Задача A. Фабрика массивов

Имя входного файла:	`arrayfactory.in`
Имя выходного файла:	`arrayfactory.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Василий работает на фабрике по производству массивов. Фабрика производит только массивы из целых чисел длины n. Недавно специалисты отдела продаж выяснили, что покупатели предпочитают покупать массивы с суммой чисел не больше s, и решили продавать только такие массивы. Менять оборудование оказалось слишком дорого, гораздо проще брать уже готовые массивы и отрезать у них несколько элементов слева и справа так, чтобы сумма оставшихся элементов была привлекательной для покупателей.

Именно в этом заключаются обязанности Василия. При этом за более длинные массивы покупатели платят больше, поэтому Василий должен отрезать минимально возможное количество элементов, чтобы компания не терпела убытки. Работа эта очень нудная, поэтому Василию нужна ваша помощь.

Формат входных данных

В первой строке даны два целых числа n и s ($1 \leqslant n \leqslant 200\,000$, $-10^{18} \leqslant s \leqslant 10^{18}$).

Во второй строке даны n целых чисел a_1, a_2, …, a_n — элементы массива ($-10^9 \leqslant a_i \leqslant 10^9$).

Формат выходных данных

Если не существует непустого подотрезка массива с суммой элементов не больше s, выведите -1.

Иначе в первой строке выведите одно целое число — длину итогового массива. Во второй строке выведите два целых числа — количества элементов, отрезанных с начала и с конца массива соответственно.

Если существует несколько ответов, выведите тот, в котором количество элементов, отрезанных с начала, минимально.

Пример

arrayfactory.in	arrayfactory.out
5 3 2 4 -2 1 1	3 1 1

Задача B. Покупки и бонусы

Имя входного файла:	`bonuses.in`
Имя выходного файла:	`bonuses.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

В интернет-магазине Aweson действуют следующие правила оплаты покупок. Все покупки оплачиваются талерами — внутренней валютой Aweson. Когда новый покупатель регистрируется в системе, для него заводится бонусный счёт, на котором изначально находится 0 талеров. В момент каждой покупки у покупателя есть выбор: либо использовать бонусы, либо копить их. При использовании бонусов можно оплатить любую часть покупки талерами с бонусного счёта, а оставшуюся часть, если что-то осталось, оплатить обычными талерами. При накоплении же пользоваться бонусным счётом для оплаты нельзя, зато на него добавляется ровно 1% от стоимости покупки в талерах.

Михаил составил план своих покупок на Aweson. Теперь он хочет узнать, какой выбор нужно делать при каждой покупке и как их оплачивать, чтобы в итоге потратить как можно меньше обычных талеров.

Формат входных данных

В первой строке записано целое число n — количество покупок ($1 \leqslant n \leqslant 100$). Во второй строке заданы n целых чисел p_1, p_2, …, p_n через пробел — стоимости первой, второй, …, n-й покупок в талерах ($0 < p_i \leqslant 10^5$, все p_i делятся нацело на 100).

Формат выходных данных

В первой строке выведите одно число s — минимальное суммарное количество обычных талеров, с помощью которого Михаил может сделать все свои покупки в заданном порядке.

В следующих n строках выведите стратегию покупок, позволяющую потратить ровно s обычных талеров. Каждая из этих строк должна содержать два целых числа c_k и b_k, разделённых знаком «плюс» и отделённых от него пробелами: сколько обычных талеров и сколько талеров с бонусного счёта Михаилу следует потратить на k-ю покупку. Разумеется, эти числа должны быть неотрицательными, а их сумма должна быть равна стоимости k-й покупки. Если $b_k = 0$, то k-я покупка совершается с накоплением бонусов, а если $b_k > 0$ — с использованием бонусов. Конечно, ни в какой момент Михаил не может потратить бонусов больше, чем их есть, а сумма всех чисел c_k должна быть в точности равна s.

Количество талеров на бонусном счёте после всех n покупок не имеет значения. Если стратегий с минимальным s несколько, выведите любую из них.

Примеры

bonuses.in	bonuses.out
2 100 100	199 100 + 0 99 + 1
3 100 10000 100	10100 100 + 0 10000 + 0 0 + 100

Пояснения к примерам

В первом примере Михаил получает с первой покупки один бонусный талер. Следует потратить его для частичной оплаты второй покупки.

Во втором примере Михаил также получает с первой покупки один бонусный талер. Однако тратить его на второй покупке невыгодно. Вместо этого Михаил может получить ещё 100 талеров со второй покупки. Тогда третью покупку удастся полностью оплатить бонусными талерами, и на бонусном счету останется ещё один талер.

Задача C. Количество решений

Имя входного файла:	`c2sat.in`
Имя выходного файла:	`c2sat.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Дана логическая формула F от n переменных $x_1, x_2, \ldots, x_n$, имеющая вид $\bigwedge_{i=1}^{m} a_i \vee b_i$, где каждое из a_i и b_i равно либо одной из переменных, либо её отрицанию. Каждая переменная x_j может принимать два значения: истина и ложь.

Необходимо посчитать количество выполняющих наборов для этой формулы, то есть таких способов выбрать значения для всех x_j, что для всех i из a_i и b_i истинно хотя бы одно.

Посмотрите на пояснение к примеру для лучшего понимания.

Формат входных данных

В первой строке даны два целых числа n и m ($1 \leqslant n \leqslant 50$). В следующих m строках даны пары целых чисел a_i и b_i ($1 \leqslant |a_i|, |b_i| \leqslant n$). Положительным числом k от 1 до n обозначена переменная x_k, отрицательным числом k от -1 до $-n$ — отрицание $\neg x_{|k|}$.

Гарантируется, что для всех i выполнено $a_i < b_i$, а также что все пары (a_i, b_i) различны.

Формат выходных данных

Выведите одно целое число — количество выполняющих наборов формулы.

Пример

c2sat.in	c2sat.out
3 2 -2 1 2 3	4

Пояснение к примеру

Пример задаёт формулу $F(x_1, x_2, x_3) = (\neg x_2 \vee x_1) \wedge (x_2 \vee x_3)$. В таблице ниже представлены значения этой формулы во всех возможных случаях. Видно, что есть четыре выполняющих набора.

x_1	x_2	x_3	$\neg x_2 \vee x_1$	$x_2 \vee x_3$	F
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Задача D. Разрезание картошки

Имя входного файла:	`cut-potatoes.in`
Имя выходного файла:	`cut-potatoes.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Пётр хочет сварить картошку. У него есть n картофелин объёма a_1, a_2, ..., a_n, и все эти картофелины Пётр хочет сварить.

Пётр считает, что варить лучше картошку примерно одинакового объёма. Для этого он не против разрезать некоторые картофелины на части, но не хочет резать ни одну картофелину более чем на k частей.

Пётр придумал следующую формальную модель. Если в кастрюлю положить несколько картофелин или их частей, будем считать *неравномерностью* этого множества объектов отношение наибольшего из их объёмов к наименьшему. Помогите Петру выбрать, на какое количество равных частей разрезать каждую из его картофелин, чтобы неравномерность получившегося множества была как можно меньше.

Формат входных данных

В первой строке записаны два целых числа n и k — количество картофелин и максимальное количество частей, на которые можно разрезать одну картофелину ($1 \leqslant n, k \leqslant 100$). Во второй строке заданы n целых чисел a_1, a_2, ..., a_n через пробел — объёмы картофелин ($1 \leqslant a_i \leqslant 100$).

Формат выходных данных

Выведите в строке через пробел n целых чисел c_1, c_2, ..., c_n — на сколько равных частей нужно разрезать картофелины 1, 2, ..., n, чтобы неравномерность получившегося множества объектов была как можно меньше. Каждое c_i должно быть в пределах от 1 до k включительно. Имейте в виду, что разрезать картофелины можно на части с нецелым объёмом. Если возможных ответов с минимальной неравномерностью несколько, выведите любой из них.

Примеры

cut-potatoes.in	cut-potatoes.out
4 3 5 5 3 2	3 3 2 1
2 2 1 4	1 2

Пояснения к примерам

В первом примере картофелины имеют объёмы 5, 5, 3 и 2. Разрезать каждую из них Пётр может на одну, две или три части. Оптимальное решение — получить шесть частей объёма 5/3, две объёма 3/2 и одну объёма 2/1. Неравномерность равна $(2/1)/(3/2) = 4/3$.

Во втором примере Петру следует разрезать большую картофелину на две части, а маленькую не резать вовсе. Неравномерность равна 2/1.

Задача E. Разделяй и властвуй

Имя входного файла:	`dnc.in`
Имя выходного файла:	`dnc.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Виктория изучает метод «разделяй и властвуй». Для этого она рассматривает многоугольник, причём она выбирает не произвольный, а выпуклый многоугольник, у которого к тому же чётное количество вершин. Виктория знает, что, чтобы «разделять и властвовать», нужно сначала выбрать, как разделять. Поэтому она реализует следующую рекурсивную функцию `Fun`, которая получает на вход какое-то подмножество S вершин многоугольника (изначально — множество всех вершин многоугольника):

- Выберем из S две вершины A и B так, что $(x_A, y_A) < (x_B, y_B)$ (сначала сравниваются x-координаты, а если они равны — y-координаты).
- После этого разрежем S по отрезку между этими вершинами на две части: «левую», в которой лежат все такие точки (x, y), что $(x_B - x_A) \cdot (y - y_A) > (x - x_A) \cdot (y_B - y_A)$, и «правую», в которой, наоборот, $(x_B - x_A) \cdot (y - y_A) < (x - x_A) \cdot (y_B - y_A)$.
- Далее рекурсивно запустим функцию `Fun` для вершин из S, оказавшихся в «левой» части, а когда она закончит работать — рекурсивно запустим `Fun` для вершин из S, оказавшихся в «правой» части.
- Под конец в функции должна следовать часть, где результаты работы двух рекурсивных вызовов комбинируются для получения результата текущего вызова `Fun`, но пока Виктория решила для простоты опустить эту часть и сосредоточиться на разделении.
- Конечно, следует уделить внимание крайним случаям. Если `Fun` запущена от пустого множества вершин S, то она сразу заканчивает работу. Чтобы не разбирать случай одной вершины, Виктория решила, что после каждого разделения в каждой части должно оказываться чётное количество вершин.

Виктория решила посмотреть на разные стратегии разбиения. Для этого, как только очередной вызов `Fun` выбирает две вершины A и B, Виктория добавляет в специальный список их координаты. В конце список содержит все вершины по одному разу в каком-то порядке. Этот список и задаёт стратегию разбиения.

Представим себе, что все возможные стратегии разбиения данного многоугольника выписаны и упорядочены как последовательности чисел. Помогите Виктории узнать, какая стратегия имеет в этом порядке номер k. Стратегии нумеруются целыми числами, начиная с нуля.

Формат входных данных

В первой строке записано целое число n — количество вершин многоугольника ($4 \leqslant n \leqslant 30$, число n чётно).

В каждой из следующих n строк заданы два целых числа x и y через пробел — координаты очередной вершины ($|x|, |y| \leqslant 10\,000$). Вершины перечислены в порядке следования против часовой стрелки. Гарантируется, что никакие две вершины не совпадают и никакие три не лежат на одной прямой, а также что заданный многоугольник выпуклый.

Наконец, в следующей строке задано целое число k — номер стратегии, которую хочет увидеть Виктория. Стратегии нумеруются целыми числами, начиная с нуля. Гарантируется, что стратегия с требуемым номером существует.

Формат выходных данных

Выведите координаты n вершин в порядке их следования в записи k-й стратегии. Координаты каждой вершины следует выводить в отдельной строке.

Примеры

dnc.in	dnc.out
4 0 0 1 0 1 1 0 1 3	1 0 1 1 0 0 0 1
6 0 0 1 -1 2 0 2 1 1 2 0 1 8	0 0 2 1 0 1 1 2 1 -1 2 0

Пояснения к примерам

В первом примере стратегий всего четыре, и они имеют номера 0, 1, 2 и 3:
$(0,0)$, $(0,1)$; $(1,0)$, $(1,1)$;
$(0,0)$, $(1,0)$; $(0,1)$, $(1,1)$;
$(0,1)$, $(1,1)$; $(0,0)$, $(1,0)$;
$(1,0)$, $(1,1)$; $(0,0)$, $(0,1)$.

Во втором примере стратегия с номером 8 начинает с того, что разделяет вершины многоугольника отрезком $(0,0)$–$(2,1)$ на два подмножества: «слева» оказываются вершины $(0,1)$ и $(1,2)$, а «справа» — вершины $(1,-1)$ и $(2,0)$. В таком порядке они и попадают в запись стратегии.

Задача F. Удвоение

Имя входного файла:	`doubling.in`
Имя выходного файла:	`doubling.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Программа на языке «Удвоение» — это строка, в которой могут встретиться символы «`1`», «`[`» и «`]`». Результат выполнения такой программы — целое число. Программа строится и работает по следующим правилам:

- $R(\varepsilon) = 0$: результат выполнения пустой программы равен нулю.
- $R(\texttt{1}) = 1$: программа «`1`» даёт результат 1.
- $R(\texttt{[}A\texttt{]}) = 2 \cdot R(A)$: квадратные скобки удваивают результат программы внутри них.
- $R(AB) = R(A) + R(B)$: результат конкатенации двух программ равен сумме их результатов.

Можно показать, что такое рекурсивное определение однозначно определяет множество возможных программ и результаты их выполнения. Программа, не удовлетворяющая этим правилам, является некорректной, и результат её выполнения не определён.

Дано целое число n от единицы до миллиарда. Выведите кратчайшую программу, результат работы которой равен n. В случае нескольких возможных ответов выведите любой.

Формат входных данных

В первой строке записано целое число n — требуемое число ($1 \leqslant n \leqslant 10^9$).

Формат выходных данных

В первой строке выведите программу минимальной длины, результатом которой будет число n. Если таких программ несколько, выведите любую из них.

Примеры

doubling.in	doubling.out
1	1
10	[11111]

Пояснения к примерам

В первом примере программа «`1`» даёт результат 1.

Во втором примере программа «`11111`» даёт результат 5, а значит, программа «`[11111]`» даст требуемый результат 10. Есть и другие правильные ответы: «`[[11]1]`» и «`[1[11]]`».

Задача G. Новая коллекция

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Это интерактивная задача.

В коллекционной карточной онлайн-игре CCG вышло новое дополнение. Игроки могут покупать случайные карты дополнения по одной: каждая купленная карта равновероятно и независимо от других покупок окажется одной из карт нового дополнения.

Пётр — один из основных авторов CCG Scene — сайта игроков, куда попадают известные данные обо всех картах CCG, а также их достоинства, недостатки, стратегии использования и многое другое.

Один из важных параметров дополнения — сколько в нём всего карт. Чтобы поддержать свою репутацию в сообществе, Пётр хочет как можно скорее выложить эту информацию. Известно, что в каждом из предыдущих дополнений количество карт было целой степенью числа 10: от десяти до десяти миллионов включительно. Пётр предположил, что это правило соблюдается и в новом дополнении, и что количество карт с одинаковой вероятностью может оказаться каждой из возможных степеней десятки.

Ресурсы Петра позволяют ему купить до 10 000 карт. У каждой карты CCG есть идентификатор, но известно только, что эти идентификаторы одинаковые у совпадающих карт и разные у различных. Пётр может покупать карты по одной, и после каждой покупки узнаёт идентификатор только что купленной карты. Кроме того, в любой момент Пётр может прекратить покупки и написать на CCG Scene количество карт в новом дополнении.

Напишите программу, которая поможет Петру найти правильный ответ.

Протокол взаимодействия

Решение должно выводить каждое действие в стандартный поток вывода на отдельной строке. Действие имеет либо вид «`+`», означающий покупку очередной карты, либо вид «`=` n», означающий вывод ответа.

После каждой покупки в стандартный поток ввода поступает очередная строка — идентификатор только что купленной карты. Идентификатор состоит ровно из двенадцати шестнадцатеричных цифр (`0`–`9` и `A`–`F`), возможно, с ведущими нулями.

После вывода ответа решение должно сразу корректно завершить работу.

Чтобы предотвратить буферизацию вывода, после каждого выведенного запроса 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.

В каждом тесте к этой задаче зафиксировано количество карт n — целая степень числа 10 от 10^1 до 10^7 включительно. Гарантируется, что каждая купленная карта равновероятно и независимо от других покупок окажется одной из n возможных карт. Тем не менее, в каждом тесте заранее зафиксировано, моделируя это предположение, какие идентификаторы будут иметь первые 10 000 купленных карт.

Пример

действия участника	ответы проверяющей программы
+	13A17DA944F1
+	0BE186EC4732
+	1591FAA834F2
+	13A17DA944F1
+	13A17DA944F1
+	0E703FE27659
+	1A16F6A9EBB6
+	1591FAA834F2
+	0E703FE27659
+	07B569E7D7A4
+	13A17DA944F1
= 10	

Пояснение к примеру

Обратите внимание: **слева** указан **вывод** программы участника, а **справа** — то, что она после этого получает **на вход**.

В примере рассматривается первый тест в системе. Количество карт в дополнении в этом тесте равно 10, а их идентификаторы таковы:
07B569E7D7A4,
0A2001E19A1F,
0BE186EC4732,
0E703FE27659,
121657A5CA39,
13786BA38233,
13A17DA944F1,
1591FAA834F2,
189A2AAE360C,
1A16F6A9EBB6.

Решение сначала покупает одну за другой одиннадцать карт, при этом три карты встретились больше одного раза, а четыре карты вообще не встретились. После этого решение считает, что полученной информации достаточно, чтобы установить, что карт всего 10, и выводит ответ.

Задача H. Путь или раскраска

Имя входного файла:	`pathorcoloring.in`
Имя выходного файла:	`pathorcoloring.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

А вы любите NP-полные задачи? Например, посмотрим на следующие две задачи на графе:

Задача о раскраске. Дан неориентированный граф G и число k. Выберите для каждой вершины цвет $\phi(v)$ так, чтобы все цвета были целыми числами от 1 до k, и при этом у любых двух вершин, соединённых в графе ребром, цвета были различны.

Задача о простом пути длины k. Дан неориентированный граф G и число k. Найдите в графе простой путь длины k. Формально следует выбрать такую последовательность из $k+1$ *различных* вершин $v_1, v_2, \ldots, v_{k+1}$, что любые две соседние вершины в ней соединены ребром.

Вам даны неориентированный граф G и число k. Решите для них любую из этих двух задач на ваш выбор.

Формат входных данных

В первой строке задано целое число T — количество тестовых случаев, для которых необходимо решить задачу ($1 \leqslant T \leqslant 1000$). Описание каждого тестововго случая состоит из нескольких строк.

В первой строке описания каждого тестового случая даны два целых числа n и m — количество вершин и ребёр графа ($1 \leqslant n \leqslant 1000$, $1 \leqslant m \leqslant 10\,000$). В следующей строке дано целое число k ($1 \leqslant k \leqslant n$). В каждой из следующих m строк заданы даны два целых числа a и b — номера вершин, соединённых очередным ребром ($1 \leqslant a, b \leqslant n$).

Гарантируется, что во всех графах нет петель и кратных рёбер. Кроме того, гарантируется, что суммарное число рёбер во всех заданных графах не превосходит 100 000.

Формат выходных данных

Для каждого тестового случая выведите одно из трёх:

- Слово «`path`», а за ним — $k+1$ номеров вершин, составляющих простой путь длины k.
- Слово «`coloring`», а за ним — n чисел 1 до k, задающих раскраску графа в k цветов.
- Слово «`neither`», если в графе нет ни простого пути длины k, ни раскраски в k цветов.

Если в графе есть и путь, и раскраска, можно выдать либо путь, либо раскраску. Если путей или раскрасок несколько, можно вывести любой возможный путь или любую возможную раскраску, соответственно.

Пример

pathorcoloring.in	pathorcoloring.out
2 4 5 2 1 2 2 3 3 4 4 1 1 3 3 3 3 1 2 2 3 3 1	path 3 2 1 coloring 1 2 3

Задача I. Двойное перемешивание

Имя входного файла:	`shuffle-twice.in`
Имя выходного файла:	`shuffle-twice.out`
Ограничение по времени:	7 секунд
Ограничение по памяти:	512 мебибайт

Рассмотрим следующий линейный конгруэнтный генератор случайных чисел. У генератора есть состояние `state`, которое может быть любым целым числом от 0 до $2^{32} - 1$ включительно. Чтобы сгенерировать следующее случайное целое число от 0 до `range` $- 1$ включительно, используется функция `random`:

```
Function random (range):
    state := (state * 1664525 + 1013904223) mod 2^32;
    return (state * range) div 2^32;
```

Здесь `mod` и `div` — операции взятия остатка по модулю и целочисленного деления. Выбранные константы рекомендованы в книге Numerical Recipes. К примеру, если в какой-то момент `state` равно 12 345, то при вызове `random (100)` сначала `state` становится равным 87 628 868, а затем функция возвращает значение 2.

Чтобы случайным образом перемешать массив, используется функция `shuffle`:

```
Function shuffle (array):
    n := length of array;
    for i := 0, 1, 2, ..., n - 1:
        j := random (i + 1);
        array[i] <-> array[j];
    return array;
```

Здесь операция "`x <-> y`" меняет местами элементы `x` и `y`; если это один и тот же элемент массива, ничего не происходит. Массив индексируется целыми числами от 0 до $n - 1$ включительно. К примеру, если в какой-то момент `state` равно 12 345, то при вызове `shuffle` от массива `[1, 2, 3, 4, 5]` итоговое значение `state` будет равно 3 908 547 000, а функция вернёт массив `[2, 3, 4, 1, 5]`.

Фёдор выбрал целое число `seed` от 0 до $2^{32} - 1$ включительно. После этого он выполнил следующую программу:

```
state := seed;
n := 10000;
array := [1, 2, ..., n];
array := shuffle (array);
array := shuffle (array);
```

Фёдор держит в секрете значение `seed`, однако раскрыл содержимое массива `array` после выполнения этой программы. Выясните, что получится у Фёдора в массиве `array`, если он добавит в конец своей программы ещё одну строчку:

```
array := shuffle (array);
```

Формат входных данных

В первой строке записано целое число n — количество элементов в массиве (в этой задаче это число всегда равно 10 000). Во второй строке задана перестановка из n чисел от 1 до n через пробел — содержимое массива после выполнения программы Фёдора.

Формат выходных данных

В первой строке выведите перестановку из n чисел от 1 до n через пробел — содержимое массива

после выполнения программы с добавленной строчкой.

Пример

shuffle-twice.in	shuffle-twice.out
10000 8150 6681 695 ... 5947 4461	3254 6689 6294 ... 8486 8107

Пояснение к примеру

В примере значение `seed` равно 12 345. Полностью пример можно скачать на вкладке Sample Zip интерфейса тестирующей системы.

Задача J. Таймер

Имя входного файла:	`timer.in`
Имя выходного файла:	`timer.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Рассмотрим следующую конструкцию таймера. Таймер состоит из двух цилиндров: внутреннего и внешнего, который может вращаться вокруг внутреннего. На поверхность внутреннего цилиндра нанесены отметки и числа. Отметки соответствуют минутам. Внешний же цилиндр покрывает весь внутренний, оставляя видимым только маленькое окошко. Время, которое показывает таймер, можно узнать, посмотрев в центр этого окошка.

Всего на внутреннем цилиндре 60 отметок. Каждая пятая из них отмечена соответствующим числом: 0, 5, 10, 15, ..., 55. Расстояния между любыми двумя соседними отметками, включая расстояние между последней и первой отметками, равны. Ширина окошка ровно в пять раз больше расстояния между соседними отметками.

Исходно в центре окошка во внешнем цилиндре находится отметка 0 на внутреннем цилиндре. Чтобы выставить время на таймере, внешний цилиндр поворачивают вокруг внутреннего по часовой стрелке. После этого он медленно поворачивается против часовой стрелки, пока опять не достигнет отметки 0, после чего останавливается. На таймере можно выставить любое время, строго меньшее, чем 60 минут.

Анна учит своего домашнего робота Берту узнавать время, которое показывает таймер. Берта может фотографировать окошко таймера. Помогите Анне написать программу для Берты, чтобы определять время по фотографии.

Фотография окошка — это растровое изображение 60 пикселов в ширину и 12 пикселов в высоту. Будем для простоты считать, что фотография — это идеальное изображение видимой части внутреннего цилиндра, уложенной на плоскость. В таком случае расстояние между соседними отметками будет ровно 12 пикселов. Также для простоты предположим, что фотография сделана в момент, когда время, которое показывает таймер, кратно 5 секундам. Поскольку окошко не совсем прямоугольной формы, небольшие уголки на фотографии всегда закрыты внешним цилиндром; точную форму уголков можно увидеть в примере. Все остальные пикселы фотографии либо полностью белые, либо полностью чёрные.

При указанных предположениях каждая отметка выглядит как чёрный квадрат 2×2 в первых двух строках фотографии; настоящий центр отметки находится между двумя столбцами этого квадрата. Каждое число печатается чёрным шрифтом размера 8×8 в строках с 4-й по 11-ю. Каждое число (из одной или двух цифр) горизонтально выровнено так, что центр соответствующего блока размера 8×8 или 16×8 расположен по центру отметки, при которой написано это число. Все пикселы, не покрытые уголками, отметками или цифрами, отображаются как белые.

Ниже показаны изображения отдельных цифр от 0 до 9 в используемом шрифте 8×8. Сам шрифт идентичен шрифту, использовавшемуся в видеокартах *Rendition Vérité 1000* в 1990-е годы. Изображения цифр в шрифте можно также скачать на вкладке Sample Zip в интерфейсе тестирующей системы (файл `digits.txt` в каталоге задачи) :

```
.XXXXX.. ...XX... .XXXXX.. .XXXXX.. ...XXX.. XXXXXXX. .XXXXX.. XXXXXXX. .XXXXX.. .XXXXX..
XX..XXX. ..XXX... XX...XX. XX...XX. ..XXXX.. XX...... XX...XX. XX...XX. XX...XX. XX...XX.
XX.XXXX. .XXXX... .....XX. .....XX. .XX.XX.. XXXXXX.. XX...... X....XX. XX...XX. XX...XX.
XXXX.XX. ...XX... ...XXX.. ..XXXX.. XX..XX.. .....XX. XXXXXX.. ....XX.. .XXXXX.. .XXXXXX.
XXX..XX. ...XX... .XXX.... .....XX. XXXXXXX. .....XX. XX...XX. ...XX... XX...XX. .....XX.
XXX..XX. ...XX... XX...... XX...XX. ....XX.. XX...XX. XX...XX. ..XX.... XX...XX. XX...XX.
.XXXXX.. .XXXXXX. XXXXXXX. .XXXXX.. ...XXXX. .XXXXX.. .XXXXX.. ..XX.... .XXXXX.. .XXXXX..
........ ........ ........ ........ ........ ........ ........ ........ ........ ........
```

Формат входных данных

Входные данные состоят ровно из 12 строк, каждая из которых содержит ровно 60 символов: фотография окошка таймера. Поскольку окошко не совсем прямоугольной формы, небольшие уголки на фотографии всегда закрыты символами «-» (знак минус); точную форму уголков можно увидеть в примере. Все остальные символы соответствуют изображению внутреннего цилиндра и равны либо «.» (точка) для белых пикселов, либо «X» (большая буква икс) для чёрных пикселов.

Гарантируется, что таймер корректно показывает некоторое время, строго меньшее 60 минут и кратное 5 секундам.

Формат выходных данных

Выведите время, которое показывает таймер, на отдельной строке в формате «MM:SS». Здесь «MM» — две цифры, соответствующие минутам, а «SS» — две цифры, соответствующие секундам.

Пример

timer.in

```
-------X..........XX..........XX..........XX.........-------
----..XX..........XX..........XX..........XX..........XX----
-............................................................-
X.....................................................XXXXXXX..
XX.....................................................XX.......
XX.....................................................XXXXXX...
XX.....................................................XX..
XX.....................................................XX..
XX.....................................................XX...XX..
-.....................................................XXXXX..-
----.....................................................----
-------.....................................................-------
```

timer.out

```
07:05
```

Пояснение к примеру

В этом примере центр окошка находится между отметками 10 и 5. Отметка, обозначенная числом 5, хорошо видна справа. Часть нуля из числа 10 можно увидеть слева. Более тщательное изучение фотографии позволяет установить, что время, которое показывает таймер, в точности равно 7 минутам и 5 секундам.

Задача K. Ультрапростые числа

Имя входного файла:	`ultraprime.in`
Имя выходного файла:	`ultraprime.out`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мебибайт

Как известно, *простым* называется целое положительное число, у которого ровно два целых положительных делителя — единица и само число. Назовём число *ультрапростым*, если в его десятичной записи можно стереть $x \geqslant 0$ цифр спереди и $y \geqslant 0$ цифр сзади, и независимо от x и y оставшееся число — конечно, если хоть что-то осталось — обязательно будет простым.

Например, число 37 — ультрапростое, так как простыми являются числа 3, 7 и 37, получающиеся из него стиранием цифр спереди и сзади. Число 5867, напротив, не является ультрапростым, поскольку, например, число 86 не является простым.

По заданному номеру n выведите n-е в порядке возрастания ультрапростое число.

Формат входных данных

В первой строке записано целое число n — номер требуемого ультрапростого числа ($1 \leqslant n \leqslant 1000$).

Формат выходных данных

В первой строке выведите n-е в порядке возрастания ультрапростое число или -1, если числа с таким порядковым номером нет.

Примеры

ultraprime.in	ultraprime.out
2	3
6	37

Пояснения к примерам

Последовательность ультрапростых чисел начинается так: 2, 3, 5, 7, 23, 37, ...