Задача A. Система счисления с основанием $i-1$

Имя входного файла:	`base-i-1.in`
Имя выходного файла:	`base-i-1.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Гауссовым целым называется комплексное число вида $a+b\cdot i$, где a и b — это целые числа, а i — квадратный корень из -1.

Будем говорить, что гауссово целое записано в системе счисления с основанием $i-1$ в виде $\overline{c_n c_{n-1}\ldots c_1 c_0}_{i-1}$, если $a+b\cdot i=\sum\limits_{k=0}^{n} c_k\cdot(i-1)^k$, и при этом все $c_k\in\{0,1\}$. Кроме того, если $n\neq 0$, то $c_n\neq 0$. Удивительно, но для любого гауссова целого такое представление существует и единственно.

Например $1=1_{i-1}$, $-1=11101_{i-1}$, $2i+1=1111_{i-1}$.

Вам дано два гауссовых целых, записанных в системе счисления с основанием $i-1$. Посчитайте их сумму и выведите её в системе счисления с основанием $i-1$.

Формат входных данных

В первой строке задано число a, а во второй — число b. Каждое из чисел записано в системе счисления с основанием $i-1$ и содержит от одной до 500 000 цифр.

Формат выходных данных

Выведите сумму чисел a и b, записанную в системе счисления с основанием $i-1$.

Примеры

base-i-1.in	base-i-1.out	Пояснение
101 10	111	a = 1-2*i b = -1+1*i a+b = 0-1*i
10100100100 110101	11111001100001	a = 20-38*i b = 1-6*i a+b = 21-44*i

Задача B. Перемешиваем квадраты

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

Вам дано целое число n. Рассмотрим его представление в двоичной системе счисления без ведущих нулей (например, $10_{10} = 1010_2$). Сколько существует способов переупорядочить двоичные цифры этого числа так, чтобы полученное в итоге число было точным квадратом? Например, для $10_{10} = 1010_2$ существует ровно один способ: $1010_2 \to 1001_2 = 9 = 3^2$. Переупорядочивать так, чтобы в числе появились ведущие нули или чтобы число в итоге превосходило 10^{18}, не разрешается.

Формат входных данных

В единственной строке записано целое число n ($1 \leqslant n \leqslant 10^{18}$).

Формат выходных данных

Выведите единственное число: количество способов переупорядочить биты n так, чтобы в результате получился точный квадрат.

Примеры

bit-squares.in	bit-squares.out
9	1
10	1
2	0
1000000000000000000	12206114

Задача C. Цыплята

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

Откуда берутся цыплята? Вероятно, вы тоже когда-либо задавались этим вопросом. Возможно, вы даже прочитали ответ на этот вопрос в какой-нибудь книге. Но, как говорится, лучше один раз увидеть, чем сто раз услышать.

В этой задаче мы рассмотрим разновидность матрёшки — курочку-наседку. Эта игрушка состоит из нескольких частей:

- Самая внешняя оболочка матрёшки — сама курочка.
- n яиц, пронумерованных целыми числами от 1 до n. Каждое яйцо разбирается, так что в него можно что-нибудь положить.
- n цыплят, пронумерованных целыми числами от 1 до n. Цыплята не разбираются.

В собранном виде каждый цыплёнок лежит в отдельном яйце, а все яйца лежат в курочке. Не разрешается класть яйца внутрь других яиц, класть нескольких цыплят в одно яйцо или же никуда не класть каких-то цыплят.

Каждый цыплёнок характеризуется своим размером c_i ($1 \leqslant i \leqslant n$). Аналогично, каждое яйцо характеризуется своим размером e_j ($1 \leqslant j \leqslant n$). Цыплёнка с номером i можно положить в яйцо с номером j тогда и только тогда, когда $c_i \leqslant e_j$.

Ваша задача — подсчитать количество способов собирать игрушку, то есть количество способов расположить цыплят по отдельным яйцам. Два способа считаются разными, если в них какой-нибудь цыплёнок лежит в яйцах с разным номером.

Формат входных данных

В первой строке задано целое число n — количество цыплят и яиц ($1 \leqslant n \leqslant 12$). В второй строке через пробел записаны n целых чисел $c_1, c_2, \ldots, c_n$ — размеры цыплят ($1 \leqslant c_i \leqslant 100$). В третьей строке через пробел записаны n целых чисел $e_1, e_2, \ldots, e_n$ — размеры яиц ($1 \leqslant e_i \leqslant 100$).

Формат выходных данных

В единственной строке выведите единственное целое число — количество способов расположить цыплят по отдельным яйцам.

Примеры

chickens.in	chickens.out
3 1 2 3 3 3 1	2
4 1 1 1 1 100 100 100 100	24
4 100 100 100 100 1 1 1 1	0
10 1 2 3 4 5 6 7 8 9 10 2 3 4 5 6 7 8 9 10 11	512

Задача D. Светофоры на перекрёстке

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

Робот Петя сидит в новом офисе своей компании и дистанционно управляет несколькими робо-такси. Время от времени, когда все такси останавливаются, он бросает взгляд на перекрёсток под окном у офиса. Пете видны n светофоров на перекрёстке. Эти светофоры довольно необычные. На каждом из них есть только два возможных сигнала: красный свет и зелёный свет. Когда горит красный свет, светофор показывает целое положительное число: сколько секунд осталось до зелёного света. Когда горит зелёный, никакие числа не показываются. Показания всех светофоров меняются одновременно, один раз в секунду.

Робот Петя знает, что все светофоры на перекрёстке связаны в единую систему. У этой системы есть период в T секунд: если посмотреть на светофоры два раза с промежутком ровно в T секунд, все показания совпадут. За этот период цвет каждого светофора меняется ровно дважды: на светофоре i в течение X_i секунд непрерывно горит красный, а затем в течение $Y_i = T - X_i$ секунд непрерывно горит зелёный. Кроме того, в каждый момент времени хотя бы на одном светофоре горит красный свет. При этом Петя не знает ни периода T, ни конкретных чисел X_i и Y_i для каждого светофора, он знает лишь общие принципы устройства системы, описанные выше.

Робот Петя хочет узнать период T. У роботов хорошая память, и Петя помнит результаты m наблюдений: каждый раз, когда Петя выглядывал в окно, он запоминал показания всех n видимых светофоров. Помогите ему узнать по известным наблюдениям, чему в точности равно число T, или выяснить, что по известным данным это сделать невозможно.

Формат входных данных

В первой строке заданы числа n и m — количество светофоров на перекрёстке и количество наблюдений ($2 \leqslant n \leqslant 20$, $2 \leqslant m \leqslant 250$). Следующие m строк описывают наблюдения. Каждая из них содержит n одновременных показаний светофоров: первого, второго, ..., последнего. Если на светофоре горит зелёный свет, это обозначено как «`X`» (большая английская буква «икс»). Если же горит красный, показания выражаются целым строго положительным числом — количеством секунд, оставшихся до зелёного сигнала. Соседние показания в строке разделены хотя бы одним пробелом, а кроме того, перед каждым показанием могут быть дополнительные пробелы для лучшей читаемости.

Гарантируется, что светофоры на перекрёстке устроены так, как описано в условии. Про числа T, X_i и Y_i Петя точно знает, что все они целые и строго положительные. Кроме того, Пете известно, что X_i и Y_i ограничены снизу числами около 10 секунд, а период T ограничен сверху числом около 200 секунд, но точных границ он не знает: возможно, границы на самом деле на несколько секунд больше или меньше.

Формат выходных данных

Если по сделанным наблюдениям можно однозначно установить точный период системы T, выведите это число. В противном случае выведите число -1.

Примеры

crossing-lights.in	crossing-lights.out
4 5 X 33 X 36 X 4 X 7 42 2 42 5 X 21 X 24 8 X 8 54	83
2 2 X 100 100 X	-1

Пояснения к примерам

В первом примере роботу Пете видны четыре светофора. Реальные параметры системы следующие. Общий период системы равен 83 секундам. Первый и третий светофоры работают одинаково, на них 40 секунд горит зелёный и 43 секунды красный свет. На втором светофоре зелёный загорается через 43 секунды после того, как зелёный загорается на первом, и горит 36 секунд. На четвёртом светофоре зелёный загорается через 46 секунд после первого и горит 29 секунд.

Наблюдения сделаны через 10, 39, 41, 22 и 75 секунд после момента, когда на первом светофоре загорается зелёный. Оказывается, что из этих наблюдений можно однозначно восстановить период.

Во втором примере всего два светофора и два наблюдения. По наблюдениям можно утверждать, что период не меньше 200 секунд. Но он может оказаться и любым числом больше 200, а робот Петя не знает точную верхнюю границу для периода. Поэтому однозначно восстановить число T невозможно.

Задача E. Десятичная дробь

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

Жюри загадало целые взаимно простые числа a и b ($0 \leqslant a < b \leqslant 10^9$). Вам дано число $\frac{a}{b}$ в форме десятичной дроби, округлённой ровно до 18 цифр после десятичной точки. Округление в этой задаче всегда происходит *к ближайшему числу*, при равенстве *к большему*. Найдите a и b.

Формат входных данных

В первой строке задано число n — количество тестовых случаев ($1 \leqslant n \leqslant 10^4$). Следующие n строк описывают тестовые случаи. Каждая из них содержит десятичную дробь, в которой ровно 18 цифр после десятичной точки. Гарантируется, что каждая дробь получена способом, описанным в условии.

Формат выходных данных

Выведите n строк. Строка с номером i должна содержать числа a и b ($0 \leqslant a < b \leqslant 10^9$), которые загадало жюри в i-м тестовом случае.

Пример

decimal-form.in	decimal-form.out
2 0.000000000000000000 0.666666666666666667	0 1 2 3

Пояснение к примеру

В первом тестовом случае $\frac{0}{1} = 0$.

Во втором тестовом случае $\frac{2}{3} \approx 0.666\,666\,666\,666\,666\,667$.

Задача F. Лабиринт на Марсе

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

Это интерактивная задача.

Исследовательский робот на Марсе собирается пройти очередной лабиринт. Лабиринт — плоское прямоугольное поле, ориентированное по сторонам света и расчерченное на $h \times w$ квадратных клеток. Между каждыми двумя соседними по стороне клетками либо стена, либо проход. Вокруг поля — сплошная стена.

Робот находится в клетке в юго-западном углу лабиринта. Задача робота — попасть в клетку в северо-восточном углу лабиринта. Раз в секунду роботу можно послать команду, чтобы он переместился в соседнюю клетку в одну из четырёх сторон света: «`N`» (на север), «`W`» (на запад), «`S`» (на юг) или «`E`» (на восток). После этого робот либо выполнит команду, если в соответствующей стороне проход, либо останется на месте, если там стена. Затем робот пришлёт ответ: «`yes`», если перемещение удалось, или «`no`» в противном случае.

Связь с роботом осуществляется через сеть спутников, и в зависимости от положения Фобоса и Деймоса задержка между передачей команды и получением ответа может быть весьма большой. В ближайшее время задержка постоянна и составляет d секунд. Это означает, что ответ на команду с номером x можно получить сразу после отправления команды с номером $x + d$. Но довольно скоро начнётся песчаная буря: через $t = 10\,000$ секунд посылать команды будет невозможно, можно будет лишь принять все оставшиеся ответы на предшествующие команды.

Возьмите на себя управление роботом и успейте привести его к цели до того, как буря сделает это невозможным.

Протокол взаимодействия

Сначала вашей программе подаются в отдельной строке три целых числа h, w и d — количество клеток с севера на юг, количество клеток с запада на восток и задержка в секундах ($2 \leqslant h, w \leqslant 20$, $0 \leqslant d \leqslant 100$). Лабиринт в каждом тесте зафиксирован заранее, но держится в секрете.

Каждую секунду в мире робота происходит следующее.

1. Если выведено меньше t команд, ваша программа должна вывести очередную команду «`N`», «`W`», «`S`» или «`E`» в отдельной строке. Пропустить команду или скомандовать стоять на месте невозможно.

   Чтобы предотвратить буферизацию вывода, после каждой выведенной команды 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.

2. Если выведено больше d команд, ваша программа должна получить ответ на команду, выведенную d секунд назад. Обычно это будет ответ «`yes`» или «`no`», означающий, получилось ли сделать соответствующее перемещение. Кроме того, если после выполнения этой команды робот достиг цели, вместо «`yes`» ответом будет «`success`», в этом случае все дальнейшие команды будут проигнорированы, и следует сразу же корректно завершить выполнение программы. Наконец, если после последней возможной команды робот так и не достиг цели, вместо ответа будет слово «`timeout`», а решение получит вердикт «`Wrong Answer`».

Гарантируется, что в каждом тесте лабиринт построен следующим образом. Сначала установим сплошную стену вокруг поля. Далее рассмотрим все возможные стены между парами соседних клеток лабиринта: $h \cdot (w - 1)$ стен, идущих с запада на восток, и $(h - 1) \cdot w$ стен, идущих с севера на юг. Зафиксируем случайный порядок на этих возможных стенах и будем рассматривать их в этом порядке. Каждую возможную стену мы будем добавлять в лабиринт, если после установки этой стены из каждой клетки лабиринта всё ещё можно добраться до всех других клеток. Можно

доказать, что в результате получится лабиринт, в котором между каждой парой клеток существует ровно один путь, не проходящий ни по какой клетке дважды.

Примеры

стандартный ввод	стандартный вывод	Лабиринт
2 3 0 no yes no yes success	 W E E N E	+-+-+-+ | | | + + + + | | | +-+-+-+
4 3 2 yes yes yes yes success	 N N E N E W W	+-+-+-+ | | | + + +-+ | | | + + + + | | | | + +-+ + | | +-+-+-+

Задача G. Мокрый крот

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

Молодой крот Фридрих недавно поселился в комфортабельной норе Цветочного холма. Но не успел он распаковать вещи и повесить на стену норы портрет кротёти, как небо затянулось свинцовыми тучами — скоро будет дождь! Будет очень неловко, если портрет тёти намокнет, так что надо срочно найти в норе место, куда вода точно не попадёт.

Как известно, кротовые норы Цветочного холма вырыты по типовому проекту: каждая нора представляет из себя прямоугольник h кретров (кротовая мера длины) в глубину и w кретров в ширину. Третье измерение у кронор эконом-класса отсутствует, чтобы не пересекаться с соседями. В каждом квадратном кретре кроноры может быть либо пусто (часть норы), либо земля.

Во время дождя вода капает сверху и заполняет все пустые клетки верхнего уровня кроноры, откуда стекает вниз и в стороны. Формально: если клетка заполнена водой, а на один кретр ниже находится пустая клетка, то эта пустая клетка тоже заполняется водой. Если же под клеткой с водой земля, а один кретр левее или правее есть пустые клетки (одна или обе), то вода заполнит и эти пустые клетки.

Вам дана схема кроноры Фридриха: прямоугольник из $h \times w$ клеток, про каждую из которых известно, земля там или пусто. Требуется найти клетку, в которую вода не попадёт, или сказать, что такой клетки нет. Конечно же, вешать портрет можно только в пустую клетку, а не закапывать в клетке с землёй.

Формат входных данных

Первая строка содержит два положительных целых числа: h и w. В каждой из следующих h строк содержится по w символов: план кроноры начиная с верхнего уровня. Каждый символ — это либо «`.`» (пусто), либо «`#`» (земля). Известно, что кронора не превосходит 500 кретров по каждому измерению. Гарантируется, что в нижнем уровне кроноры нет пустых клеток.

Формат выходных данных

В первой строке выведите слово «`Yes`» в случае, если клетка без воды найдётся, или «`No`» в противном случае. Если клетка нашлась, дополнительно выведите h строк по w символов: план кроноры в том же формате, что и во входных данных, но отметьте символом «`X`» клетку, подходящую для портрета. Если возможных вариантов несколько, выведите любой из них.

Примеры

mole.in	mole.out
4 6 #.##.# #.#.## #...## ######	Yes #.##.# #.#X## #...## ######
5 6 #.#.## #.#.## #....# ###.## ######	Yes #.#.## #.#.## #...X# ###.## ######
5 5 ..##. .##.. .#..# ...## #####	No
20 30 .....#..############....##.... .###...###########...#.##...## ..###..###########.###....#### #..#.....#########...###....## ##.#.###..#...######..##.##..# ##.#....#####..######.##.###.. ##..###...####........##.####. ###.#####....#####.#####...##. ###......###........######.... ########.########.#........### ########..#######..#..######## #########.########.##....##### #######...########...###..#### #######.############.####..... #######...##########..#######. #########..##########........# ##########.##################.. ##########...#################. ############.################. ################################	Yes .....#..############....##.... .###...###########...#.##...## ..###..###########.###....#### #..#.....#########...###....## ##.#.###..#.X.######..##.##..# ##.#....#####..######.##.###.. ##..###...####........##.####. ###.#####....#####.#####...##. ###......###........######.... ########.########.#........### ########..#######..#..######## #########.########.##....##### #######...########...###..#### #######.############.####..... #######...##########..#######. #########..##########........# ##########.##################.. ##########...#################. ############.################. ################################

Задача H. Странности

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

Джек организует вечеринку. Он владеет лучшим ночным клубом в городе. В клубе есть n комнат и m двусторонних коридоров между ними (каждый коридор соединяет две различных комнаты). Ночной клуб является связным, то есть по коридорам можно добраться от любой комнаты до любой другой. Джек хочет, чтобы его вечеринка была очень крутой и странной одновременно, поэтому он решил уничтожить часть коридоров между комнатами. Он хочет сделать это таким способом, чтобы для каждого i чётность количества коридоров комнаты i была равна фиксированному числу $a_i \in \{0, 1\}$. Поэтому Джек попросил своего знакомого Джона найти способ уничтожить коридоры.

Джон работал две недели, но не пришёл к успеху. Он обнаружил, что $\left(\sum_{i=1}^{n} a_i\right) \bmod 2 = 1$, и доказал, что в таком случае условие Джека соблюсти нельзя. Услышав это, Джек очень расстроился. Он сказал, что доказательство Джона никуда не годится, и ему нужно *железное доказательство* в течение пяти часов. Джон был достаточно умён и не спросил Джека, что случится через пять часов, а сразу побежал к вам за помощью.

В первую очередь, нужно понять, что такое *железное доказательство* в понимании Джека. Джек хочет, чтобы доказатесльство было формальным, а это значит, что оно должно содержать формулы, более того, формулы в КНФ (конъюнктивной нормальной форме). *Формула в КНФ* — это конъюнкция (логическое и) конечного количества дизъюнктов. *Дизъюнктом* называется дизъюнкция (логическое или) конечного количества *литералов*. *Литералом* называется или переменная (например Y), или отрицание переменной (например $\neg Y$). Каждая переменная может быть равна или нулю (логическая "ложь"), или единице (логическая "истина"). Например, дизъюнкт может выглядеть как $x \vee y \vee \neg z$. А формула в КНФ — как $(x \vee y) \wedge (\neg y \vee z)$.

Обозначим как $L(\varphi)$ множество всех переменных, входящих в формулу φ, и их отрицаний. Например, $L((x \vee z) \wedge \neg y) = \{x, y, z, \neg x, \neg y, \neg z\}$. Назовём биекцию $\pi : L(\varphi) \to L(\varphi)$ *автоморфизмом*, если выполняются следующие два условия:

- Для каждого литерала x: $\pi(\neg x) = \neg\pi(x)$ (мы предполагаем, что $\neg\neg\alpha = \alpha$).
- Если все литералы в формуле φ одновременно заменить на их образы в отображении π и получить формулу $\pi(\varphi)$, то $\pi(\varphi)$ является той же самой формулой, что и φ, с точностью до порядка дизъюнктов и литералов в дизъюнкте.

Например:

- У x есть только один автоморфизм: тождественное отображение.
- У $x \vee \neg y$ есть два автоморфизма: тожественное отбражение и ещё одно, которое отбражает x в $\neg y$, $\neg x$ в y, y в $\neg x$ и $\neg y$ в x. Последнее превращает формулу в $\neg y \vee x$, которая эквивалентна $x \vee \neg y$.
- У $(x \wedge x) \vee y$ есть только один автоморфизм: тождественное отображение.

Джек называет формулу φ *невыполнимой*, если невозможно выбрать такие значения переменных из $\{0, 1\}$, что φ после подстановки этих значений равна 1. Например, формула $\varphi = x \wedge \neg x$ является невыполнимой: $\varphi\big|_{x=0} = \varphi\big|_{x=1} = 0$.

Железным опровержением формулы φ (в КНФ) в терминах Джека называется последовательность дизъюнктов $C_1, \ldots, C_k$, в которой $C_k = \square$ ($\square$ означает пустой дизъюнкт, который никогда не выполнен) и для каждого $i \in \{1, \ldots, k\}$ выполнено одно из следующих условий:

1. $C_i \in \varphi$, то есть C_i — это один из дизъюнктов изначальной формулы.
2. C_i получается применением правила резолюции из двух дизъюнктов C_j и C_k, встречавшихся ранее в последовательности (то есть $j, k < i$). Они должны иметь специальный вид: $C_j = A \vee x$ и $C_k = B \vee \neg x$ для каких-то (возможно, пустых) дизъюнктов A, B и переменной x. В этом случае C_i должно быть равно $A \vee B$.

   Обратите внимание, что A и B могут пересекаться (то есть иметь общие литералы).

   Например, если есть дизъюнкты $C_j = (x \vee y \vee t)$ и $C_k = (x \vee \neg y \vee s)$, то можно получить $C_i = (x \vee t \vee s)$ применением правила резолюции по переменной y.
3. Есть один из предыдущих дизъюнктов C_j ($j < i$) и автоморфизм π формулы φ, такой, что $C_i = \pi(C_j)$. Например, если $\varphi = x \vee \neg y$, то есть единственный нетривиальный автоморфизм π, который переименует x в $\neg y$ и так далее (смотри выше).

Определим формулу ODD_G в КНФ, которая будет выполнимой тогда и только тогда, когда возможно уничтожить коридоры с соблюдением свойства, которого хочет Джек. В ней будет m логических переменных $e_1, \ldots, e_m$, которые могут принимать значения из множества $\{0, 1\}$: $e_i = 0$, если i-й коридор уничтожен, и $e_i = 1$, если нет. Теперь необходимо записать факт, что у i-й комнаты правильная чётность. Обозначим за d_i количество коридоров у комнаты i, а переменные, соответствующие этим коридорам — как $c_{i,1}, \ldots, c_{i,d_i}$ (например, $c_{i,1}$ может быть равно e_4, если у комнаты i есть коридор с номером 4). Теперь заметим, что у комнаты правильная чётность, только если у неё не неправильная чётность. А этот факт может быть записан как КНФ-формула из 2^{d_i-1} дизъюнктов, каждый из которых содержит d_i литералов. В итоге мы получаем формулу, которая проверяет, что конкретные значения переменных удовлетворяют условию Джека:

$$\mathrm{ODD}_G(e_1, \ldots, e_m) = \bigwedge_{i=1}^{n} \bigwedge_{\substack{s \in \{0,1\}^{d_i} \\ (s_1 + \ldots + s_{d_i}) \bmod 2 \neq a_i}} \underbrace{\left(\bigvee_{\substack{1 \leqslant j \leqslant d_i \\ s_j = 0}} c_{ij} \vee \bigvee_{\substack{1 \leqslant j \leqslant d_i \\ s_j = 1}} \neg c_{ij} \right)}_{\text{правда, только если переменные } c_{i1} \ldots c_{i,d_i} \text{ не имеют набор значений } s_1, \ldots, s_{d_i}}$$

Обратите внимание, что эта формула эквивалентна

$$\bigwedge_{i=1}^{n} \left(c_{i1} \oplus \ldots \oplus c_{i,d_i} \oplus \neg a_i \right),$$

где $\oplus$ — это исключающее или, но последняя формула не записана в КНФ, а потому не может быть использована в доказательстве напрямую.

Найдите *железное опровержение* формулы ODD_G, содержащее не более 1000 дизъюнктов.

Формат входных данных

В первой строке записаны два целых числа: n и m ($3 \leqslant n \leqslant 73$; $0 \leqslant m \leqslant 492$). Во второй строке находятся n целых чисел: $a_1, \ldots, a_n$ ($a_i \in \{0, 1\}$). Далее i-я из следующих m строк содержит описание i-го коридора: два целых числа $u, v \in \{1, \ldots, n\}$. Это значит, что i-й коридор соединяет комнаты u и v.

Две комнаты не могут быть соединены более чем одним коридором, коридор не может соединять комнату с самой собой. Гарантируется, что по коридорам есть путь между любыми двумя комнатами.

Формат выходных данных

Всегда, когда требуется описать литерал в формуле, целое число t обозначает переменную e_t, а целое число $-t$ обозначает её отрицание — $\neg e_t$. Ребра и соотвествующие переменные пронумерованы от 1 до m в порядке ввода.

Вывод должен содержать не более, чем 1000 строк. i-я из них должна описывать дизъюнкт доказательства C_i и должна иметь один из следующих видов:

- `declare` q l_1 $\ldots$ l_q, если в качестве C_i нужно взять один из дизъюнктов формулы ODD_g. Здесь q должно быть равно количеству литералов в этом дизъюнкте, а $l_1, \ldots, l_q$ должны описывать сами литералы. Все перечисленные литералы должны быть различны.
- `resolve` j k t. Соотвествует применению правила резолюции. j и k — номера дизюнктов в правиле, t — номер переменной. Должно быть верно, что $1 \leqslant j, k < i$, $t \in \{1, \ldots, m\}$, а кроме того, для некоторых дизъюнктов A и B верно $C_j = A \vee e_t$ и $C_k = B \vee \neg e_t$. После применения правила C_i будет равно $A \vee B$.
- `map` j a_1 $\ldots$ a_m. Соотвествует применению правила автоморфизма. Каждое a_t должно быть равно литералу, в который должно быть отображёно e_t. Отображение $e_t \mapsto a_t$, $\neg e_t \mapsto \neg a_t$ должно быть корректным автоморфизмом ODD_G в соотвествии с определением в условии. C_i будет равно $\pi(C_j)$, где π — это отображение, описанное выше.

Доказательство будет считаться корректным, только если последний дизъюнкт в нём — пустой.

Пример

oddities.in	oddities.out	Пояснение
3 3 1 0 0 1 2 2 3 1 3	declare 2 -2 1 declare 2 2 -3 resolve 2 1 2 map 3 -3 -2 -1 declare 2 -1 -3 resolve 4 5 1 map 6 -1 -2 -3 resolve 7 6 3	$C_1 = \neg e_2 \vee e_1$ $C_2 = e_2 \vee \neg e_3$ $C_3 = e_1 \vee \neg e_3$ $C_4 = \neg e_3 \vee e_1$ $C_5 = \neg e_1 \vee \neg e_3$ $C_6 = \neg e_3$ $C_7 = e_3$ $C_8 = \square$

Задача I. Сортировка на плоскости

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

Это интерактивная задача.

Есть n векторов на плоскости, ненулевых и попарно неколлинеарных. Вектор с номером i идёт из начала координат в точку (x_i, y_i). Но эти координаты мы вам не скажем.

Вместо этого вы можете задавать вопросы следующего вида: «Верно ли, что векторы i и j образуют правую пару?» Формально векторы образуют правую пару, если $x_i \cdot y_j > x_j \cdot y_i$. Геометрически правая пара означает, что, если мы стоим в начале координат и смотрим на конец вектора i, то, чтобы как можно скорее повернуться к концу вектора j, следует поворачиваться против часовой стрелки.

Нужно, задавая жюри вопросы, прийти к одному из двух выводов:

1. Все векторы лежат в одной полуплоскости, граница которой проходит через начало координат. Тогда нужно отсортировать их: вывести такой набор различных индексов $i_1, i_2, \ldots, i_n$, что для любых $p < q$ векторы i_p и i_q образуют правую пару.

2. Нет такой полуплоскости, граница которой проходит через начало координат и в которой лежат все векторы. Тогда нужно доказать это: вывести такую последовательность различных индексов $i_1, i_2, \ldots, i_k$, что каждый вектор, кроме последнего, образует правую пару со следующим, а последний (i_k) образует правую пару с первым (i_1).

Протокол взаимодействия

Сначала вашей программе подаётся в отдельной строке число n — количество векторов ($1 \leqslant n \leqslant 500$). Векторы в каждом тесте зафиксированы заранее, но держатся в секрете.

Затем вы можете делать следующее:

1. Спросить у жюри: «верно ли, что векторы i и j образуют правую пару?»

   Для этого ваша программа должна вывести строку следующего вида: «`? i j`». Индексы должны быть корректными: $1 \leqslant i, j \leqslant n$.

   В ответ программа жюри выдаст в отдельной строке число: 1, если ответ «да», и 0, если ответ «нет».

   Чтобы предотвратить буферизацию вывода, после каждого выведенного вопроса 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.

   Вы можете задать не более 20 000 вопросов.

2. Вывести ответ. В этом случае ваша программа должна вывести две строки.

   Если все векторы лежат в одной полуплоскости, первая строка должна иметь вид «`! YES`», а во второй должны быть выведены через пробел n различных чисел от 1 до n: индексы векторов в порядке сортировки.

   Если такой полуплоскости нет, выведите в первой строке «`! NO`». В начале второй строки выведите k — число векторов в доказательстве, а затем выведите k различных чисел от 1 до n — само доказательство. Если возможных доказательств несколько, выведите любое из них.

   После вывода ответа ваша программа должна сразу корректно завершить работу.

Примеры

стандартный ввод	стандартный вывод
3 0 1 0	 ? 1 3 ? 3 2 ? 2 1 ! YES 3 1 2
3 1 0 0	 ? 1 2 ? 3 2 ? 1 3 ! NO 3 1 2 3

Задача J. Середина списка сумм

Имя входного файла:	`sums-center.in`
Имя выходного файла:	`sums-center.out`
Ограничение по времени:	4 секунды
Ограничение по памяти:	256 мебибайт

Даны два массива a и b одинаковой длины n.

Рассмотрим все n^2 возможных попарных сумм $a_i + b_j$ и выпишем их в порядке нестрогого возрастания. Найдите n чисел, стоящих в точности в середине этого списка, то есть n элементов отсортированного списка с индексами от $\frac{n \cdot (n-1)}{2} + 1$ до $\frac{n \cdot (n+1)}{2}$. Элементы списка нумеруются с единицы.

Формат входных данных

В первой строке задано число n ($n \leqslant 2 \cdot 10^5$).

Во второй строке заданы n чисел a_1, a_2, …, a_n ($0 \leqslant a_i \leqslant 10^9$).

В третьй строке заданы n чисел b_1, b_2, …, b_n ($0 \leqslant b_i \leqslant 10^9$).

Формат выходных данных

Выведите n чисел — середину отсортированного списка попарных сумм в порядке их следования.

Примеры

sums-center.in	sums-center.out
2 1 3 2 4	5 5
3 1 2 3 4 6 8	7 8 9

Пояснения к примерам

В первом примере отсортированный список попарных сумм имеет следующий вид: 3, 5, 5, 7.

Во втором примере список такой: 5, 6, 7, 7, 8, 9, 9, 10, 11.

Задача K. Печеньки

Имя входного файла:	`word-chains.in`
Имя выходного файла:	`word-chains.out`
Ограничение по времени:	4 секунды
Ограничение по памяти:	256 мебибайт

На подносе лежало в ряд несколько печенек. Каждая печенька была выполнена в форме маленькой буквы английского алфавита, а вместе они составляли английское слово.

К подносу подошла Ева и решила незаметно съесть одну печеньку, а оставшиеся расставить так, чтобы они опять составляли слово. Мероприятие увенчалось успехом, никто из окружающих ничего не заметил. Тогда Ева решила повторить — и повторяла несколько раз, до тех пор пока у нее была возможность забрать одну печеньку и либо составить слово из оставшихся, либо оставить поднос пустым.

Ева не переворачивала и не поворачивала печеньки, так что никакая буква не превращалась в процессе в другую букву. О том, какие слова существуют, а какие нет, Ева справлялась в словаре.

Как мог выглядеть поднос в разные моменты времени, если Еве удалось съесть максимальное количество печенек?

Вам придётся решить эту задачу для нескольких подносов.

Формат входных данных

В первой строке ввода указано количество тестовых случаев n ($1 \leqslant n \leqslant 10^4$). В следующих n строках даны слова, по одному в строке. Каждое слово задаёт одно исходное состояние подноса. Далее в отдельной строке указан размер словаря: $m = 173\,554$ слова. В последующих m строках заданы слова из словаря, также по одному в строке. Во всех тестах словарь одинаковый. Это свободно распространяемый словарь ENABLE для игр со словами на английском языке, немного отредактированный для этой задачи (добавлены слова из одной буквы). Используемую версию словаря можно также скачать отдельно здесь: `http://acm.math.spbu.ru/171015/words.unix.txt` с переводами строк для Unix или `http://acm.math.spbu.ru/171015/words.windows.txt` с переводами строк для Windows. Гарантируется, что все исходные слова присутствуют в словаре. Не гарантируется, что с каждого подноса Еве удастся хоть что-то съесть.

Формат выходных данных

Выведите n цепочек, каждую в двух строках. В первой строке выведите длину цепочки, то есть количество состояний в ней. Во второй строке выведите все состояния подноса в этой цепочке. Каждое состояние подноса выводите как слово из словаря, или как точку («`.`»), если на подносе пусто. Состояния разделяйте с помощью последовательности символов «` -> `». Читайте пример для лучшего понимания формата вывода.

Если для исходного слова существует несколько цепочек максимальной длины, выведите любую из них.

Пример

word-chains.in	word-chains.out
5 university championships open cup cookie 173554 a aa aah ... zyzzyva zyzzyvas	11 university -> intrusive -> neuritis -> unities -> seniti -> nisei -> sine -> sei -> es -> e -> . 2 championships -> championship 5 open -> one -> ne -> e -> . 4 cup -> up -> p -> . 1 cookie

Пояснение к примеру

Как можно заметить, словарь в тексте условия приведён в сокращённом виде. Для того, чтобы получить верный ответ на пример, стоит заменить текст словаря на полный.

Строка в выводе для тестового случая «university» разбита на три части только для удобства чтения условия. Выводите всю цепочку в одной строке.

Задача L. Xor-справедливое разделение

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

Мальчики Артур и Лёшка нашли массив неотрицательных чисел A. Так как массив один, а мальчиков двое, они решили поделить элементы массива между собой так, чтобы каждый элемент массива достался кому-то из ребят и у каждого оказалось хотя бы по одному элементу.

Так как мальчики очень любят побитовую операцию «xor» ($\bigoplus$), они решили поделить массив xor-справедливо. Разделение массива A на два массива B_1 и B_2 xor-справедливо, если

$$\bigoplus_{x \in B_1} x = \bigoplus_{y \in B_2} y.$$

Теперь мальчиков заинтересовал вопрос: сколькими способами можно поделить элементы массива, выполнив все ограничения? Два способа считаются различными, если найдётся хотя бы один элемент, который в этих двух способах достался разным мальчикам.

Формат входных данных

В первой строке записано целое число n — количество элементов массива A ($1 \leqslant n \leqslant 50$).

Во второй строке задан список из n неотрицательных целых чисел $A_1, A_2, \ldots, A_n$ — элементы массива ($0 \leqslant A_i < 2^{63}$).

Формат выходных данных

Выведите одно целое число — сколькими способами Артур и Лёшка могут xor-справедливо поделить элементы массива A между собой.

Примеры

xorsep.in	xorsep.out
3 0 1 2	0
2 1 1	2

Пояснения к примерам

Во втором примере есть два способа разделения: Лёшка может взять себе первый элемент массива, а Артур — второй, и наоборот.