Задача A. Мыльные пузыри *(Division 2)*

Имя входного файла:	`bubbles.in`
Имя выходного файла:	`bubbles.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

У Хексы день рождения!

Она решила порадовать себя совершенно необычным образом: надуванием мыльных пузырей. Мыльный пузырь представляет собой окружность на плоскости.

Для начала выбирается точка на оси Ox — центр надувательства. Да-да, все мыльные пузыри должны иметь один общий центр!

Затем последовательно надуваются пузыри: i-й пузырь надувается до тех пор, пока не коснётся точки (x_i, y_i), либо пока не упрётся в другой — уже надутый — пузырь. Если один мыльный пузырь коснётся другого, то вся конструкция взорвётся мириадами фантасмагорически иллюминирующих брызг, чего никак нельзя допустить!

Интересно, а сколько существует порядков надувания мыльных шаров, при которых не происходит то, чего никак нельзя допустить?

Формат входных данных

В первой строке содержится n — количество надуваемых мыльных пузырей ($1 \leqslant n \leqslant 1000$). В последующих n строках записаны разделённые пробелами пары чисел: i-я пара содержит координаты i-й точки (x_i, y_i) ($1 \leqslant x_i, y_i \leqslant 100$).

Формат выходных данных

Выведите одно число: количество допустимых порядков надувания мыльных шаров.

Примеры

bubbles.in	bubbles.out
2 3 3 7 3	2
3 1 1 2 4 9 2	4

Пояснения к примерам

В первом примере можно надувать пузыри как в порядке $(1, 2)$ (например, находясь в координате $x = 0$), так и в порядке $(2, 1)$ (например, находясь в координате $x = 10$).

Во втором примере возможные порядки — $(1, 2, 3)$, $(2, 1, 3)$, $(2, 3, 1)$ и $(3, 2, 1)$.

Задача B. Drop7 *(Division 2)*

Имя входного файла:	`drop7.in`
Имя выходного файла:	`drop7.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Каждый будний день Жене приходится ездить на метро из дома до работы и обратно. В метро плохо ловит интернет, поэтому Женя коротает время за играми на своём iPad. Его любимая игра называется Drop7.

Даже если вы играли в игру Drop7, рекомендуется внимательно прочитать описание правил. Правила игры в этой задаче немного отличаются от правил оригинальной игры.

Игровое поле представляет собой прямоугольник: 8 клеток в высоту и 7 клеток в ширину. Каждая клетка поля либо пустая, либо содержит *диск* с цифрой от 1 до 7. Каждый диск имеет ноль, один или два защитных слоя. На диски действует сила тяжести, направленная сверху вниз. Поэтому, если клетка под диском пустая, диск перемещается вниз.

Игрок каждым ходом сбрасывает один диск в один из семи столбцов поля.

Диски на игровом поле могут взрываться, это происходит по следующим правилам. Диск с числом X *без защитных слоёв* детонирует и взрывается, если он находится в горизонтальном и/или вертикальном ряду ровно из X *подряд идущих* дисков. Детонация дисков происходит только в тот момент, когда ни один диск не двигается под воздействием силы тяжести. Все диски, удовлетворяющие критерию выше, сначала одновременно детонируют, а затем одновременно взрываются. Когда диск взрывается, диски в соседних по стороне клетках теряют один защитный слой. При этом воздействия от взрывов суммируются, то есть диск теряет столько защитных слоёв, в скольких соседних клетках произошёл взрыв. Диск не может потерять больше защитных слоев, чем у него есть. В частности, на диски без защитных слоёв соседние взрывы никак не влияют.

После взрыва всех сдетонировавших дисков некоторые диски могут прийти в движение под воздействием силы тяжести. Когда диски прекратят свое движение, может последовать следующая волна взрывов и падений. Это продолжается до тех пор, пока диски не перестанут падать и детонировать.

Игрок начинает игру на первом уровне. После совершения некоторого количества ходов игрок переходит на следующий уровень. При переходе на следующий уровень все диски на поле поднимаются на одну клетку вверх, а в самой нижней строке появляются семь новых дисков. После этого некоторые диски могут сдетонировать, что повлечёт очередную цепочку взрывов. Если какой-то диск оказывается в самой верхней строке, игра *сразу* заканчивается.

Итак, каждый ход игрока выглядит следующим образом:

- Игрок бросает новый диск.
- Если какой-то диск находится в самой верхней строке, то игра сразу заканчивается.
- Пока есть сдетонировавшие диски, происходит волна взрывов и последующих падений.
- Если игрок достиг следующего уровня, то снизу поля появляются новые диски.
- Если какой-то диск находится в самой верхней строке, то игра сразу заканчивается.
- Пока есть сдетонировавшие диски, происходит волна взрывов и последующих падений.

За каждый взорвавшийся диск игрок получает очки, причём количество очков зависит от номера волны, на которой этот диск взорвался. Волны нумеруются с единицы, после каждого сброшенного игроком диска нумерация начинается сначала, а при переходе на новый уровень нумерация волн *продолжается дальше*. За каждый взорвавшийся на i-й волне диск игрок получает $\frac{7i^3+72i^2-73i+36}{6}$ очков.

В игре есть два уровня сложности: Normal и Hardcore. При сложности Hardcore каждые 5 ходов игрок переходит на новый уровень и получает 17 000 очков за каждый новый уровень. При сложности Normal для перехода с первого на второй уровень игрок должен сделать 30 ходов, со второго

на третий — 29 ходов, ..., с 25-го на 26-й — 6 ходов. При достижении 26-го уровня переход на следующий уровень происходит каждые 5 ходов. На сложности Normal за каждый новый уровень игрок получает 7000 очков.

К сожалению, Женя не смог найти эту игру нигде, кроме AppleStore. Поэтому он пришёл к вам и для начала попросил вас написать эмулятор игры Drop7. А пока вы пишете код, Женя решил подготовить для вас тестовые данные. Он запустил новую игру, записал на бумажку начальное состояние поля и сделал несколько ходов, попутно записывая всю информацию, необходимую для восстановления игры. А именно, он записывал, куда и какой диск он бросил на каждом ходу, а при переходе на новый уровень он записывал, какие семь новых дисков появились на поле снизу. При этом, если после очередного хода был переход на новый уровень, он *обязательно* записывал появившиеся диски, даже если это был последний ход, который он сделал.

От вас требуется по записанной Женей информации вывести конечное состояние игры.

Формат входных данных

В первой строке задаётся уровень сложности игры, поэтому есть два варианта первой строки: «`Mode: Normal`» или «`Mode: Hardcore`». В следующих восьми строках задаётся игровое поле на момент начала игры. Каждая строка содержит семь символов. Символ «`.`» обозначает пустую клетку, символы «`1`»–«`7`» обозначают диск с соотвествующей цифрой без защитных слоёв, символы «`a`»–«`g`» обозначают диск с одним защитным слоем, а символы «`A`»–«`G`» — диск с двумя защитными слоями. Гарантируется, что в начале игры никакой диск не падает вниз, никакой диск не сдетонирует и никакой диск не находится в самой верхней строке.

Для компактности Женя записывал всю информацию о дальнейшем ходе игры в одну строку. Последняя строка содержит именно эту строку, будем называть её логом игры. Лог игры содержит от 1 до 10 000 символов и состоит только из символов «`1234567abcdefgABCDEFG`». Когда Женя делал ход, в лог записывалось два символа: первый символ определял диск, а второй — номер столбца слева направо, в который был сброшен этот диск. Например, запись «`b7`» означает, что Женя сбросил диск с цифрой 2 и одним защитным слоем в седьмой слева столбец. Когда Женя переходил на новый уровень, в лог записывались семь символов, описывающих новые добавленные снизу диски (первый символ — диск в первом слева столбце, последний символ — в семьмом слева столбце).

Формат выходных данных

При записи лога Женя мог допустить любые ошибки. Поэтому, если данный лог нельзя интерпретировать как корректную последовательность ходов, необходимо вывести сообщение об ошибке. Выведите строку «`Game log is not complete`», если лог является префиксом корректной записи какой-либо игры, или строку «`Error in game log at position` X», если префикс длины $X-1$ является префиксом корректной записи какой-либо игры, а префикс длины X — не является.

Если данный лог корректен, то выведите состояние игры в конце. Если Женя проиграл, вначале выведите строку «`Game is over!`». Далее выведите заработанное количество очков, максимальную за всю игру длину цепочки взрывов и достигнутый уровень. Далее выведите игровое поле на момент конца игры. Соблюдайте формат вывода из примера.

Примеры

drop7.in	drop7.out
Mode: Normal .... 3A..g.. 3314e2e2e2e2e2e2e2e2	Game is over! Score: 60 Longest chain: 2 Level reached: 1 .e..... .e..... .e..... .e..... .e..... .e..... .e..... .e..7..

drop7.in	drop7.out
Mode: Normal A1A1A1A1A1A1A1A1A1	Error in game log at position 17
Mode: Hardcore A1B	Game log is not complete
Mode: Hardcore A1B2C3D4E5	Game log is not complete
Mode: Hardcore 71g167f2b3BBbbbbbc436b232A5Abcdefg55	Score: 64418 Longest chain: 14 Level reached: 3 ....5..

Пояснения к примерам

Во втором примере Женя должен был проиграть за первые восемь ходов, что соответствует первым 16 символам лога игры. По окончании игры на поле не могли происходить никакие действия, поэтому и лог должен был закончиться после 16-го символа.

В третьем примере лог игры обрывается посередине описания очередного хода.

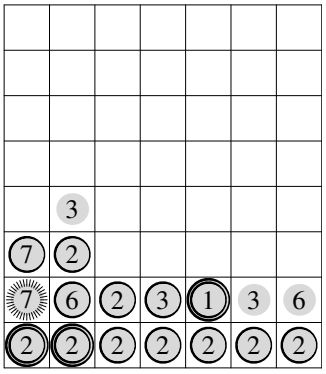
В четвёртом примере после первых пяти ходов Женя переходит на второй уровень, но в логе игры не хватает информации о том, какие диски появились в нижней строке.

В пятом примере в течение первых пяти ходов («71», «g1», «67», «f2», «b3») никакие диски не взорвались. После пятого хода Женя перешёл на второй уровень, и в нижней строке появились новые диски («BBbbbbb»). В следующие четыре хода («c4», «36», «b2», «32») по-прежнему ничего

не взорвалось. А вот десятым ходом («A5») Женя, сбросив диск с единицей с двумя защитными слоями в пятый слева столбец, запустил цепочку из 14 взрывных волн. При этом между 6-й и 7-й волной произошёл переход на третий уровень (появились новые диски: «Abcdefg»). Ниже приведены картинки, иллюстрирующие первые 8 взрывных волн и переход на третий уровень.

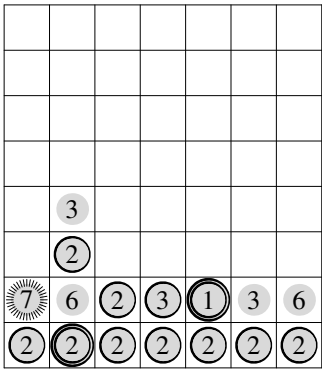
Score: 17000
Level: 2

Chain#1 +7



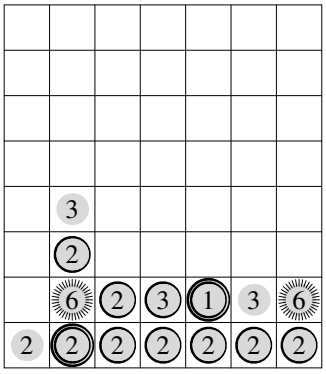
Score: 17007
Level: 2

Chain#2 +39



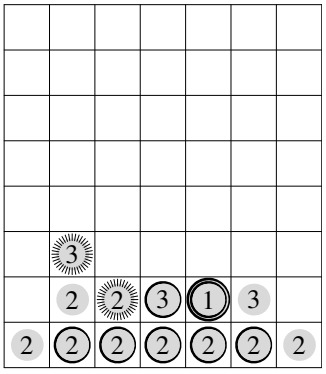
Score: 17046
Level: 2

Chain#3 +2x109



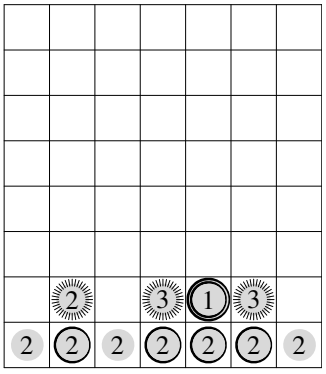
Score: 17264
Level: 2

Chain#4 +2x224



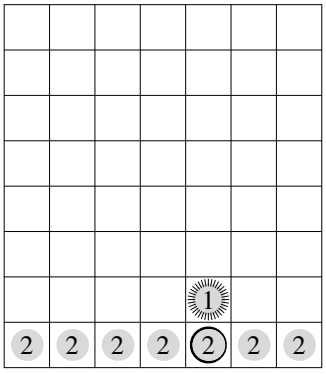
Score: 17712
Level: 2

Chain#5 +3x391



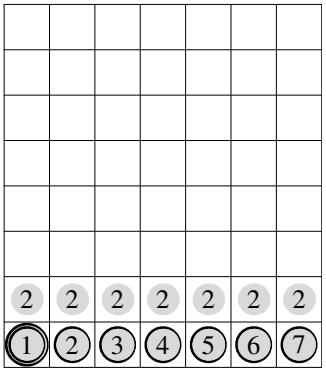
Score: 18885
Level: 2

Chain#6 +617



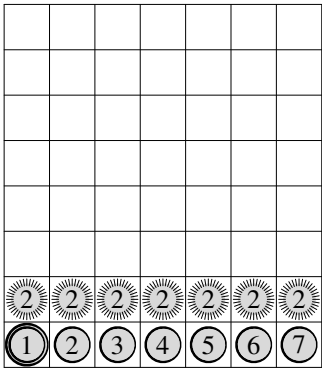
Score: 36502
Level: 3

Level up! +17000



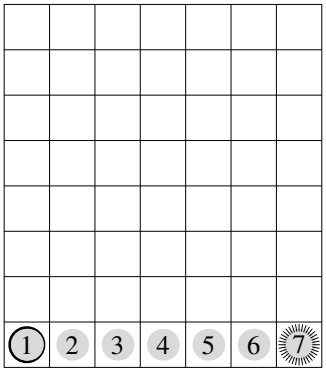
Score: 36502
Level: 3

Chain#7 +7x909



Score: 42865
Level: 3

Chain#8 +1274



Задача C. Эйлеровы графы *(Division 2)*

Имя входного файла:	`euler.in`
Имя выходного файла:	`euler.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

В ночь перед соревнованием Алиса читала книгу про графы и нашла в ней ряд интересных определений.

Графом G называется пара множеств вершин и рёбер (V, E), где множество рёбер E — это подмножество множества неупорядоченных пар вершин.

Циклом в графе (V, E) называется непустая последовательность рёбер $(u_1, v_1), \ldots, (u_m, v_m)$ в которой для каждого $i < m$ выполнено $v_i = u_{i+1}$, а $v_m = u_1$.

Эйлеровым циклом называется цикл в графе, который содержит каждое ребро один раз.

Граф называется эйлеровым, если он содержит эйлеров цикл.

Два графа $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ изоморфны, если существует такое биективное отображение $f : V_1 \to V_2$, что для любых v и u из V_1 $(v, u) \in E_1$ тогда и только тогда, когда $(f(v), f(u)) \in E_2$.

И вот, когда Алиса прочитала и поняла эти определения, ей пришёл в голову естественный вопрос: «Сколько существует попарно неизоморфных эйлеровых графов на n вершинах?»

Сможете ли вы помочь Алисе?

Формат входных данных

В первой строке задано число тестов T ($1 \leqslant T \leqslant 7$). В каждой из следующих T строк записано единственное число n ($1 \leqslant n \leqslant 7$).

Формат выходных данных

Для каждого заданного числа n выведите число попарно неизоморфных графов на n вершинах, взятое по модулю $10^9 + 7$.

Пример

euler.in	euler.out
2 3 4	5 11

Задача D. Серединка на половинку *(Division 2)*

Имя входного файла:	`halfgcd.in`
Имя выходного файла:	`halfgcd.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Как вам, без сомнения, известно, план развития периферийных районов Галактики предусматривает прокладку гиперкосмической трассы через вашу звёздную систему, и, к сожалению, ваша планета относится к числу подлежащих уничтожению. Процедура займёт не более двух земных минут. Спасибо.

— Простетник Вогон Джельц из Галактического Отдела Гиперкосмического Планирования

Как и все вогоны, Простетник Джельц любит числа и взрывать планеты. Сегодня удачный для него день — ему поручили уничтожить несколько планет для строительства гиперпространственной магистрали на их месте. Это чёрный день для жителей системы Скворншеллос не только из-за угрозы уничтожения, но и из-за того, что происходит парад планет — все планеты данной системы выстроились в ряд, создавая солнечное затмение на всех планетах.

Но у скворншеллоссцев всё ещё есть надежда — вогоны не могут просто так уничтожать планеты. Во-первых, существует указ, по которому во время парада планет они не могут уничтожать не последовательный набор планет в системе, чтобы не нарушать гармонию вселенной. Во-вторых, у каждой планеты есть свой радиус r_i, а Уничтожающий Лазер построен так, что он запустится только в том случае, если существует подмножество уничтожаемых планет, содержащее хотя бы половину из них и такое, что наибольший общий делитель всех их радиусов окажется строго больше единицы.

Помогите Простетнику Джельцу уничтожить как можно больше планет, соблюдая все вышеописанные условия.

Формат входных данных

Первая строка содержит одно целое число N — число планет в системе Скворншеллос ($1 \leqslant N \leqslant 1\,000$). Следующая строка содержит целые радиусы планет $r_1, \ldots, r_N$ в таком порядке, как они выстроились на параде планет ($1 \leqslant r_i \leqslant 10\,000$).

Формат выходных данных

Выведите два числа L и R — индексы самой левой и самой правой планет, которые нужно взорвать вогонам. Если правильных ответов несколько, то выведите любой. Если нельзя уничтожить ни одну планету, то выведите вместо этого два нуля.

Пример

halfgcd.in	halfgcd.out
5 1 2 1 2 1	2 5

Задача E. Следующее разбиение *(Division 2)*

Имя входного файла:	`next-partition.in`
Имя выходного файла:	`next-partition.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Рассмотрим все разбиения целого положительного числа N на K целых положительных слагаемых. Запишем каждое разбиение как последовательность чисел от больших слагаемых к меньшим. Отсортируем разбиения в обратном лексикографическом порядке. Найдите следующее разбиение N на K слагаемых в заданном порядке или определите, что его не существует.

Формат входных данных

В первой строке дано целое число K ($K \leqslant 10^5$, $K \leqslant N$, $1 \leqslant N \leqslant 10^9$). Во второй строке даны K целых чисел A_i — слагаемые разбиения ($1 \leqslant A_i \leqslant N$).

Формат выходных данных

Если следующее разбиение не существует, выведите -1. Иначе в первой строке выведите K. Во второй строке выведите K чисел — слагаемые следующего в обратном лексикографическом порядке разбиения.

Примеры

next-partition.in	next-partition.out
6 4 2 2 2 1 1	6 3 3 3 1 1 1
3 2 2 2	-1

Пояснения к примерам

В первом примере $N = 12$ и $K = 6$. Первое разбиение в обратном лексикографическом порядке — $12 = 7{+}1{+}1{+}1{+}1{+}1$, второе — $12 = 6{+}2{+}1{+}1{+}1{+}1$, 7-е — это данное разбиение $12 = 4{+}2{+}2{+}2{+}1{+}1$, следующее — $12 = 3 + 3 + 3 + 1 + 1 + 1$ и последнее 11-е разбиение — $12 = 2 + 2 + 2 + 2 + 2 + 2$.

Задача F. Уравнение *(Division 2)*

Имя входного файла:	`polynoms.in`
Имя выходного файла:	`polynoms.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Многочленом $P(x)$ над полем $\mathbb{F}_2$ называется выражение вида $\sum_{i=0}^{n} c_i \cdot x^i$, в котором каждое из чисел c_i равно либо 0, либо 1, а $c_n = 1$. Число $n = \deg(P)$ называется степенью многочлена. Все коэффициенты c_i при $i > n$ считаются равными нулю.

Произведением многочленов $P(x) = \sum_{i=0}^{n} p_i \cdot x^i$ и $Q(x) = \sum_{j=0}^{m} q_j \cdot x^j$ называется многчлен

$$S(x) = \sum_{i=0}^{n} \sum_{j=0}^{m} p_i \cdot q_j \cdot x^{i+j}.$$

Помните, что в $\mathbb{F}_2$ результат каждого сложения и умножения коэффициентов рассматривается исключительно по модулю 2.

Утверждение $P = 0$ считается верным, если все коэффициенты многочлена $P(x)$ равны нулю.

Даны два многочлена P и Q над полем $\mathbb{F}_2$. Необходимо найти такие два многочлена A и B, чтобы выполнялось утверждение $AP + BQ \neq 0$ и при этом степень многочлена $AP + BQ$ была минимальна.

Формат входных данных

В первой строке задано число тестовых случаев T ($1 \leqslant T \leqslant 100$). В каждой из следующих T пар строк задан один тестовый случай — два многочлена. Многочлен записывается в следующей форме: $n\ c_0\ c_1\ c_2\ \ldots\ c_n$. Все c_i равны 0 или 1, а $c_n = 1$.

Все заданные многочлены имеют положительную степень. Сумма степеней всех многочленов в одном тесте не превосходит 1000.

Формат выходных данных

Для каждого теста выведите два многочлена — сначала A, а затем B — в формате, аналогичном вводу. Если ответов несколько, вы можете вывести любой. Степени выведенных многочленов не должны превосходить $\max(10, 2(\deg(P) + \deg(Q)))$. Гарантируется, что для любых P и Q существует ответ, минимизирующий степень $AP + BQ$ и удовлетворяющий этим ограничениям.

Пример

polynoms.in	polynoms.out
3 2 1 1 1 1 1 1 4 1 0 0 0 1 2 1 0 1 4 1 0 1 0 1 3 1 1 0 1	0 1 1 0 1 0 0 0 1 2 0 1 1 3 1 0 1 1

Задача G. Головоломка «Обмен» *(Division 2)*

Имя входного файла:	`puzzle-swap.in`
Имя выходного файла:	`puzzle-swap.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Головоломка «Обмен» выглядит как доска 4×4, в клетках которой расставлены числа от 1 до 16, каждое по одному разу. Каждое число записывается ровно двумя десятичными цифрами: в числах 1–9 добавлены ведущие нули.

За одну операцию с головоломкой можно взять любые две клетки таблицы и поменять их местами.

Найдите любую кратчайшую последовательность операций, выполнив которую, можно получить упорядоченную таблицу:

01	02	03	04
05	06	07	08
09	10	11	12
13	14	15	16

Формат входных данных

Входные данные состоят из четырёх строк, соответствующих строкам таблицы. В каждой строке записано четыре числа, каждое из которых состоит ровно из двух десятичных цифр — для этого в числах 1–9 добавлены **ведущие нули**. Соседние числа в строке разделены пробелом. Гарантируется, что каждое из чисел от 1 до 16 встречается в заданной таблице ровно один раз.

Формат выходных данных

В первой строке выведите одно число — количество операций k. Это число должно быть минимально возможным. В следующих k строках выведите сами операции.

Для записи операций обозначим строки буквами `a`, `b`, `c`, `d` сверху вниз, а столбцы цифрами `1`, `2`, `3`, `4` слева направо. Операция обмена двух клеток записывается как r_1c_1-r_2c_2: строка и столбец одной клетки, знак «-» (минус, ASCII-код 45), строка и столбец другой клетки.

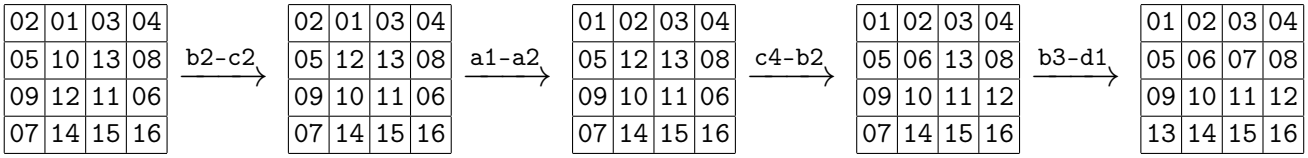
Если существует несколько возможных ответов, выведите любой из них.

Пример

puzzle-swap.in	puzzle-swap.out
02 01 03 04 05 10 13 08 09 12 11 06 07 14 15 16	4 b2-c2 a1-a2 c4-b2 b3-d1

Пояснение к примеру

Последовательность операций в примере и промежуточные состояния таблицы показаны ниже.



Задача H. Неправильное решето *(Division 2)*

Имя входного файла:	`sieve.in`
Имя выходного файла:	`sieve.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Многие из вас знают о решете Эратосфена. Это алгоритм, позволяющий найти все простые числа от 1 до некоторого N. Алгоритм устроен следующим образом.

Выпишем все числа от 1 до N. На первом шаге удалим все числа, кратные 2, кроме 2. На втором шаге удалим все числа, кратные 3, кроме 3. И далее, на k-м шаге удалим все числа, кратные $k+1$, кроме $k+1$ (которое, возможно, уже удалили раньше). Несложно показать, что после N шагов только простые числа и 1 останутся не удалёнными.

Рассмотрим небольшую модификацию этого алгоритма. На k-м шаге удалим каждое $(k+1)$-е из **оставшихся** чисел. Так, на первом шаге будут удалены все чётные числа. На втором — числа $5, 11, 17, \ldots$. И так далее. После выполнения бесконечного количества шагов последовательность будет начинаться как $1, 3, 7, 13, 19, 27, 39, 49, \ldots$.

Ваша задача состоит в том, чтобы проверить, есть ли данное число N в этой последовательности, и если есть — выдать его порядковый номер в ней, считая с единицы.

Формат входных данных

В первой строке содержится единственное число T — количетсво тестовых случаев ($1 \leqslant T \leqslant 50$). В следующих T строках записаны числа N_1, N_2, …, N_T, для которых необходимо решить задачу ($1 \leqslant N_i \leqslant 10^{12}$).

Формат выходных данных

Для каждого теста выведите одно число — номер числа N_i в последовательности или -1, если оно в ней не встречается.

Пример

sieve.in	sieve.out
5 1 2 3 42 1359	1 -1 2 -1 42

Задача I. Космический кот *(Division 2)*

Имя входного файла:	`space-cat.in`
Имя выходного файла:	`space-cat.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Коты — отличные космические путешественники, потому что благодаря их удивительной способности всегда приземляться на лапы они не боятся смены гравитации. Вот и наш солнечный Кот оказался в космосе с важной миссией — пробраться сквозь пространственный гипертоннель и принести людям *Трансцендентность*.

Пространственный гипертоннель является двумерным, у него есть пол и потолок. И пол, и потолок состоят из единичных сегментов разной высоты. Кот начинает в самом первом сегменте пола и должен оказаться в самом последнем сегменте пола.

На любом сегменте Кот может изменить гравитацию и спрыгнуть с пола на потолок или наоборот. На это действие требуется столько энергии, какова разница между высотами потолка и пола в данном сегменте.

А ещё Кот может переместиться в соседний сегмент. Но для этого высоты сегментов должны быть такими, чтобы Коту не пришлось карабкаться — это может быть опасно в гипертоннеле, да и просто лень. А вот просто идти или спрыгивать — запросто. Ну и, разумеется, наш пушистый супергерой не умеет проходить сквозь стены.

Необходимо определить минимальное количество энергии, которое потребуется для прохождения пространственного гипертоннеля.

Формат входных данных

В первой строке содержится единственное целое n — длина гипертоннеля ($1 \leqslant n \leqslant 10^5$). Во второй строке записаны n целых чисел, разделённых пробелами — высоты потолка c_i ($2 \leqslant c_i \leqslant 10^9$). В третьей строке записаны n целых чисел, разделённых пробелами — высоты пола f_i ($1 \leqslant f_i < c_i$).

Формат выходных данных

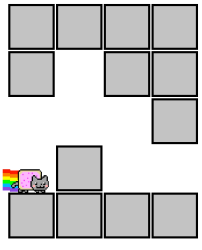
Выведите одно число: минимальное количество энергии, которое потребуется Коту для прохождения пространственного гипертоннеля, либо `-1`, если добраться от начала до конца невозможно.

Примеры

space-cat.in	space-cat.out
4 3 4 3 2 1 2 1 1	4
2 4 3 1 2	-1

Пояснения к примерам

В первом примере Коту нужно сменить гравитацию, затем сдвинуться с сегмента 1 в сегмент 2, затем снова сменить гравитацию и потом дойти до конца.



Во втором примере добраться до конца тоннеля по описанным правилам не представляется возможным.

Задача J. Вариация крестиков-ноликов *(Division 2)*

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Это интерактивная задача.

Дима играет с Петей в вариант игры «крестики-нолики». Для игры используется поле из 3×3 клеток, исходно каждая клетка пуста. Игроки делают ходы по очереди, первым ходит Дима. Каждый игрок в свой ход должен поставить свой символ (Дима ставит крестики, а Петя — нолики) в любую пустую клетку.

Дима выигрывает, если поставит три крестика в ряд: по горизонтали, вертикали или одной из двух диагоналей. Правила для ноликов отличаются от обычных: задача Пети — **не дать Диме построить ряд** из трёх крестиков. Формально Петя выигрывает, если во всех клетках доски уже стоят какие-то символы, но Дима ещё не выиграл.

Ваша задача — играть за Диму так, чтобы всегда выигрывать. Программа жюри будет играть за Петю.

Протокол взаимодействия

В каждом тесте вашей программе нужно сыграть за Диму от 1 до 100 партий в описанную игру. Тесты могут отличаться количеством партий и стратегией Пети. Каждая партия состоит из нескольких ходов.

В течение партии играющие программы обмениваются текущим положением на доске. Положение задаётся тремя строками, каждая из которых содержит по три символа. Символ «`x`» (английская маленькая буква икс) означает, что в соответствущей клетке стоит крестик, символ «`o`» (английская маленькая буква о) — нолик, а символ «`.`» (точка) говорит о том, что клетка пуста.

Каждая партия начинается с того, что программе участника на вход подаётся начальное положение на доске: все клетки пусты. Чтобы сделать ход, каждая из играющих программ должна вывести только что полученную позицию, изменённую в соответствии с правилами хода: ровно одна из пустых клеток должна стать клеткой с символом соответствующего игрока.

Партия заканчивается, когда один из игроков выигрывает. При этом, если выиграл Петя, проверка завершается с вердиктом «`Wrong Answer`». Если же выиграл Дима, игроки начинают следующую партию, если она запланирована.

В каждом тесте заранее определено, сколько партий будет сыграно и как будет играть Петя в каждой из них. Если все запланированные партии закончились победой Димы, то вместо пустой позиции, с которой начинается партия, программа участника получает на вход позицию, в которой все клетки **заняты крестиками**. Получив на вход такую позицию, программа участника должна корректно завершить свою работу.

Чтобы предотвратить буферизацию вывода, после каждой выведенной позиции следует вставить команду очистки буфера вывода: например, это может быть `fflush (stdout)` в C или C++, `System.out.flush ()` в Java, `flush (output)` в Pascal или `sys.stdout.flush ()` в Python.

Пример

стандартный ввод	стандартный вывод
... ... o.. .x. ... oxo .x. ... ... o.. .x. ... oo. .x. .x. ooo .x. xx. xxx xxx xxx	 ... .x. ... ox. .x. ... oxo .x. .x. ... .x. ... o.. .x. .x. oo. .x. xx. ooo .x. xxx

Пояснение к примеру

В примере, который также является первым тестом при проверке, необходимо сыграть две партии. В каждой партии Петя на каждом ходу ставит нолик в самую верхнюю пустую клетку. Если таких клеток несколько, он выбирает самую левую из них.

Обратите внимание: пустых строк во вводе и выводе на самом деле **нет**. Пропуски добавлены для того, чтобы все строки ввода и вывода шли в порядке времени, в которое они были переданы.

Задача K. Капитан Тарьян *(Division 2)*

Имя входного файла:	`treepaths.in`
Имя выходного файла:	`treepaths.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

— Ну вот, опять какая-то гробовая задача на деревья...
— Ну тогда давай позовём на помощь капитана Тарьяна!

Гипотетический диалог на контесте

Маленький Серёжа начал решать задачи на деревьях. Сначала он изучил алгоритм Ахо-Хопкрофта-Ульмана-Тарьяна, который быстро отвечает на запросы LCA в offline, но этого ему показалось мало. Он пошёл дальше и в одной из статей Тарьяна нашёл описание структуры heavy-light decomposition, которая специальным образом разбивает подвешенное дерево на вертикальные вершинно-непересекающиеся пути. Путь называется вертикальным, если мы можем обойти его, всегда путешествуя сверху вниз, то есть от корня к листьям. А после этого на получившихся путях можно считать всё, что душе угодно.

Серёже сразу же стало интересно, что произойдёт, если мы переподвесим дерево за другую вершину? Ведь тогда свойство путей может сломаться... Но Тарьян не так-то прост, вместе со Слейтором он открыл Splay-деревья, с помощью которых удалось реализовать структуру link/cut trees, а там одной из операций как раз является переподвешивание дерева за другой корень с корректным поддерживанием структуры вертикальных путей — то, что доктор прописал! Возможно, когда-то Тарьян тоже был маленьким, и его интересовали те же вопросы, что и нашего Серёжу?

Но вернёмся к делам насущным. Конечно же, Сережа стал изучать деревья не просто так — у него есть своё любимое дерево на N вершинах, к которому он хочет сделать M хитрых запросов вида «посчитай мне что-то на пути в дереве, ведущем из вершины u_i в вершину v_i». Как же обрабатывать такой запрос? Нужно разбить наш путь на несколько маленьких путей, которые полностью содержатся в каких-то путях из уже построенного разбиения, и получить всю необходимую информацию. Есть только одна проблема — не все рёбра попадают в наши пути! Если ребро соединяет две вершины из разных путей, то оно не лежит ни в одном из этих путей, и его придётся обрабатывать отдельно. Серёжа не любит крайние случаи, поэтому он хочет, чтобы суммарно таких рёбер в его запросах было как можно меньше.

Помогите Серёже построить разбиение его дерева на пути, которое бы минимизировало число рёбер из его запросов-путей, которые не лежат в разбиении. Известно, что Серёжа подвесил дерево за вершину с номером 1.

Формат входных данных

Первая строка содержит два числа N и M ($1 \leqslant N, M \leqslant 10^5$). В последующих $N - 1$ строках описывается исходное дерево: i-я из этих строк содержит два числа x_i и y_i — номера вершин, соединённых i-м ребром ($1 \leqslant x_i, y_i \leqslant N$). Затем в M строках описываются запросы к путям (u_i, v_i) в таком же формате.

Формат выходных данных

Выведите одно число — какое минимальное число рёбер придётся обрабатывать отдельно.

Пример

treepaths.in	treepaths.out
4 3 1 2 2 3 2 4 1 3 1 4 3 4	2

Задача L. Урок садоводства *(Division 2)*

Имя входного файла:	`unexpected-leaf.in`
Имя выходного файла:	`unexpected-leaf.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

У Хексы новое увлечение — садовоство. Выращивание деревьев — настоящая вирусная страсть!

Хекса нашла дерево, которое максимально точно описывает её тонкую душевную организацию. Это дерево состоит из целых n вершин и, как и множество других деревьев, является неориентированным графом с $n-1$ ребром, не содержащим циклов. Вирус решила вырастить точную копию этого дерева у себя в саду. И это у неё практически получилось.

Как известно, при выращивании своего декоративного дерева очень важно проконтролировать, чтобы не только выросло всё, что нужно, но ещё и не выросло то, что не нужно. К сожалению, к новому дереву случайно прирос одинокий лист. И всё было бы хорошо, если бы нумерация вершин сохранилась, но нет!

Теперь нужно найти тот самый лист. Получится?

Формат входных данных

В первой строке содержится n — количество вершин в исходном дереве ($1 \leqslant n \leqslant 100$). В последующих $n-1$ строках описывается исходное дерево: i-я из этих строк содержит два числа x_i и y_i — соединённые ребром вершины ($1 \leqslant x_i, y_i \leqslant n$). Затем в n строках описывается новое дерево из $n+1$ вершины в таком же формате. Гарантируется, что новое дерево получено из исходного путём добавления листа и перенумерацией вершин.

Формат выходных данных

Выведите единственное число — номер искомой вершины-листа. Если таких вершин несколько, то выведите ту, чей номер минимален.

Примеры

unexpected-leaf.in	unexpected-leaf.out
5 1 2 2 3 1 4 1 5 1 2 2 3 3 4 4 5 3 6	1
3 1 2 2 3 2 4 4 1 1 3	2
4 1 2 1 3 1 4 1 2 1 3 1 4 4 5	5