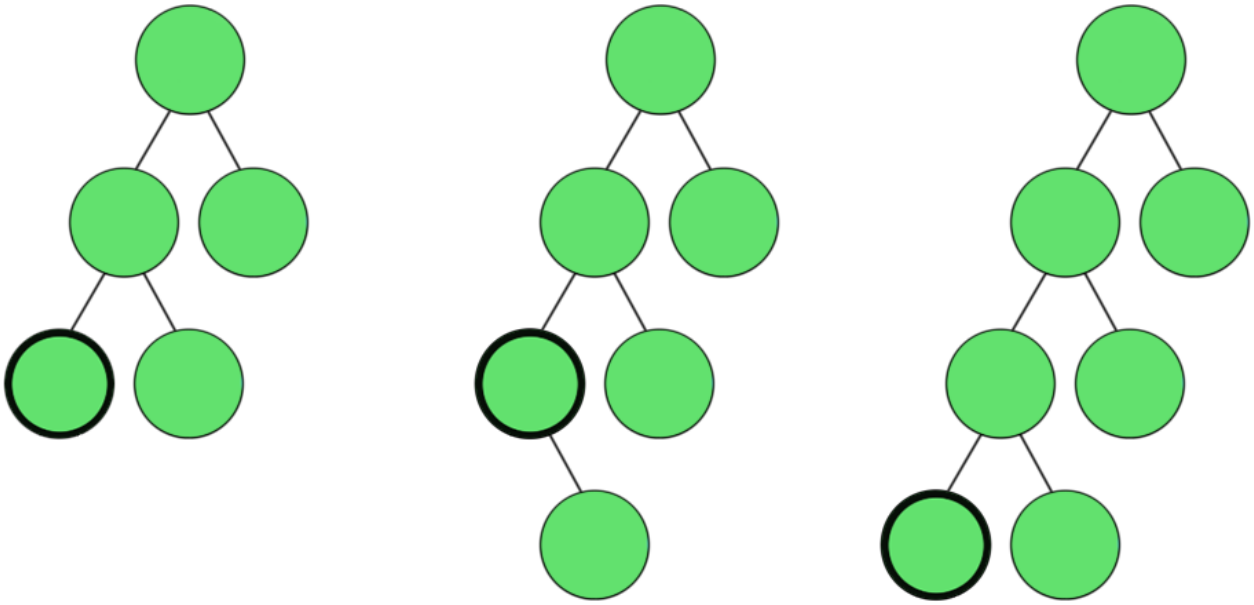
Если же $b_2 \leq 2b_1$, то выгоднее делать всего по 1 сыну:



Рассмотрим сначала, как быстро формировать второй вариант. Для каждого шарика будем хранить индекс его родителя. Для каждого цвета будем хранить два индекса: индекс первого встреченного шарика такого (понадобится, чтобы потом соединять разные цвета между собой) и индекс последнего встреченного шарика (он станет родителем следующего шарика того же цвета,

когда мы его встретим). В конце останется соединить последний шарик каждого цвета с первым шариком следующего цвета.

При формировании дерева при $b_2 > 2b_1$ будем использовать практически такой же подход, но дополнительно будем хранить количество уже имеющихся у шарика сыновей, а менять индекс последнего шарика для цвета будем только тогда, когда добавляем к предыдущему «последнему» шарику уже второго сына:



Соединение деревьев разного цвета между собой будет работать так же.

Если мы строим все разноцветные деревья одновременно, за один проход по списку шариков, то достигается сложность O(*n*).

Частичные решения

Решение, выводящее всегда последовательность 0 1 2 ... *n*-1, набирает 33 балла. Это связно с тем, что оно полностью проходит тесты, где все шарики разных цветов (их можно соединять в любом порядке, лишь бы получилось валидное дерево), и набирает некоторое количество баллов на остальных.

Решение, всегда предпочитающее добавление двух сыновей, набирает 70 баллов.

Решение, всегда предпочитающее добавление одного сына, набирает 75 баллов.

Мораль

Авторы задачи: Виктор Соловьёв, Александр Пак

Полное решение за $O(m \log m + 2^n \cdot n)$ перебором масок фракций

Заметим, что если мы уже выбрали, из какого набора фракций хотим взять отряды, то набрать их оптимальным образом можно жадно. Для этого сначала возьмём по 1 самому сильному отряду из каждой выбранной фракции, а затем будем добирать отряды до нужного количества, выбирая самые дорогие из всех отрядов выбранных фракций. Если мы заранее отсортировали все отряды по убыванию силы (за $O(m \log m)$), то набрать из них лучшие (пропуская отряды из не выбранных фракций) мы можем за $O(m)$. Следует отлавливать ситуации, когда взяв по 1 отряду каждой выбранной фракции мы уже превысили k, или когда мы не сумели набрать k отрядов, используя только выбранные фракции. Когда отряды собраны, не забудьте вычесть соответствующий числу фракций штраф за мораль.

Различных наборов фракций 2^n. Переберём их все, и для каждого наберём отряды жадно, как описано выше. Из всех получившихся сил отрядов выберем наибольшую – она и будет ответом.

Для организации перебора наборов фракций удобно использовать битовые маски (или, иначе говоря, множество, организованное на битах числа).

Итоговая сложность такого решения – $O(m \log m + 2^n \cdot n)$.

Частичные решения

Если последовательность штрафов строго неубывает, то можно слегка упростить описанное выше решение перебором масок, а именно не набирать сначала по 1 отряду каждой фракции. При этом, возможно, мы используем не весь набор выбранных фракций, а значит нужно подсчитывать, сколько разных фракций мы реально использовали (чтобы вычесть правильный штраф). Это работает, потому что в данном случае штраф за меньшее число фракций окажется не больше, чем за все фракции выбранного набора, а значит оптимальный ответ мы не пропустим. В полном же решении, возможно, стоит взять более слабый отряд, чтобы при этом расширить число фракций, и получить меньший штраф.

Также можно использовать динамическое программирование, организовав очевидную (если у вас есть опыт написания таких динамик) динамику по трём параметрам: числу просмотренных отрядов, числу взятых отрядов и маске взятых фракций.