Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

Разборы решений и идеи тестов

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

11 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 256 мегабайт

11.1. «Фантастический роман». В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов ⨕ и ⨗, которые в рабочем варианте текста Петя представляет заглавными буквами `F` и `G` (для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1\ldots\alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i =$ `F`, и 1, если $\alpha_i =$ `G`. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на число $D = 10^9 + 7$.

Такое число называется *хешем* слова w, а правило вычисления хеша — *хеш-функцией*. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют *коллизией*), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для работы со словарём нужна программа, по заданному слову находящую его хеш.

Формат входа: В первой строке задано целое число p, участвующее в вычислении хеша ($1 \leqslant p \leqslant 10^9$). Во второй строке задано слово w — непустая последовательность заглавных символов латиницы `F` и `G` длиной не более 1000 символов.

Формат выхода: Выведите единственное целое число — хеш слова w (остаток от деления $h(w)$ на D).

Пример 1

Вход:	*Выход:*
`10` `FGFG`	`1010`

Пример 2

Вход:	*Выход:*
`10` `FGFGFGFGFGFGFG`	`10030303`

11.2. «Хитрая операция». Петя Торопыжкин придумал новую операцию над целыми числами: от числа отделяется последняя цифра его десятичной записи и к получившемуся после отделения числу прибавляется эта отделённая цифра, умноженная на сто. Если предыдущее число было однозначным, то после отделения его последней цифры получается ноль. Для заданного начального числа a выдайте результат, который получится после k применений придуманной Петей операции.

Формат входа: В единственной строке через пробел заданы два целых числа: a — начальный член вычисляемой последовательности — и k — количество применений операции ($0 \leqslant a, k \leqslant 2 \cdot 10^9$).

Формат выхода: Выдайте число, получаемое после k применений операций к заданному исходному числу.

Пример

Вход:	*Выход:*
123456789 4	12924

11.3. «Логистический центр». В большом городе с квадратной застройкой введена координатная система так, что все прямые $x = a$ и $y = b$ для целых a и b — это улицы, по которым возможно передвижение транспорта. На некоторых перекрёстках расположены магазины. Владелец сети магазинов решил разместить на каком-то перекрёстке (возможно, на том, где уже есть магазин) логистический центр так, чтобы сумма расстояний (при движении по улицам) от него до всех магазинов была наименьшей. Напишите программу, которая будет находить подходящее место.

Формат входа: В первой строке задано единственное целое число n — количество магазинов ($1 \leqslant n \leqslant 10^5$). В следующих n строках через пробел перечислены пары координат x_i, y_i магазинов ($|x_i|, |y_i| \leqslant 10^9$).

Формат выхода: Через пробел выведите координаты логистического центра и сумму расстояний от него до всех магазинов. Если наилучший результат может быть обеспечен размещением центра в более, чем одной точке, выведите любую из них.

Пример

Вход:	*Выход:*
5 0 0 4 2 1 4 0 0 2 2	1 2 12

11.4. «Дискретное показательное уравнение». Часто операции с числами ведутся по модулю какого-либо числа p, например, $a \uparrow\!\!\uparrow_p b = a^b \bmod p$. Напишите программу, которая для заданных величин p и a будет находить все решения b_i и c_j уравнения $a \uparrow\!\!\uparrow_p b_i = c_j \bmod p$, выбираемые из заданных множеств. Нужно разделить исходные множества чисел b и c на пары соответствующих друг другу наборов: любое число b и любое число c из пары наборов удовлетворяют указанному уравнению.

Формат входа: В первой строке через пробел заданы два целых числа p и a ($1 \leqslant a, p \leqslant 10^9$). Во второй строке задано целое число n — размер набора целых чисел $\{b_i\}$, которые через пробел перечислены в третьей строке. В четвёртой строке задано целое число m — размер набора целых чисел $\{c_j\}$, которые через пробел перечислены в пятой строке. Выполнены ограничения $1 \leqslant n, m \leqslant 10^5$; $0 \leqslant b_i, c_j \leqslant 10^9$.

Формат выхода: Результирующие пары наборов нужно вывести парами строк: в первой строке из пары содержится набор чисел b, во второй — набор чисел c. В начале каждой строки выводится количество чисел в соответствующем наборе. Если какому-то набору b не соответствует ни одного c или какому-то набору c не соответствует ни одного b, то для такого набора строка, представляющая парный набор, должна содержать только `0`. Порядок перечисления чисел в наборе и порядок перечисления пар наборов могут быть любыми.

Пример

Вход:	*Выход:*
5 4 4 1 2 3 4 6 2 10 5 9 7 17	2 1 3 1 9 2 2 4 0 0 5 2 10 5 7 17

Примечание: $(4^3 \bmod 5 = 64 \bmod 5) = (4^1 \bmod 5 = 4 \bmod 5) = 9 \bmod 5$; $4^2 \bmod 5 = 16 \bmod 5 = 1$, $4^4 \bmod 5 = 256 \bmod 5 = 1$, ни одно из остальных чисел c_j не даёт остаток 1 при делении на 5.

11.5. «Ночёвка на трассе». Летом Петя Торопыжкин в составе компании из n друзей поучаствовал в мотопробеге в Европу (каждый участник ехал на своём мотоцикле). К концу одного из дней пробега, когда пришло время вставать на ночёвку, они были на платной дороге, состоящей из l отрезков. Для i-го участника пробега известен номер x_i отрезка дороги, на котором он находится. Стоимость проезда по k-му отрезку равна d_k, не зависит от направления проезда и взимается при въезде на этот отрезок; дальнейшее движение по отрезку не требует затрат. Во время подготовки к поездке было выяснено, что на этой дороге имеется m мотелей. Мотель с номером j расположен на отрезке дороги с номером y_j (на одном участке дороги может быть один мотель, несколько мотелей или не быть вовсе) и может вместить p_j постояльцев. Общая вместимость всех мотелей достаточна, чтобы все друзья могли заночевать под крышей. По имеющимся данным нужно определить минимальные затраты на дорожные сборы, чтобы каждый из друзей доехал до какого-нибудь мотеля.

Формат входа: В первой строке через пробел указаны целые числа l, n и m — количества участков дороги, участников мотопробега и мотелей ($1 \leqslant l, n, m \leqslant 5000$). Во второй строке через пробел перечислено l целых чисел d_k — стоимости проезда по участкам дороги в порядке следования их на дороге ($1 \leqslant d_k \leqslant 10000$). В третьей строке через пробел перечислено n целых чисел x_i — номера отрезков дороги, на которых были участники пробега в момент начала распределения на ночёвку ($1 \leqslant x_i \leqslant l$). В следующих m строках перечислены описания мотелей — перечисленные через пробел два целых числа y_j и c_j ($1 \leqslant y_j \leqslant l$, $1 \leqslant c_j \leqslant 5000$).

Формат выхода: Выведите единственное целое число — минимально возможные дорожные расходы, связанные с достижением мотелей всеми участниками.

Пример

Вход:	*Выход:*
3 5 4 50 20 10 3 1 1 2 1 2 1 3 5 1 2 3 1	30

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году
11 класс. Разбор решений и идеи тестов

11.1. «Фантастический роман». *В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов ≎ и ◊, которые в рабочем варианте текста Петя представляет заглавными буквами* `F` *и* `G` *(для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1\ldots\alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i =$* `F`*, и 1, если $\alpha_i =$* `G`*. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на число $D = 10^9 + 7$.*

Такое число называется хешем слова w, а правило вычисления хеша — хеш-функцией. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют коллизией), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для работы со словарём нужна программа, по заданному слову находящую его хеш.

По мнению программного комитета, данная задача является очень простой. От уровня утешительной её отделяет две технические тонкости.

Первая техническая тонкость заключается в том, что нужно считать степени числа p. Здесь имеются следующие методы:

1) Можно при обработке каждого символа хранить предыдущую степень p и на её основе вычислять следующую.

2) Можно применить алгоритм *быстрого возведения в степень*:

$$a^n = \begin{cases} n = 2k, & (a^2)^k, \\ n = 2k+1, & (a^2)^k \cdot a. \end{cases}$$

3) Можно применить схему Горнера вычисления значения многочлена в точке:

$$\begin{aligned} P(x) &= \alpha_m x^m + \alpha_{m-1}x^{m-1} + \alpha_{m-2}x^{m-2} + \ldots + \alpha_2 x^2 + \alpha_1 x + \alpha_0 = \\ &= (\alpha_m x^{m-1} + \alpha_{m-1}x^{m-2} + \alpha_{m-2}x^{m-3} + \ldots + \alpha_2 x + \alpha_1)x + \alpha_0 = \\ &= \big((\alpha_m x^{m-2} + \alpha_{m-1}x^{m-3} + \alpha_{m-2}x^{m-4} + \ldots + \alpha_2)x + \alpha_1\big)x + \alpha_0 = \\ &= \ldots = \Big(\ldots\Big(\big((0\cdot x + \alpha_m)x + \alpha_{m-1}\big)x + \alpha_{m-2}\Big)x + \ldots\Big)x + \alpha_0. \end{aligned}$$

То есть, чтобы вычислить значение многочлена в точке, можно, начав с нулевого значения аккумулятора, перебирать коэффициенты от старших к младшим (в нашем случае — с конца очередной строки к началу) и проводить итеративную процедуру:

предыдущее значение аккумулятора умножается на значение x (в нашем случае p) и к результату прибавляется очередной коэффициент.

Вторая техническая тонкость заключается в том, что все вычисления надо проводить по модулю D: $u + v \to (u + v) \bmod D$, $u \cdot v \to (u \cdot v) \bmod D$. А поскольку значение D достаточно большое, то вычисления следует проводить в 8-байтном целом типе `__int64` (или его аналогах в других языках).

Тесты подобраны так, что участник, не использующий длинный целый тип получит около 60% баллов.

Идеи тестов:

1. Слово `F`, $p = 2$.
2. Слово `G`, $p = 2$.
3. Слово `F`, $p = 10^9$.
4. Слово `G`, $p = 10^9$.
5. Слово из 10 символов `F`, $p = 2$.
6. Слово из 10 символов `G`, $p = 2$.
7. Слово из 30 символов `F`, $p = 2$.
8. Слово из 30 символов `G`, $p = 2$.

9–12. Случайные тесты для $p = 2$, длина слова не превосходит 30.

13–17. Случайные тесты, слова длинные.

18–20. Слова максимальной длины, различные p.

11.2. «Хитрая операция». *Петя Торопыжкин придумал новую операцию над целыми числами: от числа отделяется последняя цифра его десятичной записи и к получившемуся после отделения числу прибавляется эта отделённая цифра, умноженная на сто. Если предыдущее число было однозначным, то после отделения его последней цифры получается ноль. Для заданного начального числа a выдайте результат, который получится после k применений придуманной Петей операции.*

Задача, хотя и кажется очень простой, на самом деле содержит подвох: номер члена настолько велик, что последовательное вычисление членов последовательности не уложится в ограничения по времени. Однако несложное математическое рассуждение показывает, что долго считать не нужно.

Действительно, если число трёхзначное $\overline{a_1a_2a_3}$, то указанная операция просто переставляет последнюю цифру вперёд: $\overline{a_1a_2a_3} \to \overline{a_1a_2} + 100a_3 = \overline{a_3a_1a_2}$. Стало быть, если мы получили какое-то трёхзначное число $\overline{a_1a_2a_3}$ на шаге $m < k$, то с учётом того, что числа через три начнут повторяться, можно утверждать, что
1) если $(k - m) \bmod 3 = 0$, результат есть $\overline{a_1a_2a_3}$;
2) если $(k - m) \bmod 3 = 1$, результат есть $\overline{a_3a_1a_2}$;
3) если $(k - m) \bmod 3 = 2$, результат есть $\overline{a_2a_3a_1}$.

Остаётся вопрос: а дойдём ли мы до трёхзначного числа? Ответ — да. Если число

большое. то данная операция его уменьшает:

$$\overline{\alpha\beta}\ (\beta \in 0..9, \alpha \geqslant 1) \to \alpha + 100\beta;$$
$$\overline{\alpha\beta} - (\alpha + 100\beta) = (10\alpha + \beta) - (\alpha + 100\beta) = 9\alpha - 99\beta = 9(\alpha - 11\beta).$$

Если $\alpha > 99 \geqslant 11\beta$, то новое число будет меньше старого. А неравенство $\alpha > 99$ как раз и означает, что число $\overline{\alpha\beta}$ более чем трёхзначное. Вывод: четырёхзначные и более длинные числа при применнии данной операции уменьшаются, и мы обязательно дойдём до трёхзначного числа. Причём количество операций до достижения трёхзначного числа будет не слишком велико — не более 10–15: на каждом шаге число на 1 уменьшает свою разрядность.

Тесты подобраны так, что программа, построенная на «лобовом» вычислении получит около 45% баллов.

Идеи тестов:

1–4. $a = 0$, $k = 0$ / 10 / 10000 / 10^9.
5–8. $a = 10$, $k = 0$ / 10 / 10000 / 10^9.
9–12. $a = 123$, $k = 0$ / 10 / 10000 / 10^9.
13–15. Случайные тесты, a и k таковы, что мы не доходим до трёхзначного числа.
16–25. Случайные тесты, $k = 10^9$.

11.3. «Логистический центр». *В большом городе с квадратной застройкой введена координатная система так, что все прямые $x = a$ и $y = b$ для целых a и b — это улицы, по которым возможно передвижение транспорта. На некоторых перекрёстках расположены магазины. Владелец сети магазинов решил разместить на каком-то перекрёстке (возможно, на том, где уже есть магазин) логистический центр так, чтобы сумма расстояний (при движении по улицам) от него до всех магазинов была наименьшей. Напишите программу, которая будет находить подходящее место.*

Сложность этой задачи, по мнению программного комитета, является средней. С одной стороны, при знании специальных алгоритмов эта задача может быть быстро решена. С другой стороны, имеется путь, не требующий специальных знаний, но для его реализации необходимо провести математический анализ задачи.

Лобовое решение, связанное с последовательным перебором точек, возможных положений центра, в случае широко расположенных точек не будет укладываться в ограничения по времени.

Оптимальное решение, требующее знаний специальных алгоритмов, основывается на математическом рассмотрении сути задачи.

Расстояние между двумя точками, используемое в задаче, часто называется *манхеттенским* и вычисляется по формуле $d_1(a, b) = |a_x - b_x| + |a_y - b_y|$. Для фиксированной точки a и переменной второй точки (x, y) для расстояния между ними имеем выражение $f_a(x, y) = |x - a_x| + |y - a_y|$. Данная функция является *выпуклой* в том смысле,

что её надграфик — выпуклое множество. *Надграфиком* функции $f(x, y)$ двух переменных называют множество в трёхмерном пространстве $\operatorname{epi} f = \big\{(x, y, z) : z \geqslant f(x, y)\big\}$. Множество называется *выпуклым*, если вместе с любыми двумя своими точками оно содержит и отрезок с концами в этих точках.

Функция, которую нужно минимизировать есть сумма манхеттенских расстояний:

$$F(x, y) = \sum_{i=1}^{n} f_{a_i}(x, y) = \sum_{i=1}^{n} \big(|x - a_{i,x}| + |y - a_{i,y}|\big) \to \min_{(x,y)}.$$

Не очень сложно доказывается, что сумма любого конечного числа выпуклых функций является выпуклой функцией. То есть можно утверждать, что минимизируемая функция $F(x, y)$ выпукла.

Также из выпуклости функции двух переменных следует, что сужение этой функции на любую прямую вида $y = y^*$ является выпуклой функцией $F(x, y^*)$ одного аргумента x. При этом от минимизации по двум переменным можно перейти к *повторной* минимизации:

$$\min_{(x,y)} F(x, y) = \min_{y} \min_{x} F(x, y) = \min_{y} g(y). \qquad (*)$$

Наконец, выпуклость функции $F(x, y)$ влечёт выпуклость функции минимумов по первому аргументу $g(y) = \min\limits_{x} F(x, y)$.

Рассуждения о выпуклости различных функций важны, поскольку выпуклая функция одного переменного является *унимодальной*, то есть слева она имеет участок строгого убывания, затем отрезок (возможно, одноточечный), где функция принимает своё минимальное значения, после чего имеется участок строгого возрастания. Участки убывания и/или возрастания могут отсутствовать. Кроме того, важно, что функция не имеет участков постоянства, кроме участка минимума. В силу такого хорошего устройства унимодальные функции допускают очень быстрый алгоритм приближенного поиска минимума, который называется *тернарный (или троичный) поиск*.

Суть этого алгоритма в случае минимизации унимодальной функции по вещественному аргументу заключается в следующем. Пусть в начале нам указан отрезок $[\alpha_0, \beta_0] = [\alpha, \beta]$, на котором надо найти минимум унимодальной функции h. На каждой итерации алгоритма по имеющемуся отрезку $[\alpha_k, \beta_k]$ вычисляются две точки: $p_1 = (2\alpha_k + \beta)/3$ и $p_2 = (\alpha_k + 2\beta)/3$, делящие отрезок на три равные части. Затем сравниваются значения функции h в точках p_1 и p_2, и рассматриваемый отрезок сужается:

1. если $h(p_1) > h(p_2)$, то в качестве нового отрезка берём $[\alpha_{k+1}, \beta_{k+1}] = [p_1, b]$;
2. если $h(p_1) < h(p_2)$, то в качестве нового отрезка берём $[\alpha_{k+1}, \beta_{k+1}] = [a, p_2]$;
3. случай $h(p_1) = h(p_2)$ можно включить в любой из рассматриваемых случаев или же положить в этом случае $[\alpha_{k+1}, \beta_{k+1}] = [p_1, p_2]$.

Действительно, в первом случае из-за того, что $h(p_1) > h(p_2)$, отрезок $[p_1, p_2]$ пересекается с участком убывания функции h, и, значит, минимум находится правее точки p_1. Соответственным образом сужаем отрезок. Наоборот, во втором случае отрезок $[p_1, p_2]$ пересекается с участком возрастания функции, и минимум расположен левее точки p_2.

Данная операция повторяется до тех пор, пока длина отрезка $[\alpha_k, \beta_k]$ не станет меньше требуемой точности отыскания точки минимума. В качестве приближения точки минимума можно выдать любую точку этого маленького отрезка, например, середину или один из концов.

В случае поиска минимума по целому аргументу деления при вычислении точек p_1 и p_2 выполняются нацело, и операция повторяется до тех пор, пока длина отрезка не станет равной 2 (и, соответственно, перестанет содержать две целых внутренних точки). Из оставшихся трёх точек — концов отрезка и его середины — прямым сравнением находится точка минимума.

Видно, что на каждом шаге длина отрезка сокращается в полтора раза, а значит сложность тернарного поиска есть $O(\log l)$, где l — длина исходного отрезка.

Таким образом, вычисление минимума (*) связано с вычислением минимума унимодальной функции $g(y)$, осуществляемым при помощи тернарного поиска. При этом значение $g(y)$ для каждого требуемого значения y^* вычисляется как минимум унимодальной функции $F(x, y^*)$, что также производится при помощи тернарного поиска. Общая сложность алгоритма с учётом того, что для вычисления значения $F(x, y)$ требуется вычислить и просуммировать n слагаемых, есть $O(n \cdot \log \Delta x \cdot \log \Delta y)$, где Δx и Δy — размеры прямоугольника, на котором ищется минимум функции двух переменных.

Однако есть путь решения, не требующий знания указанного алгоритма поиска минимума функции. Рассмотрим минимизируемую функцию:

$$F(x,y) = \sum_{i=1}^{n} f_{a_i}(x,y) = \sum_{i=1}^{n} \big(|x - a_{i,x}| + |y - a_{i,y}|\big) = \sum_{i=1}^{n} \big(|x - a_{i,x}|\big) + \sum_{i=1}^{n} \big(|y - a_{i,y}|\big).$$

Во-первых, мы видим, что выражение распалось на два слагаемых, одно из которых зависит только от координаты x подбираемой точки, а второе — только от координаты y. Это значит, что мы можем независимо работать с подбором x и с подбором y. Такая ситуация была бы невозможной, если бы рассматривалось расстояние, в котором переменные были бы связаны, например, обычное евклидово расстояние $d_2(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$ или «бесконечное» («шахматное») расстояние $d_\infty(a,b) = \max\big\{|a_x - b_x|, |a_y - b_y|\big\}$.

Рассмотрим теперь поведение каждого из этих слагаемых на примере первого (второе ведёт себя аналогично). Для начала рассмотрим ситуацию с двумя модулями: $g(x) = |x - \alpha| + |x - \beta|$. Если $\alpha = \beta$, то имеем функцию $g(x) = 2|x - \alpha|$, имеющую минимум при $x = \alpha$; или можно формально сказать, что при $x \in [\alpha, \beta]$. Если же $\alpha \neq \beta$ (для определённости, пусть $\alpha < \beta$), то форма этой функции известна: убывание при $x < \alpha$, плато при $x \in [\alpha, \beta]$, возрастание при $x > \beta$. Значит, такая функция в этом случае принимает минимальные значения также при $x \in [\alpha, \beta]$.

Вернёмся теперь обратно к большому количеству точек: $h(x) = \sum_{i=1}^{m} |x - \alpha_i|$. Пусть значения α_i упорядочены по возрастанию: $\alpha_1 \leqslant \alpha_2 \leqslant \alpha_3 \leqslant \ldots \leqslant \alpha_{m-1} \leqslant \alpha_m$. Рассмотрим два случая.

Первый — $m = 2k$, число чётное. Тогда мы можем разбить числа на пары (α_1, α_m), (α_2, α_{m-1}), (α_3, α_{m-2}), ... Сумма h разобьётся на пары модулей, как это было у функции g, причём плато пары слагаемых с меньшим номером первого из чисел α_i охватывает плато пары с бóльшим номером первого из чисел α_i:

$$[\alpha_1, \alpha_m] \supset [\alpha_2, \alpha_{m-1}] \supset [\alpha_3, \alpha_{m-2}] \supset \dots$$

Значит, минимума функция h будет достигать на плато $[\alpha_{m/2}, \alpha_{m/2+1}]$ самой последней пары чисел $(\alpha_{m/2}, \alpha_{m/2+1})$.

Второй случай — $m = 2k + 1$, число нечётное. Тогда аналогичными рассуждениями можно понять, что минимум функция h получает в точке $\alpha_{(m+1)/2}$ — средней точке отсортированного набора α_i.

В математике значение, разделяющее некоторый набор величин пополам, называется *медианой* (это значение не обязательно принадлежит набору). Таким образом, задача минимизации функции h сводится к поиску медианы набора $\{\alpha_i\}$. Этот поиск производится посредством сортировки: набор $\{\alpha_i\}$ сортируется по возрастанию и, если количество чисел в нём нечётно, берётся средняя точка отсортированного набора, а если чётно, то любая точка из отрезка между двумя средними точками в отсортированном наборе.

Таким образом, задача может быть решена следующим образом: ищем x^* — медиану набора $\{a_{i,x}\}$, ищем y^* — медиану набора $\{a_{i,y}\}$, считаем значение функции $F(x^*, y^*)$, которое будет минимальным. Выдаём ответ.

Сложность этого решения $O(n \log n)$ — нужно провести две сортировки массивов объёмом n. Потенциально это работает быстрее в сравнении с тернарным поиском.

Идеи тестов:

1. Один магазин.
2. Два совпадающих магазина.
3. 100 совпадающих магазинов.
4. 10^5 совпадающих магазинов.
5. Три магазина, два совпадают.
6. 100 магазинов, 99 совпадают.
7. 10^5 магазинов, 99999 совпадают.
8. 4 магазина — две группы по 2 совпадающих.
9. 100 магазинов — две группы по 50 совпадающих.
10. 10^5 магазинов — две группы по 50000 совпадающих.
11. Небольшой квадрат со сторонами, параллельными осям, по 1 магазину в каждой вершине.
12. Небольшой квадрат со сторонами, параллельными осям, по 25 магазинов в каждой вершине.
13. Небольшой квадрат со сторонами, параллельными осям, по 25000 магазинов в каждой вершине.

14. Большой квадрат со сторонами, параллельными осям, по 1 магазину в каждой вершине.
15. Большой квадрат со сторонами, параллельными осям, по 25 магазинов в каждой вершине.
16. Большой квадрат со сторонами, параллельными осям, по 25000 магазинов в каждой вершине.
17. Небольшой квадрат со сторонами, наклонёнными под 45°, по 1 магазину в каждой вершине.
18. Небольшой квадрат со сторонами, наклонёнными под 45°, по 25 магазинов в каждой вершине.
19. Небольшой квадрат со сторонами, наклонёнными под 45°, по 25000 магазинов в каждой вершине.
20. Большой квадрат со сторонами, наклонёнными под 45°, по 1 магазину в каждой вершине.
21. Большой квадрат со сторонами, наклонёнными под 45°, по 25 магазинов в каждой вершине.
22. Большой квадрат со сторонами, наклонёнными под 45°, по 25000 магазинов в каждой вершине.

23–29. Случайные тесты: мало магазинов, сосредоточенных в небольшом квадрате.
30–36. Случайные тесты: много магазинов, сосредоточенных в небольшом квадрате.
37–43. Случайные тесты: мало магазинов, сосредоточенных в большом квадрате.
44–50. Случайные тесты: много магазинов, сосредоточенных в большом квадрате.

Видно, что лобовой алгоритм получит 40–50% от полного балла.

11.4. «Дискретное показательное уравнение». *Часто операции с числами ведутся по модулю какого-либо числа p, например, $a \uparrow\!\!\uparrow_p b = a^b \bmod p$. Напишите программу, которая для заданных величин p и a будет находить все решения b_i и c_j уравнения $a \uparrow\!\!\uparrow_p b_i = c_j \bmod p$, выбираемые из заданных множеств. Нужно разделить исходные множества чисел b и c на пары соответствующих друг другу наборов: любое число b и любое число c из пары наборов удовлетворяют указанному уравнению.*

Данная задача, по мнению программного комитета, является достаточно сложной, поскольку для своего оптимального решения требует знания специфических структур данных.

Идейно задача не слишком сложна. Нужно сгруппировать значения b_i с одинаковыми значениями $a \uparrow\!\!\uparrow_p b_i$ (то есть с одинаковыми остатками $a^{b_i} \bmod p$), для чего для каждого b_i надо вычислить это значение. Понятно, что лобовое возведение в степень на основе многократного умножения не пройдёт по времени для указанных ограничений на b_i. Здесь поможет алгоритм *быстрого возведения в степень*:

$$a^n = \begin{cases} n = 2k, & (a^2)^k, \\ n = 2k+1, & (a^2)^k \cdot a. \end{cases}$$

На каждой итерации степень уменьшается вдвое, так что сложность одного такого возведения в степень есть $O(\log B)$, где B — ограничение на величину степени.

Второй вопрос: как быстро группировать каждое очередное значение b_i, для которого вычислено значение $a \uparrow\!\!\uparrow_p b_i$. Из-за того, что p слишком велико, невозможно хранить списки b_i для каждого возможного значения остатка от деления на p. Здесь полезной будет структура данных под названием *словарь* (или *ассоциативный массив*). Она хранит значения, для доступа к которым используется индекс, также называемый *ключом*, вообще говоря, не являющийся последовательными целыми числами, а берущий свои значения из любого множества, на котором определён порядок (то есть имеется операция сравнения больше-меньше). Эта структура позволяет добавить, удалить элемент, а также проверить наличие элемента с заданным ключом (а следовательно, и получить доступ к такому элементу) за время $O(\log n)$, где n — количество элементов в хранилище. В языках программирования соответствующий тип называется `map`, или `Dictionary`, или `SortedDictionary`.

Соответственно, в нашем случае разумно воспользоваться словарём, ключом в котором является остаток от деления на p, а элемент хранит два списка (массива): список тех b_i, которые дают этот остаток при вычислении $a \uparrow\!\!\uparrow_p b_i$, и список c_j, которые дают этот остаток при делении на p. После группировки всех b_i и c_j можно, перебирая элементы словаря, вывести соответствующие друг другу наборы чисел b и c.

Общая сложность такого алгоритма есть $O\big(n \log B + (n+m)\log(n+m)\big)$. Первое слагаемое есть время вычисления $a \uparrow\!\!\uparrow_p b_i$ для всех b_i, второе — время помещения всех чисел b_i и c_j в словарь, а потом извлечение их оттуда и вывод.

Идеи тестов:

1. $m = n = 1$, $b_1 = 10$, нет решений.
2. $m = n = 1$, $b_1 = 10$, есть решение.
3. $m = n = 1$, $b_1 = 10^9$, нет решений.
4. $m = n = 1$, $b_1 = 10^9$, есть решение.
5. $m = n = 100$, $B = 10$, все b имеют парные c, все c имеют парные b.
6. $m = n = 100$, $B = 10$, есть значения b без пары c, все c имеют парные b.
7. $m = n = 100$, $B = 10$, все b имеют парные c, есть значения c без пары b.
8. $m = n = 100$, $B = 10$, есть значения b без пары c, есть значения c без пары b.

9–12. То же, что в тестах 5–8, только $m = 20000$.

13–16. То же, что в тестах 5–8, только $m = n = 20000$.

17–20. То же, что в тестах 5–8, только $m = 20000$, $B = 10^9$.

21–24. То же, что в тестах 5–8, только $m = n = 20000$, $B = 10^9$.

25. Максимальный тест: $n = m = 10^5$, $B = 10^9$.

11.5. «Ночёвка на трассе». *Летом Петя Торопыжкин в составе компании из n друзей поучаствовал в мотопробеге в Европу (каждый участник ехал на своём мотоцикле). К концу одного из дней пробега, когда пришло время вставать на ночёвку,*

они были на платной дороге, состоящей из l отрезков. Для i-го участника пробега известен номер x_i отрезка дороги, на котором он находится. Стоимость проезда по k-му отрезку равна d_k, не зависит от направления проезда и взимается при въезде на этот отрезок; дальнейшее движение по отрезку не требует затрат. Во время подготовки к поездке было выяснено, что на этой дороге имеется m мотелей. Мотель с номером j расположен на отрезке дороги с номером y_j (на одном участке дороги может быть один мотель, несколько мотелей или не быть вовсе) и может вместить p_j постояльцев. Общая вместимость всех мотелей достаточна, чтобы все друзья могли заночевать под крышей. По имеющимся данным нужно определить минимальные затраты на дорожные сборы, чтобы каждый из друзей доехал до какого-нибудь мотеля.

По мнению программного комитета данная задача является очень сложной, поскольку для своего решения требует весьма нетривиального применения принципа динамического программирования и достаточно необычных структур данных.

Обозначим через $(a, b]$ совокупность всех целых чисел, лежащих между числами a и b, включая b и исключая a. При $a < b$ имеем $(a, b] = \{a + 1, a + 2, \ldots, b - 1, b\}$; при $a > b$ получаем $(a, b] = \{b, b + 1, \ldots, a - 2, a - 1\}$; при $a = b$ положим $(a, b] = \varnothing$. Тогда стоимость переезда с участка дороги с номером a на участок дороги с номером y, рассмотренная как функция от y, есть

$$f_a(y) = \sum_{k \in (a,y]} d_k. \qquad (*)$$

Перед началом работы алгоритма отсортируем мотели и участников по возрастанию номеров участков дороги, на которых они находятся. Если несколько мотелей (или участников) находятся на одном участке дороги, то среди них нас устроит любое упорядочение. В дальнейшем, когда говорится о переборе объектов по возрастанию или по убыванию, имеется в виду именно этот порядок.

Пусть $\mathtt{dp}[i, j]$ — минимальные суммарные затраты первых i участников, которые разместились в первых j мотелях. (Каждая ячейка массива $\mathtt{dp}$ — 8-байтное целое число.) Если такое размещение невозможно, то есть если суммарная вместимость первых j мотелей меньше i, полагаем эту величину равной $+\infty$ (каким-то числом, заведомо бóльшим рассматриваемых затрат, например, $2 \cdot 10^{18}$); точнее, в процессе вычислений эта ячейка будет оставаться бесконечной. При этом считаем, что какое-то ненулевое количество последних по счету участников (хотя бы i-й) разместилось в j-м мотеле.

В качестве начальных условий считаем $\mathtt{dp}[0, j] = 0$ для всех $0 \leqslant j \leqslant m$: размещение нуля участников в любом числе мотелей не стоит ничего. Также считаем $\mathtt{dp}[i, 0] = +\infty$ для всех $1 \leqslant i \leqslant n$: размещение ненулевого количества участников в нуле мотелей невозможно. Остальные ячейки массива $\mathtt{dp}$ также положим равными $+\infty$.

Пересчёт на шаге динамического программирования:

$$\mathtt{dp}[i, j] = \min_{1 \leqslant p \leqslant \min\{c_j, i\}} \Big\{ \sum_{k=i-p+1}^{i} f_k(y_j) + \min_{0 \leqslant r \leqslant j-1} \mathtt{dp}[i - p, r] \Big\}. \qquad (**)$$

Смысл этого выражения в следующем. Мы ищем минимальное по стоимости размещение первых i участников по первым j мотелям так, чтобы в j-м мотеле хоть кто-то жил. Это внешний минимум. Переменная p, по которой этот минимум ищется — это количество участников, заселяемых в j-й мотель: она строго больше 0 — кто-то в j-м мотеле жить должен — и не превосходит ни вместимости j-го мотеля c_j, ни общего числа размещаемых участников i.

Минимизируемое выражение в фигурных скобках имеет следующий смысл. Первое слагаемое — суммарные затраты p последних участников на доезд до j-го мотеля, куда их селят. Второе слагаемое — наиболее дешёвое размещение остальных участников по первым $(j-1)$-му мотелю.

Пересчёт в (**) ведётся внешним циклом по j и для каждого j для всех i. После того, как цикл по j от 1 до m завершён, ответом будет $\min\big\{\mathtt{dp}[n,j] : 1 \leqslant j \leqslant m\big\}$.

Сложность «лобовой» реализации этого вычисления следующая: мы проводим вычисления для каждой из $n \cdot m$ ячеек массива $\mathtt{dp}$; каждое вычисление внешнего минимума требует $O(C)$ операций ($C = \max c_j$), каждая из которых требует C операций для вычисления суммы в первом слагаемом, причём $f_k(y_j)$ тоже требует до l сложений, и m операций для поиска минимума во втором слагаемом. То есть общая сложность есть $O\big(nmC(Cl+m)\big)$, что слишком много в указанных ограничениях. Лобовое вычисление нужно оптимизировать.

Достаточно просто оптимизируется вычисление выражения внутри минимума. Оптимизация второго слагаемого производится хранением дополнительного массива $\mathtt{MinDP}[i] = \min\big\{\mathtt{dp}[i,r] : 0 \leqslant r \leqslant j\big\}$. После того, как для какого-то очередного значения j элементы массива $\mathtt{dp}[i,j]$ просчитаны для всех i, делается ещё один проход по i и вычисляется $\mathtt{MinDP}[i] = \min\big\{\mathtt{MinDP}[i], \mathtt{dp}[i,j]\big\}$. Это снижает сложность вычисления второго слагаемого до константы: для $j-1$ значения максимума просчитаны и помещены в массив $\mathtt{MinDP}$.

Вычисление слагаемых $f_k(y_j)$ есть суммирование отрезка массива $d[j]$. Весьма известным методом оптимизации этой операции является введение нового массива

$$\mathtt{D}[j] = \sum_{k=1}^{j} d_k;\ \text{рекурсивное определение: } \mathtt{D}[1] = d_1,\ \mathtt{D}[k+1] = \mathtt{D}[k] + d_{k+1},$$

который называется массивом *префиксных сумм набора* d_k. Тогда (*) можно переписать в виде

$$f_a(y) = \sum_{k \in (a,y]} d_k = \begin{cases} y > a, & d_{a+1} + d_{a+2} + \ldots + d_y = \mathtt{D}[y] - \mathtt{D}[a], \\ y = 0, & 0, \\ y < a, & d_{a-1} + d_{a-2} + \ldots + d_y = \mathtt{D}[a-1] - \mathtt{D}[y-1]. \end{cases}$$

Здесь удобно считать, что $\mathtt{D}[0] = 0$. Таким образом, вместо сложности $O(Cl)$ вычисления первого слагаемого в (**) имеем только $O(C)$.

То есть в итоге сложность уменьшена до $O\big(nmC(C+1)\big) = O(nmC^2)$.

В принципе, вычисление суммы $f_k(y_j)$ так же можно оптимизировать при помощи массива префиксных сумм. Перед началом цикла по i при очередном значении j можно предпросчитать массив $\mathtt{F}$, в котором $\mathtt{F}[k]$ есть сумма всех $f_q(y_j)$ для $1 \leqslant q \leqslant k$. Тогда сложность вычисления выражения под минимумом станет константной, а общая сложность алгоритма — $O(nmC)$. Впрочем для построения наиболее оптимального алгоритма это улучшение не требуется; оно будет включено в алгоритм автоматически.

Дальнейшая оптимизация связана одновременно с упрощением вычисления внешнего минимума и суммы слагаемых $f_k(y_j)$. Рассмотрения будем проводить для фиксированного j, поэтому в следующих рассуждениях для компактности записи будем опускать аргумент у f_k и аргумент, по которому берётся минимум. Рассмотрим выражение, стоящее внутри фигурных скобок.

Пусть сначала $i \leqslant c_j$. Если $p = 1$, то это выражение имеет вид

$$\sum_{k=i-1+1}^{i} f_k + \min \mathtt{dp}[i-1, r] = f_i + \mathtt{MinDP}[i-1]. \qquad (***)$$

При $2 \leqslant p \leqslant i$ можно проделать следующие преобразования:

$$\sum_{k=i-p+1}^{i} f_k + \min \mathtt{dp}[i-p, r] = f_i + \sum_{k=i-p+1}^{i-1} f_k + \min \mathtt{dp}[i-p, r] =$$

$$= f_i + \sum_{k=(i-1)-(p-1)+1}^{i-1} f_k + \min \mathtt{dp}[(i-1)-(p-1), r] = \Big[i' = i-1,\ p' = p-1\Big] =$$

$$= f_i + \sum_{k=i'-p'+1}^{i'} f_k + \min \mathtt{dp}[i'-p', r].$$

Отметим, что $1 \leqslant p' \leqslant i' = i - 1$.

Сравнивая начало преобразования и его конец, можно заметить, что форма выражений совпадает. Кроме того, диапазон изменения p' соответствует диапазону, по которому берётся внешний минимум в (**), для $i' = i - 1$. Стало быть, можно сделать вывод, что при вычислении минимума при каком-то i мы имеем дело с теми же самыми числами, что и при работе с $i - 1$, только увеличенными на f_i, к которым ещё добавлена величина (***), где также фигурирует слагаемое f_i.

Эти рассуждения приводят к необходимости иметь структуру данных, которая позволяет:

1) добавлять числа в себя;
2) быстро получать минимум из имеющегося в данный момент набора;
3) быстро увеличивать все числа в наборе на указанную величину.

Структура данных, реализующая пункты 1) и 2) известна, она называется *очередь с минимумами* (см., например, `http://e-maxx.ru/algo/stacks_for_minima`, раздел «Модификация очереди. Способ 2»).

Для реализации пункта 3) требуется следующая модификация. Дополнительно храним в структуре поле `off`, которое содержит аккумулированное значение прибавленных величин; в начале `off` $= 0$. При вызове метода увеличения всех элементов на величину `val` набора делаем `off` $\leftarrow$ `off` $+$ `val` без работы с элементами очереди. При добавлении элемента v в очередь реально заносится величина $v -$ `off`. При запросе минимума возвращается сумма минимума, полученного из модификации очереди, и текущего значения `off`. При извлечении элемента из очереди возвращается величина `head` $+$ `off`, где `head` — элемент из головы очереди. (Впрочем, в нашем случае нам будет достаточно операции удаления головного элемента из очереди без получения его значения.)

Тогда работа цикла по i, если $i \leqslant c_j$, выглядит следующим образом. Пусть `queue` — очередь, модифицированная для быстрого поиска минимума по всем своим элементам и добавления ко всем своим элементам заданной величины. В начале цикла по i она опустошается. При $i = 1$ имеем $\mathtt{dp}[1, j] = f_1(y_j)$. Эта же величина добавляется в `queue`. При переходе в цикле к следующему значению i в очередь добавляется величина $\mathtt{MinDP}[i-1]$, после чего вызывается метод, увеличивающий все элементы в очереди $f_i(y_j)$. После чего из очереди получаем значение минимума по всем её элементам, которое записывается в $\mathtt{dp}[i, j]$.

Теперь перейдём к анализу случая, когда $i \geqslant c_j$. В этом случае при переходе к $i+1$ не происходит увеличение количества величин, по которым в (**) берётся внешний минимум, но происходит изменение их состава. Так же в набор вводится величина (***), но при этом удаляется величина, которая соответствовала $p = c_j$ на предыдущем шаге, то есть величина, которая была введена в набор раньше остальных. Соответственно, нужно следующим образом изменить алгоритм, изложенный в предыдущем абзаце: при переходе в цикле к следующему значению i, если $i > c_j$, из очереди `queue` извлекается головной элемент, после чего в очередь `queue` добавляется величина $\mathtt{MinDP}[i-1]$, затем вызывается метод, увеличивающий все элементы в очереди на $f_i(y_j)$.

Таким образом, в итоге имеем алгоритм сложности $O(nm)$.

Весьма существенными являются затраты памяти на хранение массива `dp`: он содержит до $5000 \times 5000 = 25\,000\,000$ ячеек, каждая по 8 байт, то есть всего $200\,000\,000$ байт. Остальные структуры для своего хранения требуют существенно меньше памяти, так что ограничение по памяти в 256 Мб выполняется.

Идеи тестов:

1. $n = 1$, $m = 1$, $x_1 = y_1$.
2. $n = 1$, $m = 1$, $x_1 \neq y_1$.
3. $n = 20$, $m = 1$, все $x_i = y_1$.
4. $n = 20$, $m = 1$, некоторые $x_i = y_1$.
5. $n = 20$, $m = 1$, все $x_i \neq y_1$.
6. $n = 1$, $m = 20$, x_1 совпадает с несколькими y_j.
7. $n = 1$, $m = 20$, x_1 совпадает с одним y_j.
8. $n = 1$, $m = 20$, x_1 не совпадает ни с одним y_j.

9. $n = 20$, $m = 20$, все y_j совпадают, $c_j \geqslant 20$, все x_i совпадают.
10. $n = 20$, $m = 20$, все y_j различны, $c_j \geqslant 20$, все x_i совпадают.
11. $n = 20$, $m = 20$, все y_j совпадают, $c_j \geqslant 20$, все x_i различные.
12. $n = 20$, $m = 20$, все y_j различны, $c_j \geqslant 20$, все x_i различные.
13. $n = 20$, $m = 20$, все y_j совпадают, $c_j < 20$, все x_i совпадают.
14. $n = 20$, $m = 20$, все y_j различны, $c_j < 20$, все x_i совпадают.
15. $n = 20$, $m = 20$, все y_j совпадают, $c_j < 20$, все x_i различные.
16. $n = 20$, $m = 20$, все y_j различны, $c_j < 20$, все x_i различные.

17–22. Случайные тесты, которые обрабатываются наивной реализацией динамического программирования: $n, m, c_j < 50$.

23–30. Случайные тесты, которые обрабатываются реализацией с первичной оптимизацией минимизируемого выражения: $n, m, c_j < 170$.

31–38. Случайные тесты, которые обрабатываются реализацией с глубокой оптимизацией минимизируемого выражения: $n, m, c_j < 1000$.

39–45. Случайные тесты, которые обрабатываются оптимальным алгоритмом: $n, m, c_j < 4000$.

46–50. Максимальные случайные тесты $n, l, m = 5000$, $c_j \approx 50$.