Задача A. 12:30PM

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

Люди из стран, где обычно для обозначения времени используется 24-часовая система, часто путаются, когда попадают в страну, где используется 12-часовая система (AM и PM). Ведь в этой системе есть несколько относительно нелогичных моментов, например, 12:00PM означает полдень и наступает сразу после 11:59AM. Но еще хуже жителям одной планеты в системы Альдебарана. Они используют особо сложную и неудобную систему обозначения времени.

На этой планете используют две системы обозначения времени. В каждое системе n-часовой день делится на несколько отрезков различной длины и в каждом отрезке часы отсчитываются независимо.

В система Альде день делится на m отрезков. Первый отрезок содержит a_1 часов и обозначается AA, второй отрезок содержит a_2 часов и обозначается AB, $\ldots$, m-й отрезок содержит a_m часов и обозначается A\$, где \$ — m-й символ латинского алфавита.

В система Баран день делится на k отрезков. Первый отрезок содержит b_1 часов и обозначается BA, второй отрезок содержит b_2 часов и обозначается BB, $\ldots$, k-й отрезок содержит b_k часов и обозначается B#, где # — k-й символ латинского алфавита.

Однако есть небольшая проблема. На самом деле все не так просто. В каждой системе день действительно делится на отрезки соответствующей длины, но порядок отрезков в каждом дне не фиксирован. Так, например, в один из дней отрезки в системе Альде могут идти в порядке 3, 2, 1, а отрезки в системе Баран — в порядке 1, 4, 2, 3 (если в этих системах 3 и 4 отрезка, соответственно).

Однажды к вам в руки попал дневник жителя этой планеты. Он делал отметки о различных интересных моментах времени в обеих временных системах. Так что у вас есть некоторая информация о соответствии времен в различных системах. Можете ли вы выяснить порядок следования отрезков в обеих системах в этот день?

Формат входного файла

Первая строка входного файла содержит n — количество часов в дне на планете ($1 \le n \le 99$). Вторая строка содержит число m, после которого следуют числа $a_1, a_2, \ldots, a_m$ ($1 \le m \le 8$, $1 \le a_i \le n$, сумма всех a_i равна n). Третья строка содержит число k, после которого следуют числа $b_1, b_2, \ldots, b_k$ ($1 \le k \le 8$, $1 \le b_i \le n$, сумма всех b_i равна n).

Четвертая строка входного файла содержит l — количество известных соотношений между временами в обеих системах ($0 \le l \le 100$). Следующие l строк содержат соотношения. В каждом соотношении времена заданы как XX:YYCC, где XX — количество часов, YY — количество минут, а CC означает название отрезка. XX изменяется от 00 до 98 (к счастью, жители планеты используют 00:00AA, 00:01AA, и т.д. для обозначения времени в первом часе отрезка), YY изменяется от 00 до 59, CC изменяется от AA до A\$ или от BA до B#.

Формат выходного файла

Если вы не можете восстановить информацию о порядке следования отрезков в дне в обеих системах из-за неоднозначности, выведите “`Ambiguous`” во входной файл. Если не существует ни одного способа восстановить порядок следования отрезков, выведите “`Inconsistent`”. В противном случае выведите порядок следования отрезков в каждой из систем, следуя формату, приведенному в примере.

Примеры

ampm.in	ampm.out
24 2 14 10 2 12 12 1 06:30AA=04:30BA	Alde system: 2 1 Baran system: 2 1
24 2 12 12 2 12 12 1 11:30AA=11:30BB	Ambiguous
24 2 12 12 2 12 12 1 11:30AA=11:31BB	Inconsistent

В первом примере если в системе Альде 14-часовой отрезок был бы первым, то 06:30AA соответствовало бы 06:30BA (или 06:30BB). Так что 10-часовой отрезок идет первым. А значит в системе Баран первым идет отрезок BB.

Задача B. Шары

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

Злой волшебник Лиахи взял два шара бесконечно малого размера и одинаковой массы и расположил их в различных точках с целочисленными координатами x_i на вещественной оси. Но на этом Лиахи не успокоился. В момент времени 0 он придал i-му шару скорость v_i и стал наблюдать за поведением шаров. Но все происходило очень медленно, а Лиахи интересно, где окажутся и какие скорости будут иметь шары в момент времени T. Шары сталкиваются абсолютно упругим образом. Так как шары имеют одинаковую массу, то это означает, что после столкновения каждый шар движется со скоростью другого шара, также перенимая у него и направление движения.

Злой маг требует от вас написать программу, которая выводила бы состояния шаров в момент времени T. Если программа будет работать неправильно, то он поместит вас в один из этих шаров.

Формат входного файла

Первая строка входного файла содержит два числа — x_1 и v_1. Вторая строка — x_2 и v_2. В третьей строке содержится единственное число T. Все числа во входном файле целые и не превосходят по абсолютной величине 10^4.

Формат выходного файла

В выходной файл выведите две строки: в первой строке — местоположение и скорость первого шара в момент времени T, во второй строке — второго. Все значения округляйте к ближайшему целому. Числа в каждой строке должны быть разделены пробелом. Гарантируется, что в момент времени 0 и в момент времени T координаты шаров различны.

Примеры

balls.in	balls.out
1 2 2 2 5	11 2 12 2
8 2 12 -2 10	-8 -2 28 2

Задача C. Блэкджек

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

Блэкджек представляет собой популярную азартную игру, использующую одну или несколько колод карт. В этой задаче описывается упрощенная версия игры, в которой некоторые действия, разрешенные в казино, не используются. Как можно увидеть из решения этой задачи, в этой игре, как и в остальных азатрных играх, практикуемых в казино, игрок имеет отрицательное математическое ожидание выигрыша.

Используется одна колода игральных карт, содержащая карты со следующими значениями: A (туз), 2, 3, 4, 5, 6, 7, 8, 9, T (десятка), J (валет), Q (дама) и K (король), в колоде четыре карты каждого значения. Каждой карте соответствует определенное количество очков: картам от 2 до 9 соответствует количество очков, равное их значению, T, J, Q и K соответствует 10 очков, тузу по желанию игрока может соответствовать 1 или 11 очков. Цель игры — набрать максимальное количество очков, не превышающее 21.

Игрок делает ставку в 1 доллар. Игроку сдаются две карты, одна карта сдается казино. Игрок может либо взять еще одну карту (“hit”), либо остановиться (“stand”). Если игрок решает взять еще одну карту, ему сдается еще одна карта и у него снова есть аналогичный выбор. Игрок может брать карты пока он либо не наберет 21 очко, либо количество очков у него не превысит 21. В последнем случае игрок немедленно проигрывает. Когда подсчитываются очки, туз считается как 11 очков, если при этом количество очков у игрока не превышает 21, иначе туз считается как 1 очко.

Если игрок превысил 21 очко, он проигрывает и теряет свою ставку. Иначе представитель казино сдает карты себе. Сначала он сдает себе вторую карту. После этого он берет карты до тех пор, пока не наберет хотя бы 17 очков. Когда подсчитываются очки, туз считается как 11 очков, если при этом количество очков у казино не превышает 21, иначе туз считается как 1 очко. Например, если у казино A и 5, то он должен взять карту. Если он берет T, то он должен взять еще одну карту, поскольку A, 5 и T в сумме дают 16 очков.

Если казино набирает больше 21 очка, то игрок выигрывает и получает обратно удвоенную ставку. В противном случае, если у игрока больше очков, чем у казино, то он выигрывает и получает обратно удвоенную ставку, если у казино больше очков, то игрок проигрывает и теряет свою ставку, если у игрока и казино поровну очков, то игра считается завершившейся вничью и игрок получает обратно свою ставку, но ничего не выигрывает.

В отличие от большинства игр, практикуемых в казино, например рулетки, в блэкджеке у игрока есть выбор — взять еще карту или остановится, таким образом игрок может повлиять на результат игры. Для каждой комбинации из двух карт у игрока и одной карты у казино у игрока существует оптимальная стратегия, которая максимизирует ожидаемый средний выигрыш игрока (усреднение проводится по всем возможным вариантам расположения остальных карт в колоде). Ваша задача — найти эту выигрышную стратегию и сообщить, следует ли брать карту, либо остановиться.

Формат входного файла

Входной файл содержит три символа, разделенных пробелом: карты игрока и карту казино.

Формат выходного файла

Выведите “`HIT`”, либо “`STAND`”.

Примеры

blackjack.in	blackjack.out
2 3 A	HIT
2 Q 5	STAND

В первом примере у игрока 2 и 3, которые в сумме дают 5 очков. Не брать карту не имеет смысла — очки игрока увеличатся, а превысить 21 невозможно.

Второй пример — один из наиболее удивительных математических результатов относительно блэкджека. Имея “твердые 12” (12 очков без туза, который считается за 11) против 5 у казино, лучше остановиться и не брать карту, хотя вероятность превысить 21 кажется небольшой. Если взять еще одну карту, то ожидаемый выигрыш равен -0.1636 доллара, а если не брать, то -0.144.

Задача D. Разработка микросхем

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

Разработка микросхем в рамках электротехники отличается от исследования логических схем, скажем, в теории сложности. При создании микросхем необходимо, помимо всего прочего, учитывать временные задержки, которые происходят между изменением сигнала на входе логического элемента и его срабатыванием, что влечет изменение сигнала на выходе, а также конечность скорости распространения электрических сигналов по проводам.

Логическая схема представляет собой ациклический ориентированный граф, каждая вершина которого представляет собой либо вход, либо выход, либо логический элемент. Входы логической схемы не имеют входящих ребер и могут иметь несколько исходящих. Выходы логической схемы не имеют исходящих ребер и имеют ровно одно входящее. В данной задаче рассматриваются следующие виды логических элементов: «или» (`or`), «и» (`and`) и «не» (`not`). Элементы «или» и «и» имеют ровно два входящих ребра и могут иметь любое количество выходящих ребер, элемент «не» имеет ровно одно входящее ребро и может иметь любое количество исходящих ребер.

Если в графе есть ребро uv, то вершина u называется *предшественником* вершины v.

Каждая вершина может иметь одно из двух логических значений: 0 или 1. Значения входов схемы являются ее параметрами — это «входные данные», передаваемые схеме. Значения остальных вершин вычисляются следующим образом.

Выход логической схемы имеет ровно одного предшественника, его значение равно значению предшественника.

Элемент «или» равен 1, если хотя бы один из его предшественников равен 1.

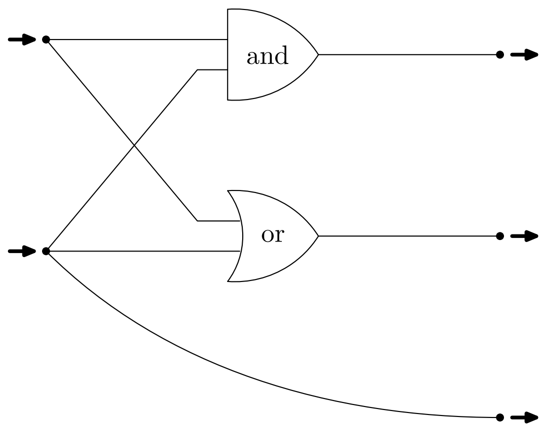
Элемент «и» равен 1, если оба его предшественника равны 1.

Элемент «не» равен 1, если его предшественник равен 0.

Поскольку граф является ациклическим, приведенное описание однозначно задает значения всех вершин при известных значениях входов логической схемы.

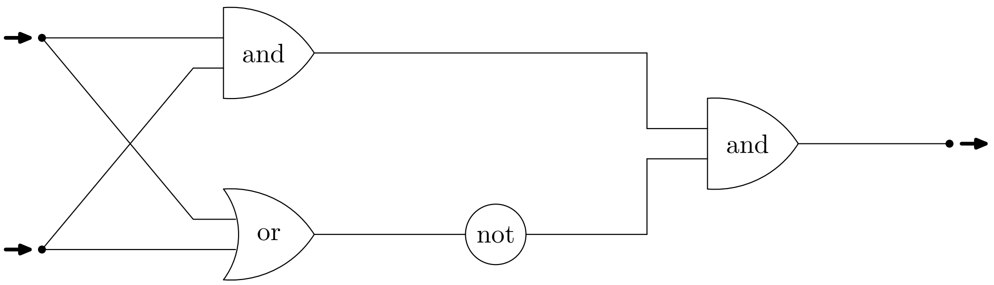
Если значение на некотором входе логической схемы изменяется, то могут измениться и значения других вершин схемы. Однако при изменении значений одновременно на нескольких входах, могут наблюдаться эффекты рассинхронизации схемы из-за небольших задержек между изменениями. Также эти эффекты могут наблюдаться из-за конечности скорости распространения сигналов по проводам. Эти эффекты могут приводить к тому, что некоторые вершины могут временно содержать некорректные значения — так называемые *паразитные значения*.

Например, рассмотрим схему, приведенную на следующем рисунке.



Если изменить значения на входах с $(0,1)$ на $(1,0)$, и первый вход изменит свое значение с 0 на 1 до того, как второй вход изменит свое значение с 1 на 0, это может привести к тому, что верхний выход схемы будет временно содержать 1, хотя он должен быть равен 0 и для начальных и для конечных значений входов.

Еще более неожиданные эффекты могут возникнуть из-за задержек, связанных с конечностью распространения сигналов по проводам. Рассмотрим схему, приведенную на следующем рисунке.



Кажется, что паразитные значения возникнуть не могут — какой бы вход не изменил свое значение первым, на выходе будет 0. Но это не так. Пусть снова входы схемы меняются с $(0, 1)$ на $(1, 0)$. Пусть оба входа изменяют свое значение одновременно, но расстояние от верхнего входа до элемента «и» намного меньше, чем до элемента «или». Наоборот, пусть расстояние от нижнего входа до элемента «и» намного больше, чем до элемента «или». Тогда «и» «узнает» об изменении верхнего входа раньше, чем об изменении нижнего и переключается на 1. Аналогично, «или» узнает об изменении нижнего входа раньше чем об изменении верхнего и переключается на 0. Элемент «не» изменяет значение на 1, и наконец самый правый элемент «и» и выход схемы изменяют свое значение на 1. Затем информация о переключениях постепенно «доходит» до всех элементов и они переключаются в правильные значения.

Чтобы формализовать описанный процесс, будем считать, что каждое ребро графа может иметь различные значения в начале и в конце. Когда изменяется значение в начале ребра, значение в конце ребра может измениться не сразу, а через некоторое время.

Рассмотрим логическую схему. Вам заданы начальные и конечные значения на всех входах схемы. Процесс изменения значений остальных вершин схемы называется ее *переключением*. В процессе переключения следующие события могут происходить в любом порядке:

- один вход схемы изменяет свое значение с начального на конечное, новое значение становится начальным значением всех исходящих из этой вершины ребер;
- один элемент или один выход схемы, который содержит значение, не соответствующее конечным значениям входящих в него ребер, изменяет значение на правильное, в результате этого новое значение становится начальным значением всех исходящих из этой вершины ребер;
- у одного из ребер конечное значение становится равно начальному значению.

Переключение завершается, когда все входы схемы изменили свое значение, начальные и конечные значения всех ребер совпадают, и значения всех вершин корректны.

Для каждого выхода схемы можно найти ее начальное значение i и конечное значение f. Рассмотрим значение в этой вершине перед переключением и после каждого события в процессе переключения. Выход схемы называется *безопасным* (safe), если его значение равно i до некоторого события, и затем равно f до окончания переключения (в частности, если $i = f$, то значение выхода вообще не должно меняться). Если для некоторой последовательности событий поведение значения на выходе иное, то выход называется *небезопасным*.

По заданной логической схеме и ее переключению выясните для каджого выхода, является ли он безопасным.

Формат входного файла

Первая строка входного файла содержит число n — количество вершин в логической схеме ($2 \le n \le 1200$). Следующие n строк описывают вершины. Каждая вершина описывается следующим образом: сначала указан тип вершины: «`in`», «`out`», «`and`», «`or`» или «`not`».

Если тип вершины — «`in`», то ее описание состоит из двух целых чисел, каждое из которых равно 0 или 1: начального и конечного значения на входе.

Если вершина имеет тип «`out`» или «`not`», то ее описание состоит из одного числа — номера ее предшественника.

Если вершина имеет тип «`and`» или «`or`», то ее описание состоит из двух чисел — номеров ее предшественников.

Вершины нумеруются от 1 до n в порядке, в котором они заданы во входном файле. Предшественники вершины всегда имеют номер, не превышающий номера вершины.

Формат выходного файла

Для каждого выхода выведите, является ли он безопасным, следуя формату, приведенному в примере.

Примеры

circuit.in	circuit.out
7 in 0 1 in 1 0 and 1 2 or 1 2 out 3 out 4 out 2	Node 5 is not safe. Node 6 is not safe. Node 7 is safe.
7 in 0 1 in 1 0 and 1 2 or 1 2 not 4 and 3 5 out 6	Node 7 is not safe.

Задача E. Странные цифры

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

В десятичной системе счисления мы используем цифры от 0 до 9. Если бы мы не использовали какую-нибудь цифру, то было бы невозможно записать некоторые числа, например, если бы мы не использовали цифру 1, то мы не смогли бы записать число 10. Наоборот, если бы мы использовали больше чем 10 цифр, то некоторые числа можно было бы записать несколькими способами. Например, если бы мы использовали цифру A (со значением, равным 10), то число 110 можно было бы также записать как AA. Аналогичные рассуждения можно провести в системе счисления с любым основанием b.

Рассмотрим систему счисления с основанием b и цифрами $c_1, c_2, \ldots, c_k$, выбранными из множества $\{0, 1, \ldots, 9, A, B, \ldots, Z\}$ (веса цифр от 0 до 9 равны их обычным значениям, вес цифры A равен 10, вес цифры B равен 11, и т. д., вес цифры Z равен 35). Вам задано число n. Выясните, можно ли записать его в системе счисления с основанием b, используя только заданные цифры, и если это так, то верно ли, что существует единственный способ это сделать.

Формат входного файла

Первая строка входного файла содержит b ($2 \le b \le 36$). Вторая строка содержит цифры $c_1, c_2, \ldots, c_k$ в возрастающем порядке, без пробелов ($1 \le k \le 36$, $c_1 = 0$). Третья строка содержит число n ($1 \le n \le 10^{100}$), оно записано в десятичной системе счисления.

Формат выходного файла

Если записать n невозможно, выведите в выходной файл слово “`Impossible`”.

Если число можно записать единственным способом, выведите в выходной файл “`Unique`”, а на второй строке выведите запись числа n в описанной системе счисления.

Если n можно записать несколькими способами, выведите “`Ambiguous`”, а на второй строке выведите любой способ записать n в описанной системе счисления.

Примеры

digits.in	digits.out
10 0123456789A 110	Ambiguous AA
10 023456789 10	Impossible
10 023456789A 10	Unique A

Задача F. Два формата

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

Байтом называется единица информации, состоящая из восьми двоичных разрядов. Таким образом, в одном байте можно хранить 2^8 различных вариантов целого числа — от 0 до 255. Для хранения бо́льших чисел в компьютере отводится несколько байт, в которые записываются разряды числа, переведенного в систему счисления с основанием 256. То есть для хранения числа $A = \sum_{i=0}^{k} 256^i \cdot A_i$ $(0 \le A_i \le 255)$ будет отведено $(k+1)$ байт, в которых будут записаны числа $A_0, A_1 \ldots, A_k$. Например, для записи числа 2000000 потребуется 3 байта: $2000000 = 128 \cdot 256^0 + 132 \cdot 256^1 + 30 \cdot 256^2$. Соответственно, в этих байтах будут записаны числа 128, 132 и 30.

В разных устройствах принято хранить байты одного числа в разном порядке. Самыми распространенными порядками являются *тупоконечный* (*big-endian*) и *остроконечный* (*little-endian*). Тупоконечным называют порядок от байта, соответствующего старшему разряду числа в системе счисления с основанием 256, к байту, соответствующему младшему, то есть в порядке $A_k, A_{k-1}, \ldots A_1, A_0$. В случае с числом 2000000 это будет порядок 30, 132, 128. Остроконечным называют порядок, обратный тупоконечному.

Теперь представим, что нам нужно передать число с устройства, хранящего байты в тупоконечном порядке, на устройство, хранящее байты в остроконечном порядке. Однако на втором устройстве байты этого числа будут восприниматься как, возможно, совсем другое число. Например, при передаче числа 2000000 будут переданы байты в следующем порядке: 30, 132 и 128. Второе устройство воспримет это как $30 \cdot 256^0 + 132 \cdot 256^1 + 128 \cdot 256^2 = 8422430$.

Пусть известно число N, записанное на втором устройстве (хранящем байты в остроконечном порядке) в результате передачи на него числа M с первого (хранящего байты в тупоконечном порядке). Требуется восстановить число M.

Формат входного файла

В первой строке входного файла задано единственное целое число N $(1 \le N < 2^{24})$.

Формат выходного файла

В выходной файл выведите число M.

Примеры

formats.in	formats.out
8422430	2000000
257	257

Задача G. Бряк

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

— Эй, Брейн, чем мы будем заниматься сегодня вечером?
–– Тем же, чем и всегда, Пинки... Попробуем завоевать мир!
Пинки и Брейн.

Как-то раз, в одной очень секретной лаборатории две чрезвычайно сообразительные мыши выбрались из клетки и решили завоевать мир. Для этого они пробрались в хранилище, где в нескольких емкостях содержался раствор совершенно секретного вещества. В разных емкостях объемы и концентрации растворов могли различаться.

Для осуществления своего коварного плана мышам потребовался раствор определенной концентрации. Для того, чтобы получить его они стали переливать некоторые объемы раствора из одной емкости в другую. К большому сожалению, как раз в тот момент, когда нужный раствор уже был получен, пришли охранники, и мышам пришлось скрыться.

К несчастью для зверьков, камеры наблюдения записали все, что происходило в хранилище. Ученым, работающим в этой лаборатории крайне интересно, что же хотели сделать их подопытные, поэтому они хотят по записи действий мышей определить концентрацию раствора в каждой из емкостей.

Формат входного файла

Первая строка входного файла содержит целое число n — количество емкостей в хранилище ($2 \le n \le 100$). Следующие n строк содержат пары целых чисел v_i и c_i, разделенные пробелом — объем жидкости в i-й емкости в литрах и концентрация раствора в ней в процентах ($0 \le v_i \le 10^9$, $0 \le c_i \le 100$). Будем считать, что в пустой емкости концентрация равна нулю.

Следующая строка входного файла содержит целое число m — количество операций, произведенных мышами ($1 \le m \le 100$). Далее следует m строк, содержащих три целых числа a, b и k ($1 \le a, b \le n$, $a \ne b$, $1 \le k \le 10^9$). Данная запись означает, что из сосуда с номером a перелили k литров в сосуд с номером b. При этом гарантируется, что в сосуде a содержится хотя бы k литров жидкости. Сосуды нумеруются с единицы в порядке их упоминания во входном файле. Будем считать, что все сосуды достаточно велики, чтобы в них поместился любой объем жидкости.

Формат выходного файла

В выходной файл выведите объемы и концентрации растворов в емкостях после всех переливаний в таком же формате, в каком они заданы во входном файле. Концентрация каждой жидкости должна отличаться от правильной не больше чем на 10^{-4} процента.

Примеры

narf.in	narf.out
2 10 50 10 100 1 1 2 10	2 0 0.0000 20 75.0000

Задача H. Расписание

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

Недавно Сережа устроился на работу секретарем. Чтобы не просиживать штаны просто так, он попросил нагрузить его какой-нибудь работой. Ох, и зря он это сделал!

Начальник поставил перед ним весьма непростую задачу. Оказалось, что есть много неподписанных документов, которые нужно найти, привести в порядок и отнести их на подпись шефу. Сережа — мальчик неторопливый, к тому же любит поспать на работе, поэтому в течение одного дня он может работать только с одной бумагой. К тому же каждый документ должен быть подписан до какого-то определенного срока. За каждый просроченный день по каждому просроченному документу у Сережи вычитают из зарплаты 100 рублей.

Сережа в недоумении, ведь он не знает в каком порядке ему стоит работать с документами, чтобы понести наименьшие потери. Помогите ему!

Формат входного файла

Первая строка входного файла содержит число документов N ($1 \le N \le 100$). Во второй строке через пробел записаны N чисел — сроки сдачи документов ($1 \le t_i \le 1000000$).

Формат выходного файла

В выходной файл выведите N чисел — номера дней, в которые Сережа должен обработать соответствующий документ. Эти числа не должны превосходить 10^8. Если решений несколько, выведите любое. Учитывайте, что Сережа не обязательно должен работать с документами каждый день.

Примеры

schedule.in	schedule.out
3 2 3 1	2 3 1
4 1 2 1 2	1 4 2 3

Задача I. Треугольник

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

Петя с Васей любят разные геометрические задачи и игры на их основе. Однажды, Петя придумал такую игру и рассказал про нее Васе.

Игра состоит в следующем — Петя рисует на плоскости любой невырожденный треугольник, потом отмечает середины сторон такого треугольника и стирает весь треугольник, оставляя только точки, обозначающие середины сторон. Вася, в свою очередь, должен по этим данным восстановить исходную фигуру.

Немного подумав, Вася решил, что он всегда сможет однозначно восстановить исходный треугольник, если данные ему три точки не лежат на одной прямой. Для доказательства этого факта он попросил Вас написать программу, делающую это.

Формат входного файла

В трех строках входного файла заданы по два целых числа, не превосходящих 1000 по абсолютной величине — координаты середин сторон треугольника. Данные три точки не лежат на одной прямой.

Формат выходного файла

В выходной файл выведите ответ на задачу: три строки, содержащие по два целых числа — координаты вершин исходного треугольника в любом порядке.

Примеры

triangle.in	triangle.out
1 0 0 1 1 1	0 0 2 0 0 2