Problem A - Hippity Hopscotch

The game of hopscotch involves chalk, sidewalks, jumping, and picking things up. Our variant of hopscotch involves money as well. The game is played on a square grid of dimension n: each grid location is labelled (p,q) where 0 <= p < n and 0 <= q < n. Each grid location has on it a stack of between 0 and 100 pennies.

A contestant begins by standing at location (0,0). The contestant collects the money where he or she stands and then jumps either horizontally or vertically to another square. That square must be within the jumping capability of the contestant (say, k locations) and must have more pennies than those that were on the current square.

Given n, k, and the number of pennies at each grid location, compute the maximum number of pennies that the contestant can pick up before being unable to move.

Input Specification

- a line containing an integer t, the number of test cases
- a line containing two integers between 1 and 100: n and k
- n lines, each with n numbers: the first line contains the number of pennies at locations (0,0) (0,1) ... (0,n-1); the next line contains the number of pennies at locations (1,0), (1,1), ... (1,n-1), and so on.

Output Specification

- For each test case, a line containing a single integer giving the number of pennies collected

Sample Input

```
1
3 1
1 2 5
10 11 6
12 12 7
```

Output for Sample Input

```
37
```

Problem B: Doublets

A *Doublet* is a pair of words that differ in exactly one letter; for example, "booster" and "rooster" or "rooster" and "roaster" or "roaster" and "roasted".

You are given a dictionary of up to 25143 lower case words, not exceeding 16 letters each. You are then given a number of pairs of words. For each pair of words, find the shortest sequence of words that begins with the first word and ends with the second, such that each pair of adjacent words is a doublet. For example, if you were given the input pair "booster" and "roasted", a possible solution would be: ("booster", "rooster", "roaster", "roasted") provided that these words are all in the dictionary.

The Input

Input consists of the dictionary followed by a number of word pairs. The dictionary consists of a number of words, one per line, and is terminated by an empty line. The pairs of input words follow; the words of each pair occur on a line separated by a space.

The Output

For each input pair, print a set of lines starting with the first word and ending with the last. Each pair of adjacent lines must be a doublet. If there are several minimal solutions, any one will do. If there is no solution, print a line: "No solution." Leave a blank line between cases.

Sample Input

```
booster
rooster
roaster
coasted
roasted
coastal
postal

booster roasted
coastal postal
```

Output for Sample Input

```
booster
rooster
roaster
roasted

No solution.
```

Problem C: CDVII

Roman roads are famous for their longevity and sound engineering. Unfortunately, sound engineering does not come cheap, and a number of neo-Caesars have decided to recover the costs through automated tolling.

A particular toll highway, the CDVII, has a fare structure that works as follows: travel on the road costs a certain amount per km travelled, depending on the time of day when the travel begins. Cameras at every entrance and every exit capture the license numbers of all cars entering and leaving. Every calendar month, a bill is sent to the registered owner for each km travelled (at a rate determined by the time of day), plus one dollar per trip, plus a two dollar account charge. Your job is to prepare the bill for one month, given a set of license plate photos.

The Input

The first line of input contains a single integer, the number of cases to follow. Each case has two parts: the fare structure, and the license photos. The fare structure consists of a line with 24 non-negative integers denoting the toll (cents/km) from 00:00 - 00:59, the toll from 01:00 - 00:59, and so on for each hour in the day. The next line contains a single integer, the number of photo records to follow. Each photo record consists of the license number of the vehicle (up to 20 alphanumeric characters), the time and date (mm:dd:hh:mm), the word "enter" or "exit", and the location of the entrance or exit (in km from one end of the highway). All dates will be within a single month. Each "enter" record is paired with the chronologically next record for the same vehicle provided it is an "exit" record. "enter" records that are not paired with an "exit" record are ignored, as are "exit" records not paired with an "enter" record. You may assume that no two records for the same vehicle have the same time. Times are recorded using a 24-hour clock. There are not more than 1000 photo records.

The Output

For each case, print a line for each vehicle indicating the license number, and the total bill, in alphabetical order by license number.

Sample Input

```
1
10 10 10 10 10 10 20 20 20 15 15 15 15 15 15 15 20 30 20 15 15 10 10 10
4
ABCD123 01:01:06:01 enter 17
765DEF 01:01:07:00 exit 95
```

```
ABCD123 01:01:08:03 exit 95
765DEF 01:01:05:59 enter 17
```

Output for Sample Input

```
765DEF $10.80
ABCD123 $18.60
```

Problem D - Billiard

In a billiard table with horizontal side **a** inches and vertical side **b** inches, a ball is launched from the middle of the table. After **s** > 0 seconds the ball returns to the point from which it was launched, after having made **m** bounces off the vertical sides and **n** bounces off the horizontal sides of the table. Find the launching angle **A** (measured from the horizontal), which will be between 0 and 90 degrees inclusive, and the initial velocity of the ball.

Assume that the collisions with a side are elastic (no energy loss), and thus the velocity component of the ball parallel to each side remains unchanged. Also, assume the ball has a radius of zero. Remember that, unlike pool tables, billiard tables have no pockets.

Input

Input consists of a sequence of lines, each containing five nonnegative integers separated by whitespace. The five numbers are: **a**, **b**, **s**, **m**, and **n**, respectively. All numbers are positive integers not greater than 10000.

Input is terminated by a line containing five zeroes.

Output

For each input line except the last, output a line containing two real numbers (accurate to two decimal places) separated by a single space. The first number is the measure of the angle **A** in degrees and the second is the velocity of the ball measured in inches per second, according to the description above.

Sample Input

```
100 100 1 1 1
200 100 5 3 4
201 132 48 1900 156
0 0 0 0 0
```

Sample Output

```
45.00 141.42
33.69 144.22
3.09 7967.81
```

Problem E - Prime Distance

The branch of mathematics called number theory is about properties of numbers. One of the areas that has captured the interest of number theoreticians for thousands of years is the question of primality. A prime number is a number that is has no proper factors (it is only evenly divisible by 1 and itself). The first prime numbers are 2,3,5,7 but they quickly become less frequent. One of the interesting questions is how dense they are in various ranges. Adjacent primes are two numbers that are both primes, but there are no other prime numbers between the adjacent primes. For example, 2,3 are the only adjacent primes that are also adjacent numbers.

Your program is given 2 numbers: L and U (1<=L<U<=2,147,483,647), and you are to find the two adjacent primes C1 and C2 (L<=C1<C2<=U) that are closest (i.e. C2-C1 is the minimum). If there are other pairs that are the same distance apart, use the first pair. You are also to find the two adjacent primes D1 and D2 (L<=D1<D2<=U) where D1 and D2 are as distant from each other as possible (again choosing the first pair if there is a tie).

Input

Each line of input will contain two positive integers, L and U, with L < U. The difference between L and U will not exceed 1,000,000.

Output

For each L and U, the output will either be the statement that there are no adjacent primes (because there are less than two primes between the two given numbers) or a line giving the two pairs of adjacent primes.

Sample Input

```
2 17
14 17
```

Output for Sample Input

```
2,3 are closest, 7,11 are most distant.
There are no adjacent primes.
```

Problem F: Subway

You have just moved from a quiet Waterloo neighbourhood to a big, noisy city. Instead of getting to ride your bike to school every day, you now get to walk and take the subway. Because you don't want to be late for class, you want to know how long it will take you to get to school.

You walk at a speed of 10 km/h. The subway travels at 40 km/h. Assume that you are lucky, and whenever you arrive at a subway station, a train is there that you can board immediately. You may get on and off the subway any number of times, and you may switch between different subway lines if you wish. All subway lines go in both directions.

The first line of input contains a single integer, the number of test cases to follow. The first line of each case consists of the x,y coordinates of your home and your school. The next line contains a single integer, the number of subway lines to follow. Each subway line consists of the non-negative integer x,y coordinates of each stop on the line, in order. You may assume the subway runs in a straight line between adjacent stops, and the coordinates represent an integral number of metres. Each line has at least two stops. The end of each subway line is followed by the dummy coordinate pair -1,-1. In total there are at most 200 subway stops in the city.

For each case, output a line containing the number of minutes it will take you to get to school, rounded to the nearest minute, taking the fastest route.

Sample Input

```
1
0 0 10000 1000
2
0 200 5000 200 7000 200 -1 -1
2000 600 5000 600 10000 600 -1 -1
```

Sample Output

```
21
```

Problem G: Ferry Loading

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Two lanes of cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

The cars waiting to board the ferry form a single queue, and the operator directs each car in turn to drive onto the port (left) or starboard (right) lane of the ferry so as to balance the load. Each car in the queue has a different length, which the operator estimates by inspecting the queue. Based on this inspection, the operator decides which side of the ferry each car should board, and boards as many cars as possible from the queue, subject to the length limit of the ferry. Your job is to write a program that will tell the operator which car to load on which side so as to maximize the number of cars loaded.

The Input

The first line of input contains a single integer, the number of test cases to follow. The first line of each case contains a single integer between 1 and 100: the length of the ferry (in metres). For each car in the queue there is an additional line of input specifying the length of the car (in cm, an integer between 100 and 3000 inclusive). The final line of each case contains the integer 0. The cars must be loaded in order, subject to the constraint that the total length of cars on either side does not exceed the length of the ferry. Subject to this constraint as many cars should be loaded as possible, starting with the first car in the queue and loading cars in order until no more can be loaded.

The Output

For each case, output a line containing the number of cars that can be loaded onto the ferry. For each car that can be loaded onto the ferry, in the order the cars appear in the input, output a line containing "port" if the car is to be directed to the port side and "starboard" if the car is to be directed to the starboard side. If several arrangements of the cars meet the criteria above, any one will do.

Sample Input

```
1
50
2500
3000
1000
1000
1500
700
800
0
```

Possible Output for Sample Input

```
6
port
starboard
starboard
starboard
port
port
```

Problem H - Leaps Tall Buildings (in a single bound)

It's a bird! It's a plane! It's coming right at us!

Although it sometimes seems like it, Superman can't fly (without a plane). Instead, he makes super-human leaps, especially over tall buildings. Since he never knows when he will need to catch a criminal, he can't register flight paths. To avoid hitting planes, he tries to keep his jumps as low to the ground as he can. Given a city-scape as input, find the angle and velocity of Superman's jump that minimizes his maximum altitude.

Recall that gravity provides an acceleration of 9.8 m/s^2 downwards and the formula for Superman's vertical distance from his starting location is $d(t)=v\ t\ +\ 0.5\ a\ t^2$ where v is his initial velocity, a is his acceleration and t is time in seconds since the start of the leap.

Input:

Input consists of a sequence of city-scapes, each of the form

n
0 d_1
h_2 d_2
:
$h_{(n-1)}$ $d_{(n-1)}$
0 d_n

Superman starts at ground level and leaps $d_1+...+d_n$ metres, landing at ground level and clearing all of the buildings at heights h_2 to $h_{(n-1)}$, each with the given widths. n will be at most 100.

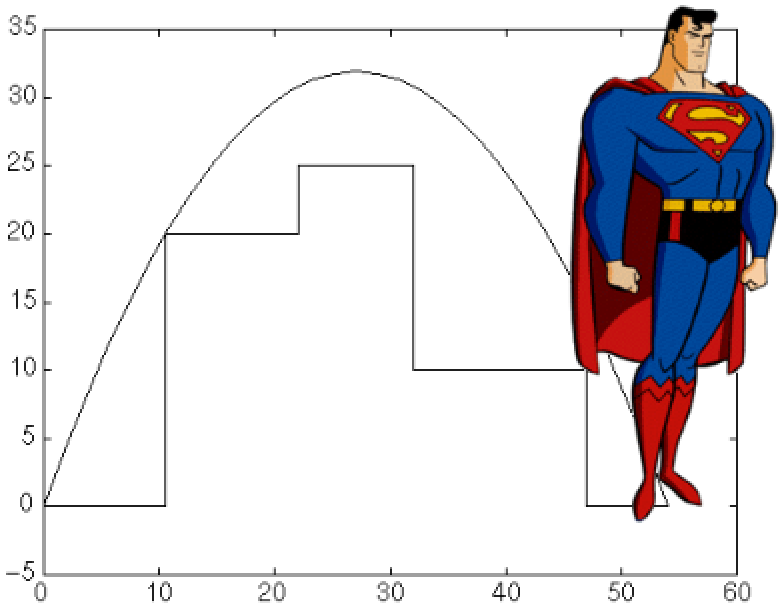
Output:

Output is the angle and initial velocity that minimizes the height that Superman attains, both appearing on the same line. The values should be given to two decimal places and be accurate within 0.01 degrees or m/s, as appropriate.

Sample Input:

```
3
0 5
10 5
0 5
5
0 10.5
20 11.5
25 10
10 15
0 7
```

Diagram for Second City-scape



(Not to scale.)

Sample Output:

```
71.57 15.65
67.07 27.16
```

Problem I: Interpreter

A certain computer has 10 registers and 1000 words of RAM. Each register or RAM location holds a 3-digit integer between 0 and 999. Instructions are encoded as 3-digit integers and stored in RAM. The encodings are as follows:

- 100 means *halt*
- 2dn means *set register d to n (between 0 and 9)*
- 3dn means *add n to register d*
- 4dn means *multiply register d by n*
- 5ds means *set register d to the value of register s*
- 6ds means *add the value of register s to register d*
- 7ds means *multiply register d by the value of register s*
- 8da means *set register d to the value in RAM whose address is in register a*
- 9sa means *set the value in RAM whose address is in register a to the value of register s*
- 0ds means *goto the location in register d unless register s contains 0*

All registers initially contain 000. The initial content of the RAM is read from standard input. The first instruction to be executed is at RAM address 0. All results are reduced modulo 1000.

The first line of input contains a single integer, the number of test cases to follow. The first line of each test case contains a single integer, the number of initial values to follow. The next n lines contain 3-digit unsigned integers, representing the contents of consecutive RAM locations starting at 0. Unspecified RAM locations are initialized to 000.

For each case, the output a line containing a single integer: the number of instructions executed up to and including the *halt* instruction. You may assume that all programs halt.

Sample Input

```
1
15
299
492
495
399
492
495
399
283
279
689
078
100
000
000
000
```

Output for Sample Input

```
16
```

Problem J: Gopher II

The gopher family, having averted the canine threat, must face a new predator.

The are *n* gophers and *m* gopher holes, each at distinct (x, y) coordinates. A hawk arrives and if a gopher does not reach a hole in *s* seconds it is vulnerable to being eaten. A hole can save at most one gopher. All the gophers run at the same velocity *v*. The gopher family needs an escape strategy that minimizes the number of vulnerable gophers.

The input contains several cases. The first line of each case contains four positive integers less than 100: *n*, *m*, *s*, and *v*. The next *n* lines give the coordinates of the gophers; the following *m* lines give the coordinates of the gopher holes. All distances are in metres; all times are in seconds; all velocities are in metres per second.

Output consists of a single line for each case, giving the number of vulnerable gophers.

Sample Input

```
2 2 5 10
1.0 1.0
2.0 2.0
100.0 100.0
20.0 20.0
```

Output for Sample Input

```
1
```