Задача A. Доска

Имя входного файла:	`board.in`
Имя выходного файла:	`board.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Не осилил — много букв.

царь Валтасар

Скив всё ещё с испугом посматривал на Ааза. Тем не менее, выхода не оставалось — после того, как неизвестный убийца уничтожил Гаркина, тот казался единственным союзником в этой ситуации. Да, Ааз был демоном, но это слово на самом деле обозначало просто «демонстратор измерений». Первоначально так называли любого, кто пришёл из другого измерения.

Неприятность была в том, что, благодаря шутке Гаркина, Ааз совершенно потерял магические способности, и был вынужден рассчитывать на пока ещё очень слабые магические умения Скива — или же на то, что среди того хлама, который Гаркин хранил у себя дома, найдётся магический артефакт.

После некоторых поисков Скив наткнулся на доску $m \times n$, заполненную маленькими буквами латинского алфавита. Ааз распознал в ней магический артефакт. Если расположить буквы определённым образом, то с помощью этого артефакта можно переместиться в другое измерение. Вопрос только, каким образом... На дне доски Ааз прочитал полустёртую инструкцию на распространённом в его родном измерении Извр языке: «... и буквы должны быть расставлены так, чтобы в i-м столбце было не менее k_i букв l_i.»

— Это то, что нам надо, — заявил Скив. Мы переберём все такие варианты... я думаю, что это не так долго.

— Ты так думаешь? Вместо того, чтобы подумать, ты мог бы и посчитать. Твоей жизни точно не хватит, чтобы перебрать все варианты... А вот хватит ли моей... — Ааз задумался.

Помогите ему подсчитать число способов, которыми можно расставить буквы так, чтобы выполнялось требование инструкции. Если число способов превосходит 10^{18}, выведите, что способов слишком много — столько времени не живут даже изверги.

Формат входного файла

Во входном файле заданы один или несколько наборов входных данных. В первой строке описания каждого набора заданы два числа $1 \le m, n \le 5$. Далее следуют n строк, каждая из которых описывает соответствующий столбец, и содержит значения k_i и l_i ($0 \le k_i \le m$, l_i — маленькая буква латинского алфавита). Далее следуют m строк по n символов в каждой — содержимое доски. Общая сумма всех значений m и n во входном файле не превосходит 500.

Входной файл завершается фиктивным набором с $m = n = 0$, который не нужно обрабатывать.

Формат выходного файла

Выведите в выходной файл количество допустимых расстановок букв. Если количество больше 10^{18}, выведите, что допустимых расстановок слишком много. Расстановки считаются одинаковыми, если в одинаковых позициях стоят одинаковые буквы.

Следуйте формату примера максимально точно.

Пример

board.in	board.out
2 2 1 a 2 b ab ba 2 2 1 a 1 b ab ba 5 4 0 a 0 b 0 c 0 d abcd efgh ijkl mnop qrst 0 0	Board 1: There are 1 ways. Board 2: There are 5 ways. Board 3: There are too many ways.

Задача B. Горящие окружности

Имя входного файла:	`circles.in`
Имя выходного файла:	`circles.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Два кольца, два конца, посередине хоббит.

Назгульская народная загадка

— Итак, для того, чтобы применять магию, надо обнаружить силовые линии. Их ты можешь увидеть через заклинание обнаружения магии, — сказал Ааз.

Скив сосредоточился. Действительно, в воздухе висело n светящихся колец.

— Как ты видишь, в этом месте силовые линии имеют форму окружностей. Более того, все окружности лежат в одной плоскости. Чтобы левитировать, необходимо сначала провести мысленно прямую так, чтобы она проходила через все эти окружности... именно через все, иначе не хватит энергии и ты упадёшь. А потом просто мысленно оттолкнуться вдоль этой прямой.

Демон подумал и добавил:

— Я не уверен, можно ли вообще провести такую прямую. Так что ты попробуй сначала просто вывести формулу этой прямой — если ты понимаешь, что это такое. А потом всё уже просто. Магия — не такая сложная штука.

Увы, Скив куда лучше владел магией, чем формулами. Так что и в этот раз без Вашей помощи не обойтись.

Формат входного файла

В первой строке входного файла задано количество наборов входных данных k ($1 \le k \le 20$). Далее следуют сами наборы, описание каждого состоит из строки, содержащей число окружностей n ($1 \le n \le 20$), за которой следуют n строк — описания окружностей. Каждая окружность описывается тремя числами: координатами центра x_i, y_i и радиусом r_i. Все эти числа вещественные и не превосходят 100 по абсолютной величине. Окружности могут пересекаться или совпадать.

Формат выходного файла

Для каждого набора выведите в отдельной строке описание прямой, на которой лежат все данные окружности, или информацию о невозможности проведения такой прямой. Если таких прямых несколько, то разрешается выводить любую. Для описания прямой необходимо вывести любую точку с минимальной суммой квадратов координат на ней и любой из нормированных направляющих векторов (т.е. с единичной суммой квадратов координат). Выводите ответ с максимально возможной точностью!

Пример

circles.in
2 2 0 0 1 5 4 2 4 0 0 1 4 4 1 4 0 1 0 4 1
circles.out
Test case 1: The circles are on (0.0, 0.0) + (0.7071067811865475244, 0.7071067811865475244) * t Test case 2: There are no suitable straight lines.

Задача C. Краны

Имя входного файла:	`faucets.in`
Имя выходного файла:	`faucets.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Теория Большого Взрыва — это подарок химиков физикам.

лекция по экспериментальной химии

В алхимической лаборатории, которую Скив и Ааз нашли в трактире Иштвана, над доской длиной X миллиметров были расположены n кранов. Все краны расположены на высоте Y миллиметров, в позициях X_1, X_2, ..., X_n миллиметров от начала доски ($0 \le X_1 < X_2 < \ldots < X_n \le X$), координаты возрастают слева направо. Краны были закрыты неплотно, и поэтому из них иногда капало. Капли из каждого крана падают с периодичностью P_i секунд ($0 < P_i < 100$) с ускорением свободного падения, равным в измерении Пент (в котором и происходит дело) $g = 10$ м/с^2 (начальная скорость каждой капли равна 0). Но капает не вода, а разные растворы. Скив, которому его наставник Ааз поручил присматривать за лабораторией, заметил, что при достижении доски капли ведут себя очень странным образом. А именно, они начинают ползти по доске влево или вправо (в зависимости от крана) с заданной скоростью $d_i > 0$ мм/с. Каждая ползущая капля оставляет за собой осадок высотой h_i миллиметров, начиная с точки ее падения. Как только встречаются две или более капель разных растворов, все встретившиеся капли взрываются (при этом в точке взрыва осадок не остается). Если встречаются ползущая и падающая капли, то это происходит на высоте не выше добавляющегося осадка ползущей капли (т.е. считается, что капля ползет сверху своего осадка, и если она встречается с не до конца упавшей каплей, находящейся на высоте ниже ползущей капли, то также происходит взрыв; если при этом падающая капля ровно в момент встречи касается поверхности, то осадок в точке взрыва также не остается). Сами по себе капли считаются точками. Падение капли происходит до высоты, равной сумме осадков проползших по этому месту до этого капель, не включая текущую каплю. При этом, свойства этих удивительных растворов таковы, что если капля переползает с участка одной высоты на участок другой высоты, то она мгновенно оказывается на нужной высоте.

Если какая-то капля доползает до края доски, то она срывается вниз и больше не появляется.

Всё это было очень интересно. И тут Скив услышал, что в дверь трактира кто-то постучал. Значит, и досмотреть, чем всё закончится, не удастся. «Интересно, что будет с доской через T секунд», — подумал Скив.

Формат входного файла

Во входном файле задано не более 5 наборов входных данных. В первой строке каждого набора заданы числа n, X, Y, T ($1 \le n \le 5$, $1 \le X, Y \le 100\,000$, $0 \le T \le 50$). Далее следуют n строк, описывающих краны. Каждый кран описывается шестью полями: X_i d_i h_i P_i R_i c_i, где $0 \le h_i \le 50$, $d_i \le 50$, c_i соответствует направлению движения капли, и равно '+', если капля ползет вправо (увеличивает x-координату) и '-' в противном случае. R_i ($0 \le R_i < P_i$) обозначает, что первая капля упадет из крана на R_i-й секунде. Все числа во входном файле целые.

Входной файл завершается фиктивным набором с $n = X = Y = T = 0$, который обрабатывать не нужно.

Формат выходного файла

Для каждого набора входных данных выведите состояние доски через T секунд (учитывая события, произошедшие ровно в момент времени T). Сначала в отдельной строке выведите информацию о номере набора. Далее выведите высоту участков доски в виде одного или нескольких интервалов, объединение которых составляет всю доску, в порядке их следования. Круглая скобка на границе

интервала обозначает, что соответствующая точка в него не включается, квадратная — что включается. Гарантируется, что в момент времени T высота всех участков доски строго меньше Y. В начальный момент времени высота всех участков равна 0.

Разделяйте выводы для различных наборов пустой строкой. Следуйте формату, показанному в примере, максимально точно. Вещественные числа выводите с максимально возможной точностью! Два интервала, описываемые подряд в выводе для одного набора, не могут иметь одинаковую высоту.

Пример

faucets.in
3 6 5000 5 1 1 5 2 1 + 4 2 3 4 0 - 6 3 7 5 4 + 0 0 0 0

faucets.out
Test case 1: on [0; 1): height 0 on [1; 1.333333333333333): height 10 on [1.333333333333333; 1.333333333333333]: height 5 on (1.333333333333333; 2.000500125062539]: height 8 on (2.000500125062539; 3.999399909972990): height 3 on [3.999399909972990; 4]: height 6 on (4; 6): height 0 on [6; 6]: height 7

Задача D. Отступы

Имя входного файла:	`indent.in`
Имя выходного файла:	`indent.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Этот трёхтомник изначально был задуман, как задача для ACM.

Дональд Кнут

Когда команда «Демоны» прибыла на Валлет для участия в Большой Игре, оказалось, что условия игры неожиданно изменились. Вместо команд должны были участвовать механические устройства, программы для которых должны были быть написаны на некоем формальном языке. Более того, для управления передвижениями устройства и его «боевой частью» использовались различные языки. Ни Скив, ни Корреш, ни даже горгул Гэс не ожидали такого поворота событий. Спокойным оставался лишь Ааз. Оказывается, в его родном измерении Извр программирование разных устройств было достаточно развито. Так что найти нескольких программистов и поставить им задачу для демона оказалось делом техники.

Однако проблемы возникли позднее, в ночь перед Большой Игрой. Ааз собрал программистов, работавших над «боевой частью», и изложил им суть проблемы:

— Как Вам известно, отступы — это пробельные символы в начале строк исходного файла, помогающие определить уровень вложенности кода в данной строке и облегчающие чтение кода. В некоторых языках, таких, как Haskell, Occam и Python, отступы не только служат подсказкой программисту, но и являются информацией о вложенности для компилятора; при этом отпадает необходимость использовать скобки или ключевые слова для обозначения начала и конца блоков. Например, нижеследующий фрагмент программы на Си++

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
    {
        c += a[i][j];
        if (c > m) assert (false);
    }
cout << c;
```

будет на Питоне выглядеть так:

```
for i in range (n):
    for j in range (n):
        c += a[i][j]
        if c > m: assert (False)
print c
```

Хотя для отступов можно использовать как пробелы, так и символы табуляции, не рекомендуется использовать и то, и другое в одной программе, поскольку в различных средах символ табуляции может быть эквивалентен разному количеству пробелов. С последствиями такого смешанного использования нам и придётся сейчас бороться.

В языке, используемом для программирования «боевой части», могут встречаться пробел и символ табуляции (далее называемые «пробельные символы»), а также буквы `a-z`, `A-Z`, цифры `0-9` и знаки `+-=^/!&|,;:()[]{}<>` (все они в дальнейшем называются «непробельные символы»). Будем считать, что каждая строка исходного файла содержит хоть один непробельный символ. Строка может заканчиваться символом ‘`:`’ (двоеточие), обозначающим начало нового блока вложенности, или любым другим непробельным символом. Назовём *отступом* строки число $k = a \cdot L + b$, где a —

количество символов табуляции до первого непробельного символа, b — количество пробелов до первого непробельного символа (порядок этих табуляций и пробелов не играет роли), а L — количество пробелов, которому эквивалентен один символ табуляции.

Программа обрабатывается парсером следующим образом. Сначала в стек кладётся число 0. Далее строки читаются подряд, начиная с первой. Если предыдущая прочитанная строка заканчивалась символом ‘:’, следующая строка должна иметь строго больший отступ; в этом случае в стек кладётся значение отступа новой строки. Если же это первая строка или предыдущая строка заканчивалась другим символом, новая строка должна иметь отступ, равный одному из значений в стеке; в этом случае со стека снимаются значения отступов до тех пор, пока верхнее число в стеке не станет равно отступу текущей строки. Программа считается корректной, если для каждой строки ввода эти требования были выполнены. Кроме того, корректная программа не должна заканчиваться строкой, последним символом которой является двоеточие.

Имея на входе программу, мы хотим узнать, для каких значений L в разумных пределах ($1 \le L \le 64$) эта программа будет являться корректной.

— Я ясно выразился? — спросил Ааз у программистов. — Если неясно, задавайте вопросы, я отвечу.

Формат входного файла

Входной файл состоит из нескольких тестов. Первая строка в описании теста содержит одно целое число N ($1 \le N \le 10\,000$) — количество строк программы. Следующие N строк — это сами строки программы. Для удобства восприятия и разбора данных все пробелы во входных файлах заменены на символы ‘.’ (точка), а все табуляции — на символы ‘*’ (звёздочка). Во входных файлах нет настоящих символов пробела и табуляции. Входной файл заканчивается тестом с $N = 0$; этот тест обрабатывать не надо.

Входной файл имеет размер не больше одного мегабайта; каждая строка входного файла имеет длину от 1 до 1 000 байт.

Формат выходного файла

Для каждого теста следует вывести три строки в выходной файл. Первая строка имеет вид “`Testcase #K:`”, где K — номер теста, считая с единицы. На второй строке должно стоять слово

· “`none`”, если ни одно значение L из указанного диапазона не позволяет получить корректную программу,

· “`unique`”, если подходит только одно значение L, или

· “`multiple`”, если подходит больше одного значения L.

В третьей строке нужно вывести через пробел те значения L, при которых программа оказывается корректной, в порядке возрастания. Если таких L нет, то эта строка пуста.

Отделяйте ответы на разные тесты пустой строкой.

Пример

indent.in	indent.out
1 **start: 5 for.i.in.range.(n): *for.j.in.range.(n): ....c.+=.a[i][j] **if.c.>.m:.assert.(False) print.c 4 if.i.==.1: ***if.j.==.2: ....n++ ***break 0	Testcase #1: none Testcase #2: unique 4 Testcase #3: multiple 1 2

Задача E. Инлайнинг

Имя входного файла:	`inline.in`
Имя выходного файла:	`inline.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

"TL" расшифровывается как "Too Lazy".

W.A. Debug-Submit

Уже наступило утро, а скоростные качества устройства были не ахти. При том, что реагировал на атаки потенциальных соперников (их роли исполняли поочерёдно все участники команды «Демоны») механический игрок просто замечательно, при передвижении машина сильно тормозила. На переделку алгоритма времени не оставалось. Значит, пришло время локальной оптимизации.

Программа, используемая для управления ходовой частью, представляет собой набор функций, который завершается строкой со словом "`end`" на ней. Оптимизация будет состоять в том, чтобы подставить код некоторых функций внутрь функций, вызывающих их, чтобы сэкономить время на вызов.

Каждая функция состоит не менее чем из двух строк и имеет следующий вид:

```
name: n
instruction_1
instruction_2
...
instruction_k
```

Здесь `name` — название функции, начинающееся с буквы, состоящее только из цифр и маленьких букв латинского алфавита и имеющее длину от 1 до 20 символов, n — время, затрачиваемое на однократный вызов функции ($1 \le n \le 128$), а каждая из строк $\texttt{instruction}_i$ — либо слово "`command`", либо название другой функции. Структура программы такова, что, во-первых, никакая функция не может вызывать — прямо или с помощью цепочки других функций — саму себя, а во-вторых, одна и та же функция не может вызываться из двух различных функций — ни прямо, ни с помощью цепочек других промежуточных функций.

Будем называть *inline-подставляемой* функцию, код которой будет подставлен во все места программы, где происходит её вызов. Каждая функция либо будет inline-подставляемой, либо будет вызываться «по-честному» всякий раз, когда встретится такая инструкция.

Листинг функции `function` — это просто её определение, в котором каждая инструкция вызова inline-подставляемой функции заменена на листинг этой функции.

Размер $S_{\texttt{function}}$ функции `function` определяется как количество инструкций в её листинге; в частности, размер функции, не содержащей вызовов inline-подставляемых функций, равен числу k инструкций в её определении.

Время на вызов $C_{\texttt{function}}$ функции `function` определяется как число n, записанное после двоеточия в первой строке её определения.

Время работы $T_{\texttt{function}}$ функции `function` — это её размер $S_{\texttt{function}}$, к которому для каждой инструкции в листинге, которая является вызовом не inline-подставляемой функции `child`, прибавлена величина $C_{\texttt{child}} + T_{\texttt{child}}$.

Названия всех функций внутри одной программы различны. Запуск программы — это однократный запуск функции `main` в ней; функция с таким именем присутствует в каждой программе. Гарантируется, что функции не будут называться "`command`" и "`end`".

Требуется таким образом выбрать, какие функции будут inline-подставляемыми, а какие — нет, чтобы минимизировать время работы программы при условии, что её итоговый размер должен оказаться не больше размера кэша первого уровня. Время выполнения программы — это сумма времени выполнения функции `main` и времени её вызова; её размер — суммарный размер всех не inline-подставляемых функций.

Формат входного файла

Входной файл состоит из одного или нескольких тестов. Первая строка в описании теста содержит одно целое число R ($1 \le R \le 64$) — размер кэша первого уровня. Далее следуют описания функций, которые завершаются строкой со словом "`end`". Входной файл заканчивается строкой, в которой $R = 0$; эту строку обрабатывать не надо. Каждая программа содержит не более 256 функций, а каждая функция — не более 256 инструкций. Всего во входном файле объявлено не более, чем 1 024 функции, а его размер не превышает одного мегабайта. Все числа во входном файле целые.

Формат выходного файла

Для каждого теста следует вывести в выходной файл строку "`Testcase #K: T`", где K — номер теста, считая с единицы, а T — оптимальное время выполнения программы при условии, что она помещается в кэш первого уровня, и это время невелико. Если поместить программу в кэш не удастся, или же если минимально возможное T больше миллиона, вместо T следует вывести слово "`impossible`". В противном случае в следующих строках для каждой функции программы требуется вывести, будет ли она inline-подставляемой. Порядок функций такой же, как порядок их объявления во входном файле; функцию `main` inline-подставляемой сделать нельзя. Как можно точнее следуйте формату вывода, приведённому в примерах. При существовании нескольких оптимальных ответов можно вывести любой из них. Отделяйте ответы на разные тесты пустой строкой.

Пример

inline.in	inline.out
64 main: 2 function1 function1: 4 command function2 command function2: 1 command command command end 11 main: 2 function1 function1 function2 function2 function1: 4 command command command function2: 1 command command command end 1 main: 1 command command end 0	Testcase #1: 7 main: no function1: yes function2: yes Testcase #2: 18 main: no function1: yes function2: no Testcase #3: impossible

Задача F. Лошадью ходи!

Имя входного файла:	`knights.in`
Имя выходного файла:	`knights.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Который год нам нет житья от этих "Фрицев".

из интервью Крамника

Сказать, что деволы любят азартные игры — это значит ничего не сказать. Деволы азартные игры просто обожают, особенно — такие, на которых можно неплохо заработать. Вы спросите, а кто такие деволы. Так вот, деволы — это жители измерения Девы, лучшие торговцы во всех измерениях. А в их родном измерении они организовали грандиозный Базар-На-Деве, где можно достать всё, что угодно... и сыграть во всё, что угодно.

Скив уже не в первый раз был на Базаре, и тем не менее впечатление, произведённое на него базаром, было огромным. К сожалению для Скива, Ааз и Тананда, с которыми он прибыл сюда, сейчас искали какие-то артефакты, необходимые для реализации очередного замысла Ааза. И когда Скив оказался среди зрителей, глазеющих на странную игру на клетчатой доске, предупредить его о здешних порядках было некому.

То, во что здесь играли, чем-то напоминали Скиву известные на Пенте (как называлось его измерение) шахматы. По крайней мере, одну из фигур он узнал — это был конь. Шахматная доска же была несколько странных пропорций. Размер её был $m \times n$.

— Сыграем? — неожиданно предложил Скиву владелец палатки.

Скив понял, что отказываться будет уже не с руки — он оказался в центре толпы. Правила игры были просты. Ведущий называл число k, а Скив должен был или спасовать, или принять ставку. В случае, если ставка принималась, Скив должен был за две секунды расставить на доске максимально возможное количество коней таким образом, чтобы каждый из коней бил ровно k других коней.

Было названо первое k. Скив задумался и вдруг услышал явное слово «пас». Он поднял голову и увидел пробравшегося сквозь толпу Ааза. Скив спасовал, к неудовольствию ведущего. Зато на следующее k он удвоил ставку...

— Вечно ты находишь проблемы! Тебе повезло, что я немало времени провёл за этой игрой, — отчитывал Ааз Скива, когда они удалялись от палатки с крупным выигрышем.

Судя по всему, Ааз знал, как здесь надо играть. А вот знаете ли это Вы?

Формат входного файла

Во входном файле задан один или несколько тестовых наборов. Каждый набор описывается одной строкой, содержащей три числа: m, n, k ($1 \le m, n \le 8$, $0 \le k \le 8$). Сумма всех чисел, встречающихся во входном файле, не превосходит 60. Входной файл завершается фиктивным набором из трех нулей.

Формат выходного файла

Для каждого из наборов выведите расстановку коней, как показано в примере, или информацию о том, что такой расстановки не существует. Порядок следования коней неважен. Координаты каждого из коней выводите маленькой латинской буквой, соответствующей столбцу, и числом, соответствующим строке (всего на доске n столбцов, пронумерованных, начиная с буквы `a`, и m строк, пронумерованных числами от 1 до m). Следуйте формату примера максимально точно.

Пример

knights.in	knights.out
3 3 1 2 2 4 0 0 0	Test set 1: a1, c2, c3, a2 Test set 2: IMPOSSIBLE

Задача G. Погоня на Лимбе

Имя входного файла:	`mirror.in`
Имя выходного файла:	`mirror.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Свет мой, зеркальце, ска...

Медуза Горгона

— А вот это уже интересно, — заметил Ааз, смотря куда-то вниз. Скив посмотрел туда же, но ничего особенного не увидел. Какие-то осколки... всего n выпуклых фигур. Скив присмотрелся и заметил, что граница осколков составлена из отрезков и дуг окружностей, более того, граница каждого осколка имеет хотя бы одну дугу. Всё равно непонятно, что в этом нашёл Ааз...

— Что там, босс? — спросил Скива Гвидо.

— Просто осколки зеркала, — ответил Скив.

— И откуда же им взяться в Лимбо? — Ааз перешёл к своей обычной манере.

Скив понял, про что говорил его учитель. Лимбо — это измерение, заселённое вампирами. Вампиры, скажем прямо, не большие охотники пользоваться зеркалами. А значит, зеркало могло принадлежать тем, кого они искали.

Гвидо вспомнил, что вроде бы действительно, что-то типа круглого зеркала он у мошенников видел.

— Не факт, что разбившееся зеркало круглое, — скептически заметил Скив.

Требуется определить, можно ли сложить осколки в один круг.

Формат входного файла

Во входном файле задано не более десяти наборов входных данных. Количество наборов T задано в первой строке. Описание каждого набора начинается с числа выпуклых фигур n ($1 \le n \le 8$). Описание каждой фигуры начинается со строки, содержащей количество ограничивающих её примитивов p_i ($1 \le p_i \le 8$). После этого идет p_i строк, описывающих примитивы. Граница каждого кусочка рисуется, начиная с точки $(0, 0)$, и заканчивая в этой же точке, в порядке обхода против часовой стрелки. Все отрезки и дуги задаются тремя числами — x_j y_j k_j, где x_j и y_j — точки окончания соответствующего примитива (отрезка или дуги), заданной относительно текущей точки, а k_j — его кривизна, которая для отрезка равна нулю, а для окружности является величиной, обратной к радиусу. Хотя бы одно из чисел x_j и y_j для каждого j отлично от 0. Все числа — вещественные, заданные с максимально возможной точностью, и не превосходят 100 по модулю. Зеркало является односторонним, и кусочки изначально повёрнуты зеркальной стороной вверх.

Формат выходного файла

Для каждого из наборов выведите, может ли он описывать разбитое круглое зеркало, как показано в примере. Следуйте формату выходного файла максимально точно.

Пример

mirror.in
2 2 2 1 0 0 -1 0 2 2 0.7071067811865475244 0.7071067811865475244 0 -0.7071067811865475244 -0.7071067811865475244 2 2 2 1 0 0 -1 0 2 2 0.7071067811865475244 0 0 -0.7071067811865475244 0 2
mirror.out
Test case 1 describes a broken round mirror. Test case 2 does not describe a broken round mirror.

Задача H. Кривые числа

Имя входного файла:	`numbers.in`
Имя выходного файла:	`numbers.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Счастье — это когда тебя поднимают.

афоризм игроков в драконий покер

— К вам клиенты, Босс, — доложил Нунцио, один из телохранителей, присланных доном Брюсом после того, как Скив согласился представлять интересы Синдиката на Базаре-на-Деве.

— Пусть войдут.

Вошедший девол оказался торговцем счастливыми номерами. Он в совершенстве знал все приметы, связанные с числами в разных измерениях, и торговал различными объектами, содержащими "счастливые" числа — от клочков бумаги с 6 цифрами до денежных купюр с соответствующими номерами.

— Мне грозит полное разорение, — начал он.

Оказалось, что некие недобросовестные конкуренты пустили по Базару слух о том, что продаваемые им номера, возможно, и счастливые — но какие-то кривые.

— То есть кривые? — спросил подошедший Ааз, деловой партнёр Скива, а в прошлом — его наставник.

— Они говорят, что число называется *k-кривым*, если любые k его последовательных цифр являются в совокупности взаимно простыми числами (т.е. имеют наибольший общий делитель, равный единице).

— А насколько дорого достаётся Вам ваш товар? — уточнил Ааз.

— Ну, не очень... дело в том, что в разных измерениях счастливыми считаются различные числа. Так что если бы вы помогли с помощью своей магии проверять числа на кривость, это бы решило мою проблему.

Требуется написать программу, проверяющую заданные числа на кривость.

Формат входного файла

В каждой строке входного файла содержится целое число $M_i > 0$ длиной не более 10^6 цифр, и k_i, обозначающее, что нужно проверить M_i на k_i-кривость ($2 \le k_i \le L_i$, где L_i обозначает длину числа M_i). M_i не может начинаться с цифры 0. Входной файл завершается строкой из двух нулей, которую не нужно обрабатывать. Размер входного файла не превосходит одного мегабайта, и общее количество строк в файле не превосходит 1 000.

Формат выходного файла

Для каждого числа M_i выведите на отдельной строке, является ли оно k_i-кривым, как показано в примере. Следуйте формату выходного файла максимально точно. Помните, что $\gcd(0, x) = \gcd(x, 0) = x$.

Пример

numbers.in	numbers.out
237 2 239 2 236 2 222333222 4 0 0	Number 1 is crooked. Number 2 is not crooked. Number 3 is not crooked. Number 4 is crooked.

Задача I. Многомерность

Имя входного файла:	`packing.in`
Имя выходного файла:	`packing.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Камерная музыка? Это слишком плоско!

жители $k+1$-мерного мира

— А всё-таки, как путешествовать по измерениям? — в очередной раз спросил Скив у Ааза.

— Ааз, тебе стоит ему это объяснить. Если кто-нибудь на Базаре узнает, что Великий Скив не может путешествовать по измерениям, это скажется на нашей репутации и на наших доходах, — поддержала Скива Тананда.

— Что же, я могу попробовать что-то объяснить. Но начну с теории. То, что мы называем «измерениями» — Дева, Пент, Извр — на самом деле есть трёхмерные подпространства в d-мерных пространствах. И в зависимости от того, какое из пространств мы выберем, мы можем там перемещаться. Начинающему магу приходится ограничиваться перемещением по «уголкам».

— Это как?

— Назовём уголком d-мерный гиперкуб $2\times2\times\ldots\times2$ с вырезанным из угла гиперкубом $1\times1\times\ldots\times1$.

Скив попытался представить... получилось с трудом. А Ааз продолжал:

— Для того, чтобы ты понял, как легко заблудиться в измерениях, вот тебе задача: в d-мерный гиперкуб со стороной 2^n нужно засунуть как можно больше непересекающихся уголков так, чтобы их стороны были параллельны осям координат. Решишь задачу — тогда займёмся практикой.

Формат входного файла

В первой строке входного файла заданы два целых числа d и n. Известно, что $d, n \geq 2$, и $n \cdot d < 24$; кроме того, d и n таковы, что в правильном ответе будет не более, чем 100 000 строк.

Формат выходного файла

В первой строке необходимо вывести m — количество засунутых уголков. В следующих m строках требуется вывести описание уголка в каждой. Каждое описание состоит из двух групп по d чисел. Первые d из них — минимальные координаты вершины гиперкуба со стороной 2, содержащего данный уголок. Вторые d — минимальные координаты вершины вырезанного из него гиперкуба со стороной 1. Все числа в выходном файле должны быть целыми. Уголки поворачивать не разрешается.

Если решений с максимальным m несколько, разрешается выводить любое из них.

Пример

packing.in	packing.out
2 2	5 2 2 3 3 0 2 1 2 2 0 2 1 0 0 1 1 1 1 2 2

Пример разобран на рисунке.

Задача J. Суффиксный пулемёт

Имя входного файла:	`suffix.in`
Имя выходного файла:	`suffix.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Или зачёт, или автомат.

Ганнибал Ректор

Теоретическая подготовка новобранцев армии Поссилтума включала в себя не только занятия по военному праву, но и начала криптографии. Лекции читал майор Мега Байт, не чуждый солдатского юмора. Гвидо и Нунцио, в чьё задание входил развал армии Поссилтума изнутри, решили на этом сыграть, внеся путаницу в терминологию. В начале очередной лекции Нунцио поднял руку и спросил:

— Вот вы на прошлой лекции рассказывали про конечные автоматы. А про конечные пулемёты расскажете?

Мега Байт не растерялся.

— Суффиксный пулемёт — это конечный автомат, принимающий все суффиксы данной строки (от нулевого до L-го включительно, где L — длина строки), и только их. Сержант Гвидо!

— Я, господин майор!

— Вы сможете отличить автомат от пулемёта?

— Так точно, господин майор!

— Вам дан конечный автомат. Требуется проверить, является ли он суффиксным пулемётом данной строки.

К сожалению, написание программ такого типа не входило в обязанности Гвидо и Нунцио как в Синдикате, так и в корпорации М. И. Ф. Так что соответствующую программу придётся писать Вам.

Формат входного файла

Во входном файле задан один или несколько тестовых наборов. В первой строке каждого набора заданы количество состояний автомата N, количество переходов M, а также количество принимающих состояний T ($1 \le T \le N \le 50\,000$, $1 \le M \le 100\,000$). Во второй строке через пробел заданы T различных чисел в пределах от 1 до N — принимающие состояния автомата, в возрастающем порядке. В последующих M строках заданы переходы в виде a_i b_i c_i, где $1 \le a_i, b_i \le N$, а c_i — маленькая буква латинского алфавита. Переход производится из состояния a_i в состояние b_i по букве c_i. Из каждого состояния a_i есть не более одного перехода по символу c_i. Последняя строка описания набора — это строка S, для которой автомат должен являться пулемётом. Она состоит только из маленьких латинских букв, и ее длина лежит в пределах от 1 до 50 000 включительно. Кроме того, сумма всех N и суммарная длина всех строк, для которых необходимо произвести проверку, не превосходит 50 000, а сумма всех M не превосходит 100 000.

Файл заканчивается фиктивным набором, в котором $N = M = T = 0$.

Начальным состоянием автомата является первое. Если при интерпретации какой-то строки в автомате отсутствует соответствующий переход, то автомат вываливается по ошибке и строку не принимает. Таким образом, строка принимается, только если при её интерпретации были найдены все переходы, и по их завершении автомат оказался в принимающем состоянии (при этом неважно, были по пути принимающие состояния, или нет).

Формат выходного файла

Выведите в выходной файл, является ли данный автомат пулемётом, следуя формату примера.

Пример

suffix.in	suffix.out
2 1 2 1 2 1 2 a a 2 2 2 1 2 1 1 a 1 2 b ab 0 0 0	Automaton 1 is a machinegun. Automaton 2 is not a machinegun.