

UW Madison's
2005 ACM-ICPC Team Test #2
October 29, 1:00-6:00pm, 1350 CS

Overview:

This test consists of eight problems, which will be referred to by the following names (respective of order):

radio, toll, movers, edit, phobos, sum, willy, strings

Please note that some problems also include a problem number (or letter). Please ignore these labelings and use the names above.

As this is a team competition, you are allowed one computer for each team. Also, you may only use 25 pages of printed documentation in addition to the online STL and Java documentation.

Input/Output:

Your programs should take input from standard in (i.e. the keyboard) and output to standard out (i.e. the terminal). As is standard practice for the ICPC, you may assume that the input strictly follows the description in the problems. If a problem description does not indicate how the input will be terminated, you may assume it will be with the EOF character. It is your responsibility to ensure that your output **precisely** matches that described in the problems, or else risk your program being rejected with a "Presentation Error".

Problem Submission:

To submit a program, send e-mail to mwa+icpc@cs.wisc.edu and attach your source code. The subject line should contain only the problem name, **prob**, that you are submitting, and the attached source code should be named **prob**.{c|cpp|java}. For example, if you are using C++ and submitting the problem "radio", the file should be named "radio.cpp". You will receive the results of your submission as soon as possible via e-mail. You may submit from any e-mail account of a team member, but please try to be consistent and use only one throughout the contest.

Clarifications:

As in the ICPC, you may submit clarification requests as well. They should be sent to mwa+icpc@cs.wisc.edu, with a subject of "Clarification-**prob**", where **prob** is the name of the problem you wish to be clarified. Replace **prob** with "general" if there is an issue with the contest as a whole. Accepted clarification requests will be answered to all those taking the test via e-mail. Experience of previous years learns that most clarification requests are rejected, and receive a simple response such as "Read the problem description".

Printing:

You may print to the printer at any time during the test.

GOOD LUCK!

Not an official ACM page

[[Problem D](#)] | [1992 ACM finals problem set](#) | [My ACM problem archive](#) | [my home page](#)

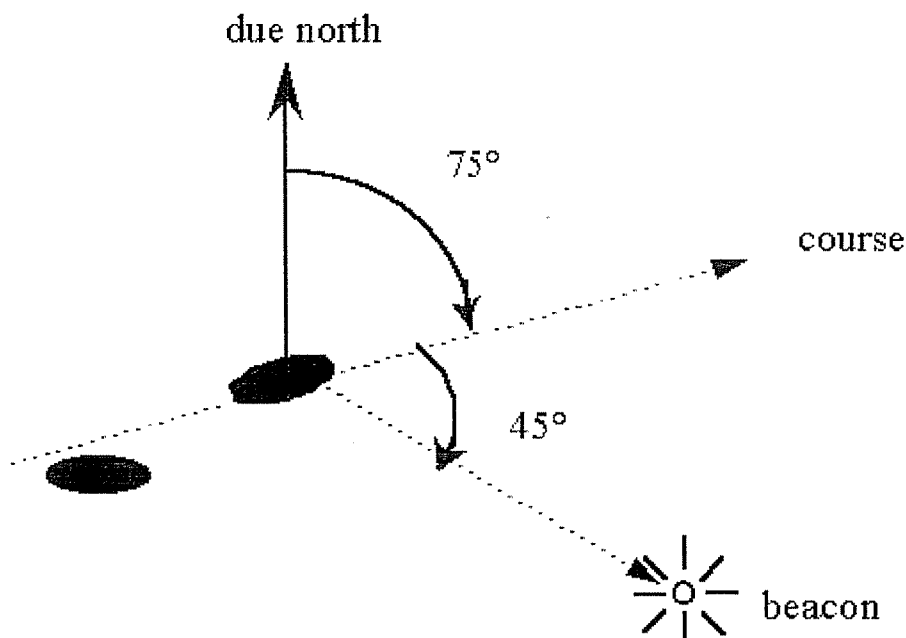
1992 ACM Scholastic Programming Contest Finals

sponsored by AT&T EasyLink Services

Problem C Radio Direction Finder

A boat with a directional antenna can determine its present position with the help of readings from local beacons. Each beacon is located at a known position and emits a unique signal. When a boat detects a signal, it rotates its antenna until the signal is at maximal strength. This gives a relative bearing to the position of the beacon. Given a previous beacon reading (the time, the relative bearing, and the position of the beacon), a new beacon reading is usually sufficient to determine the boat's present position. You are to write a program to determine, when possible, boat positions from pairs of beacon readings.

For this problem, the positions of beacons and boats are relative to a rectangular coordinate system. The positive x -axis points east; the positive y -axis points north. The course is the direction of travel of the boat and is measured in degrees clockwise from north. That is, north is 0° , east is 90° , south is 180° , and west is 270° . The relative bearing of a beacon is given in degrees clockwise relative to the course of the boat. A boat's antenna cannot indicate on which side the beacon is located. A relative bearing of 90 means that the beacon is toward 90° or 270° .



The boat's course is 75° . The beacon has a relative bearing of 45° from the boat's course.

Input

The first line of input is an integer specifying the number of beacons (at most 30). Following that is a

line for each beacon. Each of those lines begins with the beacon's name (a string of 20 or fewer alphabetic characters), the x-coordinate of its position, and the y-coordinate of its position. These fields are single-space separated.

Coming after the lines of beacon information is an integer specifying a number of boat scenarios to follow. A boat scenario consists of three lines, one for velocity and two for beacon readings.

Data on input line	Meaning of data
course speed	the boat's course, the speed at which it is traveling
time#1 name#1 angle#1	time of first beacon reading, name of first beacon, relative bearing of first beacon
time#2 name#2 angle#2	time of second reading, name of second beacon, relative bearing of second beacon

All times are given in minutes since midnight measured over a single 24-hour period. The speed is the distance (in units matching those on the rectangular coordinate system) over time. The second line of a scenario gives the first beacon reading as the time of the reading (an integer), the name of the beacon, and the angle of the reading as measured from the boat's course. These 3 fields have single space separators. The third line gives the second beacon reading. The time for that reading will always be at least as large as the time for the first reading.

Output

For each scenario, your program should print the scenario number (Scenario 1; Scenario 2, etc.) and a message indicating the position (rounded to 2 decimal places) of the boat as of the time of the second beacon reading. If it is impossible to determine the position of the boat, the message should say "Position cannot be determined." Sample input and corresponding correct output are shown below.

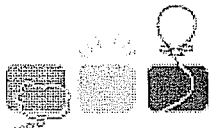
Sample Input

```
4
First 2.0 4.0
Second 6.0 2.0
Third 6.0 7.0
Fourth 10.0 5.0
2
0.0 1.0
1 First 270.0
2 Fourth 90.0
116.5651 2.2361
4 Third 126.8699
5 First 319.3987
```

Sample Output

```
Scenario 1: Position cannot be determined
Scenario 2: Position is (6.00, 5.00)
```

This page maintained by [Ed Karrels](#).
Last updated September 20, 1999

	<h2 style="margin: 0;">2730 – Toll</h2> <h3 style="margin: 0;">World Finals – Beverly Hills – 2002/2003</h3>		
<u>PDF</u>	<u>PostScript</u>	<u>Submit</u>	<u>Ranking</u>

Sindbad the Sailor sold 66 silver spoons to the Sultan of Samarkand. The selling was quite easy; but delivering was complicated. The items were transported over land, passing through several towns and villages. Each town and village demanded an entry toll. There were no tolls for leaving. The toll for entering a *village* was simply one item. The toll for entering a *town* was one piece per 20 items carried. For example, to enter a town carrying 70 items, you had to pay 4 items as toll. The towns and villages were situated strategically between rocks, swamps and rivers, so you could not avoid them.

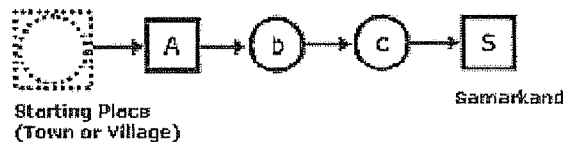


Figure 1: To reach Samarkand with 66 spoons, traveling through a town followed by two villages, you must start with 76 spoons.

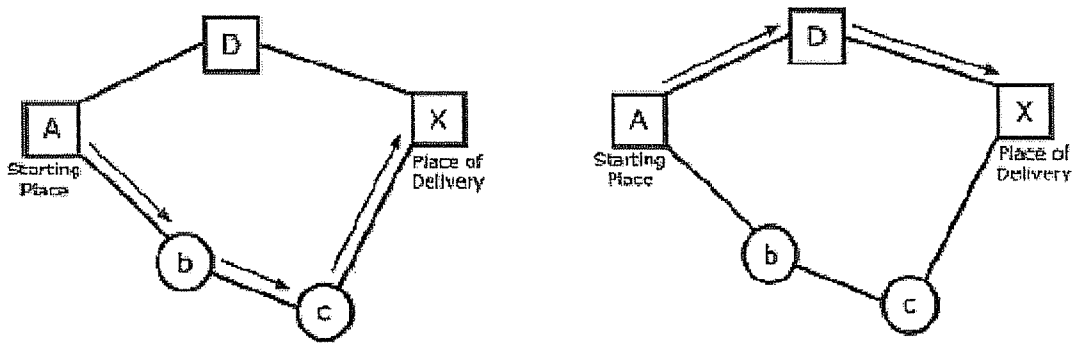


Figure 2: The best route to reach X with 39 spoons, starting from A, is $A \rightarrow b \rightarrow c \rightarrow X$, shown with arrows in the figure on the left. The best route to reach X with 10 spoons is $A \rightarrow D \rightarrow X$, shown in the figure on the right. The figures display towns as squares and villages as circles.

Predicting the tolls charged in each village or town is quite simple, but finding the best route (the cheapest route) is a real challenge. The best route depends upon the number of items carried. For numbers up to 20, villages and towns charge the same. For large numbers of items, it makes sense to avoid towns and travel through more villages, as illustrated in Figure 2.

You must write a program to solve Sindbad’s problem. Given the number of items to be delivered to a certain town or village and a road map, your program must determine the total number of items required at the beginning of the journey that uses a cheapest route.

Input

The input consists of several test cases. Each test case consists of two parts: the roadmap followed by the delivery details.

The first line of the roadmap contains an integer n , which is the number of roads in the map ($0 \leq n$).

Each of the next n lines contains exactly two letters representing the two endpoints of a road. A capital letter represents a town; a lower case letter represents a village. Roads can be traveled in either direction.

Following the roadmap is a single line for the delivery details. This line consists of three things: an integer p ($0 < p \leq 1000$) for the number of items that must be delivered, a letter for the starting place, and a letter for the place of delivery. The roadmap is always such that the items can be delivered.

The last test case is followed by a line containing the number -1.

Output

The output consists of a single line for each test case. Each line displays the case number and the number of items required at the beginning of the journey. Follow the output format in the example given below.

Sample Input

```
1
a Z
19 a Z
5
A D
D X
A b
b c
c X
39 A X
-1
```

Sample Output

```
Case 1: 20
Case 2: 44
```

Beverly Hills 2002–2003

Problem 3: The Tree Movers

Given two binary search trees, A and B, with nodes identified by (that is, having keys equal to) positive, non-zero integers, and the use of commands "delete K" and "add K" (defined below), what is the smallest number of commands that can be used to transform tree A into tree B?

Recall that in a binary search tree, the keys of all nodes in the left subtree of a node with key K must be less than K. Similarly, the keys of all nodes in the right subtree of a node with key K must be greater than K. There are no duplicate nodes.

The "delete K" command will delete the tree (or subtree) with its root at the node with the key K. Deleting the root of the entire tree leaves an empty tree. The "add K" command will add a new node identified by the integer K. This node will naturally be a leaf node.

Since we seek to transform tree A into tree B, it follows that commands will be applied only to tree A; tree B is "read only."

It is easy to see that it should never require more than $N + 1$ commands to achieve the transformation of A into B, since deletion of the root node of tree A followed by the addition of one node for each of the N nodes in B (in the proper order) will achieve the desired goal. Equally easy to determine is the minimum number of commands required: if A and B are identical, then zero commands are required.

Input

There will be multiple input cases. For each case, the input contains the description of tree A followed by the description of tree B. Each tree description consists of an integer N that specifies the number of nodes in the tree, following by the keys of the N nodes in an order such that N "add" commands would create the tree. The last case is followed by the integer -1. No node will have a key larger than 10^9 , and N will be no larger than 100.

Output

For each case, display a single line containing the input case number (1, 2, ...) and the number of commands required to transform tree A into tree B, formatted as shown in the examples below.

Sample Input

```
4 5 2 7 4 6 5 3 7 1 4 9
0 0
1 100 0
0 1 100
3 100 49 37 2 200 152
-1
```

Expected Output

```
Case 1: 5 commands.
Case 2: 0 commands.
Case 3: 1 command.
Case 4: 1 command.
Case 5: 3 commands.
```


Problem C: Edit Step Ladders

An *edit step* is a transformation from one word x to another word y such that x and y are words in the dictionary, and x can be transformed to y by adding, deleting, or changing one letter. So the transformation from *dig* to *dog* or from *dog* to *do* are both edit steps. An *edit step ladder* is a lexicographically ordered sequence of words w_1, w_2, \dots, w_n such that the transformation from w_i to w_{i+1} is an edit step for all i from 1 to $n-1$.

For a given dictionary, you are to compute the length of the longest edit step ladder.

Input

The input to your program consists of the dictionary – a set of lower case words in lexicographic order – one per line. No word exceeds 16 letters and there are no more than 25000 words in the dictionary.

Output

The output consists of a single integer, the number of words in the longest edit step ladder.

Sample Input

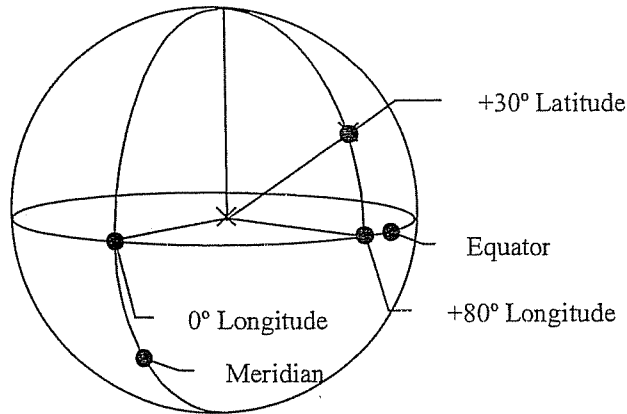
```
cat
dig
dog
fig
fin
fine
fog
log
wine
```

Sample Output

```
5
```


Problem 5: Communication Planning for Phobos

Life has been found on Phobos, one of the satellites of Mars! Unfortunately, the life forms there aren't quite as advanced as those on Earth, and they don't have modern communications (at least by Earth standards). The Advanced Communication Management Company (ACM) has decided to build a central office and connect the Phobosians' homes for communication (telephone, television, Internet, and so forth). They naturally want to minimize their capital outlay in this effort, and they need to decide how to lay fiber optic cable (essentially on the surface) so the smallest amount is used. Since ACM uses digital broadband technology, it is only necessary that there be a cable path that connects every subscriber and the central office. That is, there does not necessarily need to be a separate cable from the central office to each subscriber's home.



We know the precise location of each Phobosian's home and the planned ACM central office on the surface. These are given using longitude and latitude. Longitude is measured from an arbitrary meridian on the surface of Phobos, and has values in the range -180 degrees to $+180$ degrees. Latitude is measured from the equator, and has values in the range -90 degrees to $+90$ degrees. For planning purposes we assume Phobos is perfectly spherical, exactly 16.7 miles in diameter. The figure to the left illustrates one possible location ($+80^\circ$ longitude, $+30^\circ$ latitude).

INPUT

There will be one or more sets of input data. Each set will contain, in order, an integer N no larger than 100, but at least 2, followed by N pairs of real numbers, each pair giving the unique longitude and latitude, in degrees, of a Phobosian's home or the central office. A single integer zero will follow the last data set.

OUTPUT

For each input data set print a single line containing the data set number (1, 2, ...) and the number of miles of cable required to connect all the Phobosian's homes and the central office; show two fractional digits in the distance.

SAMPLE INPUT

```

3
0 0    0 90    0 -90

3
0 0    0 90    90 0

3
0 0    90 0    45 0

6
-10 10    -10 -10    0 0    90 0    80 20    100 -10

0
    
```

EXPECTED OUTPUT

```

Case 1: 26.23 miles
Case 2: 26.23 miles
Case 3: 13.12 miles
Case 4: 21.16 miles
    
```


Maximum Sum

Background

A problem that is simple to solve in one dimension is often much more difficult to solve in more than one dimension. Consider satisfying a boolean expression in conjunctive normal form in which each conjunct consists of exactly 3 disjuncts. This problem (3-SAT) is NP-complete. The problem 2-SAT is solved quite efficiently, however. In contrast, some problems belong to the same complexity class regardless of the dimensionality of the problem.

The Problem

Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the *maximal sub-rectangle*. A sub-rectangle is any contiguous sub-array of size 1×1 or greater located within the whole array. As an example, the maximal sub-rectangle of the array:

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

is in the lower-left-hand corner:

9	2
-4	1
-1	8

and has the sum of 15.

Input and Output

The input consists of an $N \times N$ array of integers. The input begins with a single positive integer N on a line by itself indicating the size of the square two dimensional array. This is followed by N^2 integers separated by white-space (newlines and spaces). These N^2 integers make up the array in row-major order (i.e., all numbers on the first row, left-to-right, then all numbers on the second row, left-to-right, etc.). N may be as large as 100. The numbers in the array will be in the range $[-127, 127]$.

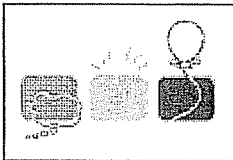
The output is the sum of the maximal sub-rectangle.

Sample Input

```
4
0 -2 -7 0 9 2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Sample Output

15



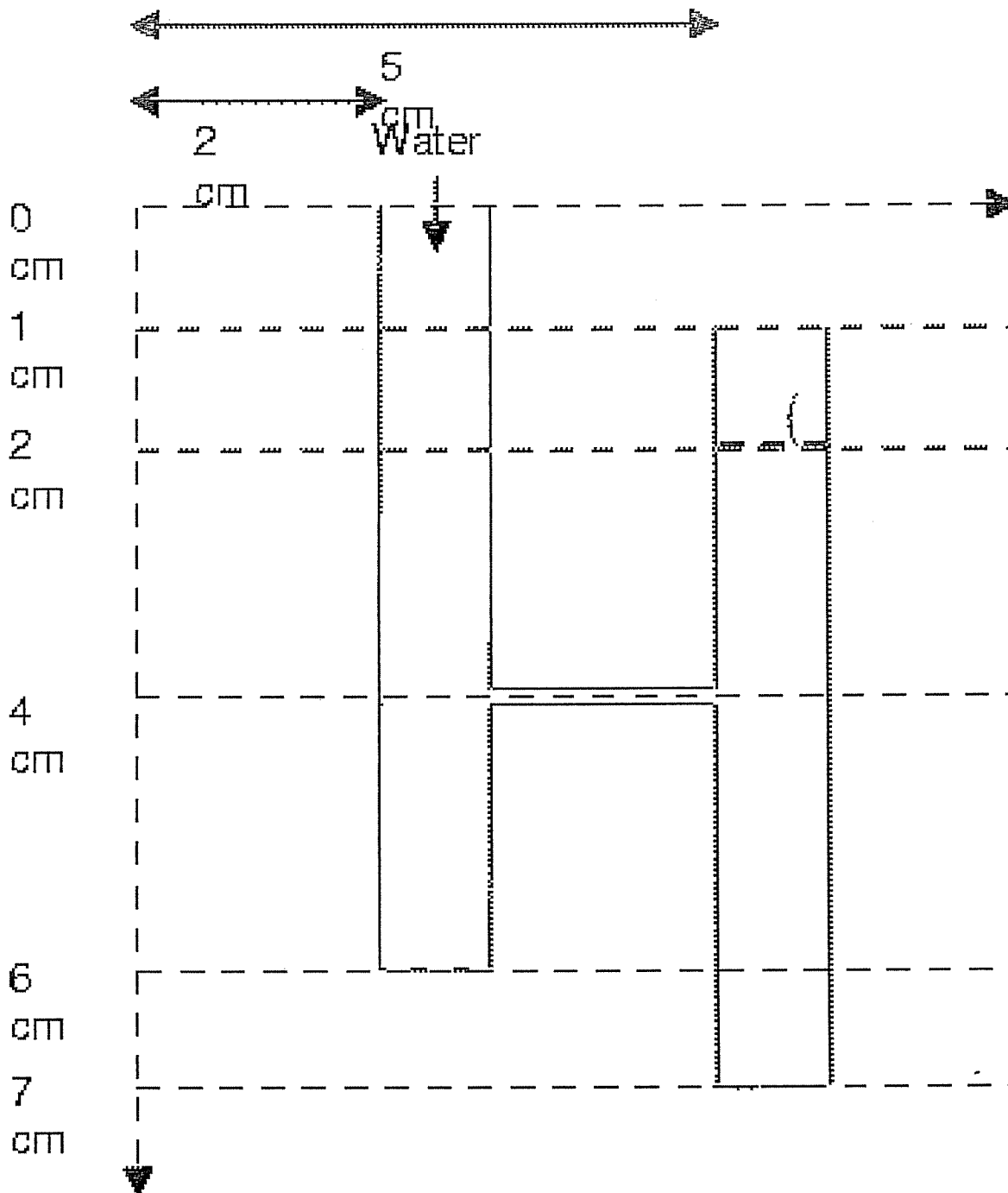
2343 – The Willy Memorial Program

Asia – Tehran – 2001/2002

Willy the spider used to live in the chemistry laboratory of Dr. Petro. He used to wander about the lab pipes and sometimes inside empty ones. One night while he was in a pipe, he fell asleep. The next morning, Dr. Petro came to the lab. He didn't notice Willy while opening the valve to fill the pipes with hot water. Meanwhile, Stanley the gray mouse got what was going to happen. No time to lose! Stan ran hard to reach the valve before Willy gets drawn, but... Alas! He couldn't make it!

Poor Willy was boiled in hot water, but his memory is still in our hearts. Though Stan tried his best, we want to write a program, in the memory of Willy, to compute the time Stan had, to rescue Willy, assuming he started to run just when the doctor opened the valve.

To simplify the problem, assume the pipes are all vertical cylinders with diameter 1 cm. Every pipe is open from the top and closed at the bottom. Some of the pipes are connected through special horizontal pipes named *links*. The links have very high flow capacity, but are so tiny that at any given time, the volume of water inside them is negligible. The water enters from top of one of the pipes with a constant rate of 0.25π cm³/sec and begins to fill the pipe from the bottom until the water reaches a link through which it flows horizontally and begins to fill the connected pipe. From elementary physics we know if two pipes are connected and the surface of the water is above the connecting link, the level of water in both pipes remains the same when we try to fill one of them. In this case the water fills each pipe with a rate equal to half of the rate of incoming water. As an example, consider the following configuration:



First, the lower 2 centimeters of the left pipe is filled with water at full rate, then, the lower 3 centimeters of the right pipe is filled, and after that, the upper part of the two pipes are filled in parallel at half rate. The input to your program is a configuration of pipes and links, and a target level in one of the pipes (the heavy dotted line in the above figure). The program should report how long it takes for the level of water to reach the target level. For the above configuration, the output is 9 seconds.

It is assumed that the water falls very rapidly, such that the time required for the water to fall can be neglected. The target level is always assumed to be a bit higher than the specified level for it. As an example, if we set the target point to level 4 in the left pipe in the figure above, the elapsed time for water to reach that target is assumed to be 5 (not 2). Also note that if the water reaches to the top of a pipe (say in level x), it won't pour out outside the pipe until empty spaces in connected pipes below level x are filled (if can be filled, i.e. the level of water reaches the connecting links). (Note that there may be some links at level x , to which water is entered). After all such spaces are filled; the water level would not go up further.

Input

To describe positions, we assume the coordinates are expressed as (x, y) and the origin lies in the top-left of all pipes and links. (Note that y coordinates are increased downwards). All coordinates are integer numbers between 0 and 100, inclusive.

The first line of the input file contains a single integer t ($1 \leq t \leq 10$), the number of test cases, followed by the input data for each test case. The first line of each test case is p ($1 \leq p \leq 20$), the number of pipes, followed by p lines, each describing a pipe. Each pipe description line consists of three numbers. The first two are (x, y) coordinates of the upper-left corner of the pipe and the third number is the height of the pipe (at least 1 cm, and at most 20 cm). Note that diameter of each pipe is 1 cm.

After input data describing the pipes, there is a line containing a single integer l , which is the number of links ($0 \leq l \leq 50$). After it, there are l lines describing links. Each link description contains 3 integers. The first two (x, y) coordinates of the left end-point of the link and the third is the length of the link (at least 1 cm and at most 20 cm). It is assumed that the width of the link is zero.

The last line for each test case contains two numbers. The first is the number of target pipe (starting from one, with the order appeared in test data). The second line is the desired y for the level of water in the target pipe (note that the specified level may be out of the pipe at all).

You can assume the following about the input:

- The water enters into the first pipe.
- No link crosses a pipe.
- No two links have the same y coordinates.
- No two pipes have the same upper-left x coordinates.
- Both endpoints of each link are connected to pipes.

Output

The output file should contain exactly t lines with no blank lines in between, each corresponding to one test case. Each output line should contain the time required for the water to reach the target level in the target pipe (an integer number). If in a specific test case, the water never reaches the target level, the line should contain 'No Solution' string in it.

Sample Input

```
1
2
2 0 6
5 1 6
1
3 4 2
2 2
```

Sample Output

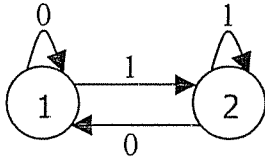
```
9
```

Tehran 2001–2002

Problem 8: Acceptable Strings

A deterministic finite automaton has a finite set of states, with directed edges leading from one state to another. Each edge is labeled with a symbol. In this problem, we are only concerned about automata (the plural of automaton) that use the binary digits 0 and 1 as symbols. Each edge is thus labeled with 0 or 1. One state is identified as the start state, and one or more states are identified as final states.

A finite automaton is usually represented by a graph. For example, consider the finite automaton represented by the graph shown below; the states are shown as circles, and are named 1 and 2 for ease of identification. In this automaton, state 1 is the start state, and state 2 is the final state.



Each automaton in this problem accepts or rejects a string as follows. Beginning in the start state, for each symbol (0 or 1) in the input string (working from left to right in sequence), the automaton follows the one edge labeled with the input symbol from the current state to the next state. After making the transition associated with the last symbol in the input string, if the automaton is in a final state, then the input is accepted. Otherwise (that is, if the automaton is not in a final state), the input is

rejected.

For the string 0101 and the automaton shown above, we start in state 1 (the start state). Since the first input symbol is 0, the edge labeled 0 from state 1 back to state 1 is followed, leaving us in state 1. The next input symbol, 1, causes a transition to state 2. The next symbol, 0, moves us back to state 1. The last input symbol, 1, causes the last transition, from state 1 to state 2. Since state 2 is a final state, the automaton accepts the string 0101. Note that the string 010 would have been rejected, since the automaton would have been in state 1 (which is not a final state) at the end of the input. This automaton happens to accept all binary strings that end with 1.

In this problem you will be given one or more automata and an integer N . For each of these, you are to find the number of binary strings having each length less than or equal to N that are accepted by the automaton. For example, for $N = 3$ with the automaton above, the output would specify 0 strings of length 0 (since state 1 is not a final state), 1 string of length 1 (1), 2 strings of length 2 (01 and 11), and 4 strings of length 3 (001, 011, 101, and 111).

Input

There will be multiple input cases. For each case the input begins with three integers N , S , and F . N (no larger than 10) specifies the maximum length of the strings that are sought. S (no larger than 100) specifies the number of states in the automaton. F (no larger than 10) specifies the number of final states. Following these three integers are S pairs of integers. Each pair specifies the labels on the edges from the states, in order, starting with state 1. The first integer in each pair specifies the state to which the edge labeled 0 connects; the second integer specifies the state to which the edge labeled 1 connects. Finally, the last F integers identify the final states. State 1 will always be the start state. The input for the last case is followed by three zeroes.

Output

For each case, display the case number (they are numbered sequentially, and start with 1). Then display the number of strings of each length (from 0 to N) accepted by the automaton, using a separate line for each length. The output must be identical in format to that shown in the examples below.

Sample Input

```
3 2 1
1 1
1 1
2
3 2 1
1 2
1 2
2
10 7 1
2 2
3 3
4 4
5 5
6 6
7 7
7 7
6
0 0 0
```

Expected Output

```
Case 1:
Length = 0, 0 strings accepted.
Length = 1, 0 strings accepted.
Length = 2, 0 strings accepted.
Length = 3, 0 strings accepted.
Case 2:
Length = 0, 0 strings accepted.
Length = 1, 1 string accepted.
Length = 2, 2 strings accepted.
Length = 3, 4 strings accepted.
Case 3:
Length = 0, 0 strings accepted.
Length = 1, 0 strings accepted.
Length = 2, 0 strings accepted.
Length = 3, 0 strings accepted.
Length = 4, 0 strings accepted.
Length = 5, 32 strings accepted.
Length = 6, 0 strings accepted.
Length = 7, 0 strings accepted.
Length = 8, 0 strings accepted.
Length = 9, 0 strings accepted.
Length = 10, 0 strings accepted.
```