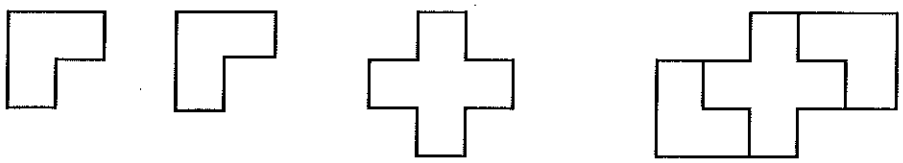
Problem A **Solving puzzles**

A friend of yours has just purchased a set of puzzles, and is having difficulty solving them by hand. Solving a puzzle involves taking one or more pieces, and arranging them into a specified shape. A piece is composed of "basic squares"; all basic squares are the same size. If a piece consists of more than one basic square, then each basic square is joined along at least one edge to one of the other basic squares in the piece. Pieces cannot contain holes. The same rules apply to shapes.

A solution for a puzzle is some arrangement of pieces that produces the desired shape. Pieces can be rotated as desired, but cannot be placed upside down. For example, the three puzzle pieces shown on the left below can be placed together to form the shape on the right. The individual basic squares in the puzzle pieces are not shown, just the outline of the whole piece; the number of basic squares in each piece is 3, 3 and 5 respectively from left to right, and the shape has 11 basic squares (3x5 - 4).



For this problem, pieces and target shapes are described by "maps". A map starts with a line containing two integers (ROWS and COLS) that specify the dimensions of the minimum rectangle which encloses the whole shape/piece. This is followed by ROWS lines giving the basic squares present in the shape/piece. Each row contains COLS digits, each a 0 or 1. A 0 indicates that there is no basic square in that position of the shape/piece, a 1 indicates that there is a basic square in that position. For example, the three pieces and the shape above are described by the maps:

```
2 2     2 2     3 3     3 5
11      11      010     00111
10      10      111     11111
                010     11100
```

Input will be from the file named PROBLEMA.DAT and will consist of a number of puzzles. Each puzzle starts with a line containing an integer giving the number of pieces in the puzzle (N). This is followed by N maps describing the pieces, followed by a map describing the target shape. You are guaranteed that:

- for each puzzle, the number of 1s in the shape map is equal to the sum of the number of 1s in all the pieces of that puzzle.
- in every map, there will be at least one 1 in the top row, the bottom row, the left column, and the right column.
- there will be at most 8 pieces in every puzzle set.
- the number of rows in a map is in the range 1 to 8, and the number of columns is in the same range.
- the number of rows in a piece is no more than the number of rows in the shape. This also applies to columns.

The input is terminated by a line having the value 0 for N.

For every puzzle in the input your program must print details of a solution, or the message "No solution". If a solution is found, it should be printed in exactly the same form as the map for the target shape, except that all 0s are replaced by '-', and the 1s are replaced by a digit giving the number of the piece that occupies that position. The first piece that appears in the input for a puzzle is number 0, the second number 1, and so on. Where there are several solutions, the one printed should be the smallest, where solutions are compared as if they were strings, with all the rows concatenated together. A blank line must appear between the output for each puzzle, but not after the last puzzle.

EXAMPLE
INPUT

```
3
2 2
11
10
2 2
11
10
3 3
010
111
010
3 5
00111
11111
11100
2
2 1
1
1
1 2
11
2 3
010
111
0
```

OUTPUT

```
--200
12220
112--

No solution
```

Problem B **Erdös numbers**

Paul Erdös (pronounced "Erdish") is a Hungarian mathematician renowned for his prolific output of scientific papers in probability theory and combinatorics. Such a large proportion of these papers has been written in collaboration with other mathematicians that it is said that mathematicians may be divided into two groups, those who have co-authored a paper with Erdös, and those who have not. The second group may be further broken down using the concept of Erdös numbers, given by the following inductive definition:

(i) Paul Erdös has an Erdös number of zero.
(ii) A person's Erdös number is one more than the smallest Erdös number of the co-authors of all of the papers that person has ever written.
(iii) A person does not have an Erdös number if all of that person's co-authors also do not have Erdös numbers.

Using this definition, Einstein wrote a paper with Straus, who wrote a paper with Erdös. However, Einstein was never a co-author with Erdös. Therefore Einstein's Erdös number is two.

You have to write a program to process a list of co-authors of scientific papers and print the Erdös number of selected authors. No more than 250 authors and 250 papers are involved, and there are no more than 7 joint authors for any paper.

Input will be from a file PROBLEMB.DAT and will consist of two sections. The first section is a list of the co-authors of the papers selected for this study. Each paper has all its authors listed on one line (each line is less than 80 characters), with the names separated by commas, and each author's name is in the form: Lastname<at least 1 blank>Initials. Every author has at least one initial, and there are no spaces between the initials if there are more than one. The total number of letters in a name is at most 30. There are an arbitrary number of blanks separating the words and commas, and capitalisation is not consistent. This section is terminated by a line consisting of a single #, and is followed by a list of the authors whose Erdös numbers are to be calculated. Each line contains a single name and is less than 80 characters in length. There will be no more than 150 names in this list; the names may again have arbitrary numbers of blanks and arbitrary capitalisation. This section is terminated by a line consisting of a single #. Due to the limitations of the ASCII character set, all authors' names in the input and output are written without accents; in particular, Erdös will appear as Erdos, both in the input and the output.

Output will be one line for each name in the second section. The line must be of the form "<Name> has an Erdos number of <number>" or of the form "<Name> does not have an Erdos number" where <Name> must be exactly the same as the line in the second section, in terms of blanks and capitalisation, and is separated from the word "has" or "does" by one blank, and <number> has no leading zeros or blanks.

EXAMPLE
INPUT

```
Slepian D ,    Kac m
Erdos p,Kac    M
kac    m   , Wolf JK
   Wolf   jK   , Riding-Hood   RED
  Erdos    P, Riding-Hood   red
#
   Wolf          Jk
Wolf   TheBigBad
#
```

OUTPUT

```
   Wolf          Jk has an Erdos number of 2
Wolf   TheBigBad          does not have an Erdos number
```

Problem C **Bacterial Identification**

A local firm manufactures devices for identifying the species of bacteria which are present in various nasty specimens provided by patients. These identifying devices are trays holding "wells" of chemicals which change colour if a bacterium grows in them. There are normally 10 different chemicals per tray, and each species has a certain probability of growing in each chemical. For example, Yersinia Pestis and Enterobacter Cloacae (we will skip the description of what these do to you) have the following "profiles"; the probabilities given are the probability of the bacterium growing in the chemical and making it change colour:

Chemical No:	1	2	3	4	5	6	7	8	9	10
Y. Pestis Prob.:	0.00	0.01	0.00	0.70	1.00	0.00	0.50	0.00	0.97	0.00
E. Cloacae Prob.:	0.75	0.92	0.25	0.75	1.00	0.15	0.95	0.97	1.00	0.97

Thus if one of these identification trays has parts of a specimen grown in each of its wells, and chemicals 4, 5, 7 and 9 are observed to have changed colour (this is denoted by ---+-+-+-+-), then there seems to be a good chance that Yersinia Pestis is present; but note that even if Yersinia Pestis actually is present, the probability of this result happening is only 0.336105 (.99x.7x.5x.97)! This set of reactions could also mean that Enterobacter Cloacae is present, although in this case the probability of this particular pattern being observed is .25x.08x.75x.75x.85x.95x.03x.03 ie 0.00000818.

If a particular pattern is observed, it would be nice to be able to ignore all bacteria for which the reaction probability is less than P for some P (called the "minimum acceptable probability"). However, while some bacterium have a few, highly likely, characteristic reactions, others have a large number of roughly equally likely reactions, all of low probability. As an extreme case, a bacterium with a 0.5 probability of a positive reaction for every chemical has 1024 equally likely patterns, each with probability 0.000977. If P is set to this value, then rare patterns such as -+---+---+- become "acceptable" for Yersinia Pestis. The solution is to have a different P for each bacterium. For each bacterium, the probability of every possible reaction is worked out (most will be 0 for Yersinia Pestis), put into descending order and then added. When the total reaches or passes 0.95, the last probability added is the minimum acceptable probability. For Yersinia Pestis, it's easy to calculate that this is 0.144045.

Input will be from a file called PROBLEMC.DAT and will consist of information for a series of bacteria. There will be two lines for each bacterium; the first will give the name of the bacterium (two or more words, total length less than 60 characters), and the second will give the 10 positive reaction probabilities, as in the examples above. Each probability will be 1.00 or 0.ab, where a, b are digits. The probabilities will be separated by spaces. The file will be terminated by a line consisting of a single #

Output will be one line for each bacterium, giving the name of the bacterium (exactly as it appears in the input file) and its minimum acceptable probability as a decimal number, rounded to 10 decimal places, and separated from the name by a single blank.

EXAMPLE
INPUT

```
Yersinia Pestis
0.00 0.01 0.00 0.70 1.00 0.00 0.50 0.00 0.97 0.00
Enterobacter Cloacae
0.75 0.92 0.25 0.75 1.00 0.15 0.95 0.97 1.00 0.97
Yersinia Fredericksenii
1.00 0.00 0.00 0.95 0.00 0.70 0.00 0.00 1.00 0.15
#
```

OUTPUT

```
Yersinia Pestis 0.1440450000
Enterobacter Cloacae 0.0028491628
Yersinia Fredericksenii 0.0427500000
```

Problem D **Plumbing Costs**

In a multi-level building, pumps are used to pump water from a lower level to a higher level, and reservoirs are used to store water to be used in a gravity feed arrangement to lower levels. The cost of the pumps and reservoirs depends on how high they have to be placed in the building, and over how many floors they have to pump or distribute water.

The cost of a pump (in thousands of dollars) is: 80 + 2 * LEVEL + 4 * LEVELS-TO-NEAREST-RESERVOIR-BELOW + 8 * LEVELS-TO-NEAREST-RESERVOIR-ABOVE.
The cost of a reservoir (in thousands of dollars) is: 12 + LEVEL + 3 * LEVELS-TO-NEAREST-PUMP-BELOW + 4 * LEVELS-TO-NEAREST-PUMP-ABOVE.

Your task is to write a program to input a series of building pump and reservoir layouts and to determine the cost of implementing each layout.

Input will be from a file PROBLEMD.DAT and consists of two lines per building. The first line contains the number of pumps followed by the floor number of each pump, not in any particular order. The second line contains the number of reservoirs followed by the floor number of each reservoir. The floor numbers range between 1 and 100, and there will be no more than 20 pumps and 20 reservoirs. There will be at least one blank between each pair of numbers. No pumps or reservoirs are placed on the ground floor (floor 0), and there may be multiple pumps and reservoirs on the same floor. The definitions of NEAREST-ABOVE and NEAREST-BELOW do not include the pumps or reservoirs on the same floor. If there is no pump or reservoir above or below, then that part of the calculation is to be discounted. The input file is terminated by a single line with a zero, denoting no pumps.

There will be one output line for each building described in the input. Each output line is to contain a single number (right justified in a field of width 6) which is the cost (in thousands of dollars) for the corresponding building in the input.

EXAMPLE
INPUT

```
3 7 4 9
2 10 6
3 1 2 3
0
1 4
1 4
0
```

OUTPUT

```
   397
   252
   104
```

Problem E **Recurrences**

Many sequences (such as the Fibonacci sequence) are defined by recurrence relations, in which each number is defined as a linear combination of its predecessors:

$$a(i) = c_1a(i-1) + c_2a(i-2) + ... + c_na(i-n)$$

where c_1, c_2, ..., c_n are fixed constants and a(i) is the i'th term in the sequence The numbers in these sequences often have surprising relationships to each other, and it is useful to be able to generate any particular number in the sequence. Of course, if the sequence is defined by an n-term recurrence relation as above, then the first n terms a(1), a(2), ..., a(n) must be given independently.

For this problem, you will be given a recurrence relation, and you must generate a term whose number is given. No relation will have more than 3 coefficients, and you will not be asked for more than the 1000th term in any series. The size of the coefficients and the initial numbers are all less than 100 (this still allows for quite large numbers, however....).

Input will be from a file PROBLEME.DAT and will consist of a series of descriptions of recurrence relations, and the number of the term to find in the sequence generated by the relation. Each relation will be described by 4 lines. The first line will give the number of terms, n in the formula above. The second line will contain n integers between 0 and 99, which are the coefficients c_1, c_2, ..., c_n of the recurrence relation, in that order. The third line contains n integers between 0 and 99, which are the first n terms in the series, in the order: first term, second term, ..., nth term. The fourth line has a single number (1000 or less) which is the number of the term you must find. Where there is more than one number on a line, the numbers will be separated by one or more blanks. Each line is less than 80 characters in length. The file will be terminated by a line consisting of a single 0 (ie an n value of 0).

Output will be one number for each recurrence relation description. The number must be printed 60 digits per line, except possibly for the last line which can have less than 60 digits. The numbers must be separated by single blank lines; there must not be a blank line at the end of the output.

EXAMPLE

INPUT

```
3
1 0 0
97 98 99
1000
1
2
1
200
2
1 1
1 1
1000
0
```

OUTPUT

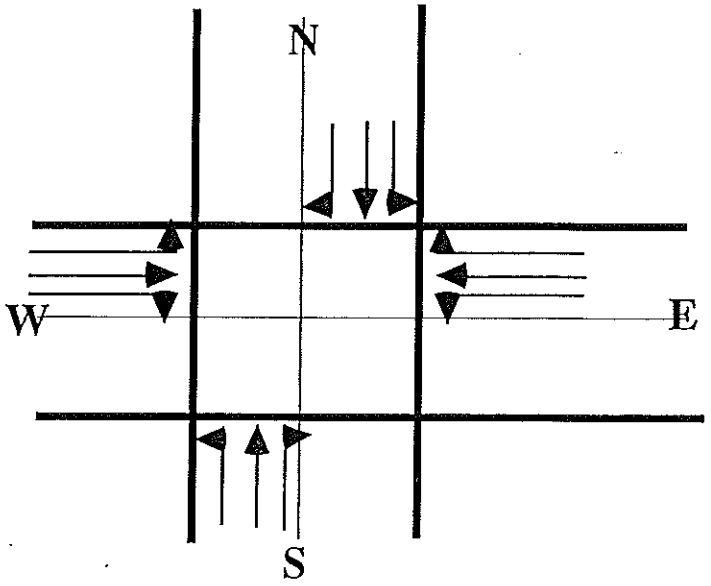
```
99

80346902212949513777098104617058130126110149689139641765068 8

43466557686937456435688527675040625802564660517371780402481 7
29089536555417949051890403879840079255169295922593080322634 7
75209689623239873322471161642996440906533187938298969649928 5
16003704476137795166849228875
```

Problem F Traffic Lights

Some years ago, New Zealand adopted a "Give Way when turning left" rule at intersections, where cars turning left into a side road have to wait for cars from the opposite direction which are turning right into that road. You have been asked to write a program to simulate traffic flow at an intersection controlled by a set of traffic lights, where this rule applies, to see if it actually does work in a situation where drivers do observe road rules. Your program need deal only with intersections at which two roads cross at right angles, as shown in the diagram below.



As can be seen from the diagram, vehicles drive on the left hand side of the road. Vehicles enter the intersection from one of four directions, which we shall call north (N), east (E), south(S) and west (W). When a vehicle reaches the intersection, it will either go left (L), straight ahead (S), or right (R). There are three separate lanes on each entrance to the intersection to allow these groups of vehicles to form separate queues.

Traffic lights regulate the flow of traffic into the intersection. The basic operation of the traffic lights is as follows. The traffic lights for one of the roads (say NS) are green, allowing vehicles from N and S roads to enter the intersection (subject to the give way rules given below). During this period the traffic lights for the other road (EW) are red, preventing any traffic from entering. After some time, the lights on the NS road change to amber (causing NS traffic to stop), then to red. At the instant that the NS lights become red, the EW lights become green allowing traffic flow in that direction. After a period, those lights turn amber, then red, and so the cycle continues. Durations of the phases of the traffic lights are a number of units (fixed in each simulation), where a unit is the time it takes a vehicle to go through the intersection. A vehicle can go through the intersection during any unit in which the appropriate light is green, or in the first unit of an amber phase, subject to the give way rules as follows:

- From any given direction, at most one car can go straight ahead, at most one car can turn left, and at most one car can turn right in any time unit.
- A car at the head of the straight ahead queue can always enter the intersection.
- A car at the head of the turning right queue can enter the intersection as long as a car from the opposite direction is not travelling straight through the intersection.
- A car at the head of the turning left queue can enter the intersection as long as a car turning right from the opposite direction is not travelling through the intersection.

You must write a program to simulate this situation and work out the total length of times vehicles must wait in a queue. Your program must perform a number of simulations; for each simulation, the times of the light phases will be given, and vehicle arrival times will be provided. Each vehicle will wish to enter the intersection in the unit after it arrives; if it cannot, it must wait in a queue until the lights change or the intersection is clear (or both). Each simulation starts with no cars waiting at the intersection. The first time unit (number 1) is the first unit of a green phase for the NS road. Your program must run each simulation until all the cars whose arrival times are given have passed through the intersection.

Input will be from the file named PROBLEMF.DAT, giving the data for a number of simulations. Each simulation begins with a line that gives the duration of the phases of the lights, containing four integers separated by one or more blanks. The first is the duration of the green phase for the NS road, the second the duration of the amber phase for the same road, and the third and the fourth the durations of the green and amber phases for the EW road.

Following these parameters are lines that specify details of vehicles arriving at the intersection. Each line contains a time, one or more blanks, an approach (N/E/S/W), one or more blanks and a specification of how the vehicle goes through the intersection (S/L/R). Times are >= 1 and <= 86400. The time in one line is not less than the time in the previous. The end of the vehicle arrivals is signified by a line with a time of -1. The end of the file is signified by a line consisting of 0 values for the four simulation parameters.

Output consists of one line for each simulation in the input, and that line consists of a single integer that is the sum of all the queuing times experienced by the vehicles whose arrivals are specified in the input, right justified in a field of width 6. The queuing time for each vehicle is the difference between its arrival time, and the time it passes through the intersection -1, so that a vehicle which passes through the intersection in the time unit after its arrival incurs a queuing time of 0 units.

EXAMPLE

INPUT

```
4 2 3 2
1 N S
1 N L
1 S R
1 E S
1 E S
1 N S
2 N L
2 N L
-1 N S
0 0 0 0
```

OUTPUT

```
    16
```

Problem G **Unrandomising Numbers**

The common random number generator used on a computer is the Congruential Random Number Generator (CRNG). It has three integer parameters, which we will label a, b and c, and depends on the fact that if these are chosen correctly then the number x, and the number

$$(ax + b) \bmod c,$$

have no apparent correlation. So if an initial seed is picked and fed into the formula, and then the output is used as the new "x", and so on, a series of apparently uncorrelated numbers will be produced.

In practice, if a, b and c are chosen carefully (eg a, b and c have no common factor, a about the same size as c, etc) then the numbers produced do satisfy randomness tests. They are not random, however, and sometimes this can cause problems. For example, if points in the plane are produced by taking as x and y coordinates successive "random" numbers, then all these points will lie on parallel straight lines. This could clearly have an effect on a simulation based on random points in the plane, although in practice problems may not occur unless only a few of the lines have points on them (which will happen if a, b and c are chosen "badly").

For this problem, you must "unravel" the parameters of CRNGs by examining their successive output. You will be given sets of 15 successive numbers produced by a CRNG - from these, you can use the definition and the "parallel line" property to possibly discover a, b and c. The a, b and c used to generate these sets all satisfy the following rules :

c is a prime less than 10000
$0 < a < c$, $0 < b < c$
a, b and c have no common factor

Input will be from a file PROBLEMG.DAT and will consist of lines of successive output from CRNGs - each line from a different generator. There are 15 numbers on each line, separated by at least one space. The file will be terminated by a line containing 15 0s.

Output will be one line for each line of input. If all the points defined by successive pairs of numbers lie on different lines in the set of parallel lines defined by the CRNG which generated the numbers, you must print the words "Insufficient data". If there is a CRNG which generates the numbers and for which at least two points lie on the same line in the set of parallel lines, you must print the a, b and c values for that CRNG, each right justified in a field of width 5. Where there is a choice of CRNG's, you must pick the smallest values, in the order; smallest c possible (remember c must be prime), for this value the smallest b which will work, for this value the smallest a which will work.

EXAMPLE
INPUT

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
401 357 151 163 389 131 409 352 446 37 351 38 292 250 393
1 0 5 17 31 35 15 4 22 6 12 19 21 11 24
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

OUTPUT

```
    1    1   17
  408  199  467
Insufficient data
```

Problem H **Bad Luck**

Friday 13th is normally regarded as a bad day, in Western cultures. Other days are bad for other cultures - for example, 24 sounds like "easy to die" in Cantonese, so Sunday 24th is the day to avoid in Hong Kong. An astrologer will be able to work out (for a small fee), for each person, what are their bad days. The object of this problem is to automate the finding of bad days in advance, so that holidays will not be marred by an unexpected bad day.

Input will be from a file PROBLEMH.DAT and will consist of years, and "bad day" descriptions; for example, 1994 Friday 13 or 2000 Sunday 24. The year will always be 1994 or greater, but not more than a billion (10^9) years in the future. The day will be one of Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and the date will be a number between 1 and 31. Each component will be separated from the others by at least one blank. The file will be terminated by a line containing a single #.

Output will be one line for each line of input, giving the numbers of the months within that year which contain the "bad day" described. If there is more than one month, the numbers must be in increasing order separated by commas. There must be no blanks in the output lines, and the numbers must have no leading zeros. If there are no months with the bad day given, the output must be the word None.

Remember that the days in each month follow the pattern:

Month number	1	2	3	4	5	6	7	8	9	10	11	12
Nbr of days	31	*	31	30	31	30	31	31	30	31	30	31

* = if (year mod 4 <> 0) or ((year mod 100 = 0) and (year mod 400 <> 0)) then 28 else 29

EXAMPLE
INPUT

```
1994 Friday 13
1995 Saturday 31
2000 Sunday 24
1000001994 Monday 1
#
```

OUTPUT

```
5
None
9,12
8
```