The 2000 24th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem A
Abbott's Revenge
Input File: abbott.in

The 1999 World Finals Contest included a problem based on a "dice maze." At the time the problem was written, the judges were unable to discover the original source of the dice maze concept. Shortly after the contest, however, Mr. Robert Abbott, the creator of numerous mazes and an author on the subject, contacted the contest judges and identified himself as the originator of dice mazes. We regret that we did not credit Mr. Abbott for his original concept in last year's problem statement. But we are happy to report that Mr. Abbott has offered his expertise to this year's contest with his original and unpublished "walk-through arrow mazes."

As are most mazes, a walk-through arrow maze is traversed by moving from intersection to intersection until the goal intersection is reached. As each intersection is approached from a given direction, a sign near the entry to the intersection indicates in which directions the intersection can be exited. These directions are always left, forward or right, or any combination of these.

Figure 1 illustrates a walk-through arrow maze. The intersections are identified as "(row, column)" pairs, with the upper left being (1,1). The "Entrance" intersection for Figure 1 is (3,1), and the "Goal" intersection is (3,3). You begin the maze by moving north from (3,1). As you walk from (3,1) to (2,1), the sign at (2,1) indicates that as you approach (2,1) from the south (traveling north) you may continue to go only forward. Continuing forward takes you toward (1,1). The sign at (1,1) as you approach from the south indicates that you may exit (1,1) only by making a right. This turns you to the east now walking from (1,1) toward (1,2). So far there have been no choices to be made. This is also the case as you continue to move from (1,2) to (2,2) to (2,3) to (1,3). Now, however, as you move west from (1,3) toward (1,2), you have the option of continuing straight or turning left. Continuing straight would take you on toward (1,1), while turning left would take you south to (2,2). The actual (unique) solution to this maze is the following sequence of intersections: (3,1) (2,1) (1,1) (1,2) (2,2) (2,3) (1,3) (1,2) (1,1) (2,1) (2,2) (1,2) (1,3) (2,3) (3,3).

You must write a program to solve valid walk-through arrow mazes. Solving a maze means (if possible) finding a route through the maze that leaves the Entrance in the prescribed direction, and ends in the Goal. This route should not be longer than necessary, of course.

Input

The input file will consist of one or more arrow mazes. The first line of each maze description contains the name of the maze, which is an alphanumeric string of no more than 20 characters. The next line contains, in the following order, the starting row, the starting column, the starting direction, the goal row, and finally the goal column. All are delimited by a single space. The maximum dimensions of a maze for this problem are 9 by 9, so all row and column numbers are single digits from 1 to 9. The starting direction is one of the characters N, S, E or W, indicating north, south, east and west, respectively.

All remaining input lines for a maze have this format: two integers, one or more groups of characters, and a sentinel asterisk, again all delimited by a single space. The integers represent the row and column, respectively, of a maze intersection. Each character group represents a sign at that intersection. The first character in the group is N, S, E or W to indicate in what direction of travel the sign would be seen. For example, S indicates that this is the sign that is seen when travelling south. (This is the sign posted at the north entrance to the intersection.) Following this first direction character are one to three arrow characters. These can be L, F or R indicating left, forward, and right, respectively.

The list of intersections is concluded by a line containing a single zero in the first column. The next line of the input starts the next maze, and so on. The end of input is the word END on a single line by itself.

Output

For each maze, the output file should contain a line with the name of the maze, followed by one or more lines with either a solution to the maze or the phrase “No Solution Possible”. Maze names should start in column 1, and all other lines should start in column 3, i.e., indented two spaces. Solutions should be output as a list of intersections in the format “(R,C)” in the order they are visited from the start to the goal, should be delimited by a single space, and all but the last line of the solution should contain exactly 10 intersections.

The first maze in the following sample input is the maze in Figure 1.

Sample Input

```
SAMPLE
3 1 N 3 3
1 1 WL NR *
1 2 WLF NR ER *
1 3 NL ER *
2 1 SL WR NF *
2 2 SL WF ELF *
2 3 SFR EL *
0
NOSOLUTION
3 1 N 3 2
1 1 WL NR *
1 2 NL ER *
2 1 SL WR NFR *
2 2 SR EL *
0
END
```

Output for the Sample Input

```
SAMPLE
  (3,1) (2,1) (1,1) (1,2) (2,2) (2,3) (1,3) (1,2) (1,1) (2,1)
  (2,2) (1,2) (1,3) (2,3) (3,3)
NOSOLUTION
  No Solution Possible
```

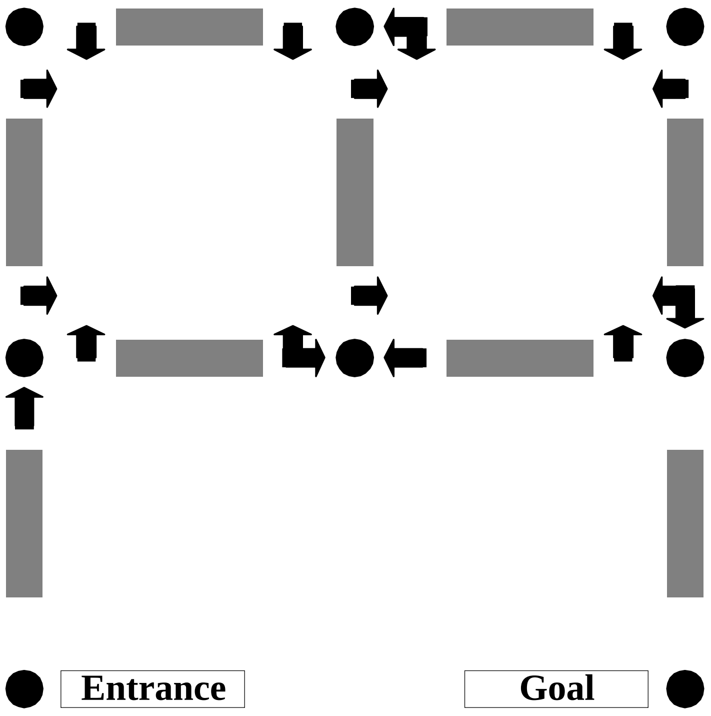


Figure 1: An Example Walk-Through Arrow Maze

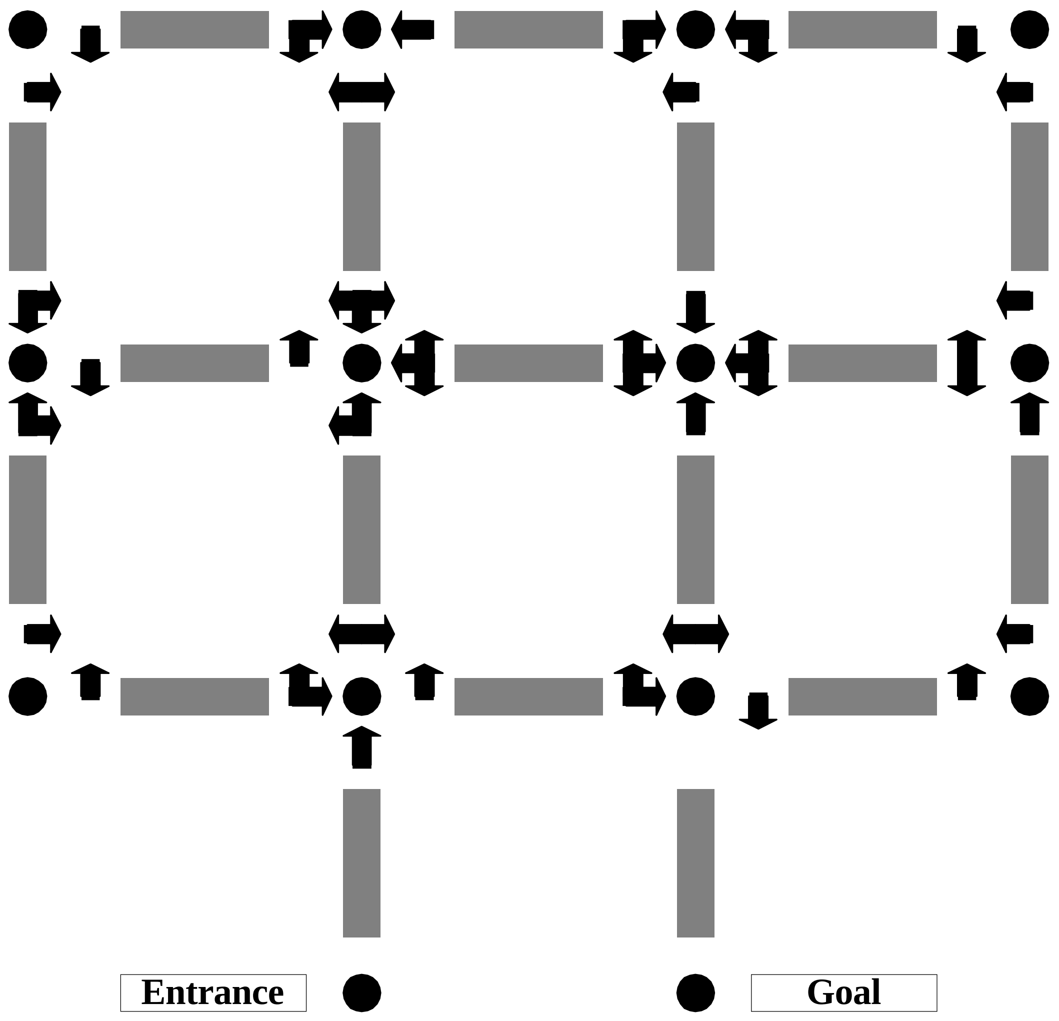


Figure 2: Robert Abbott's Atlanta Maze

Robert Abbott's walk-through arrow mazes are actually intended for large-scale construction, not paper. Although his mazes are unpublished, some of them have actually been built. One of these is on display at an Atlanta museum. Others have been constructed by the American Maze Company over the past two summers. As their name suggests these mazes are intended to be walked through.

For the adventurous, Figure 2 a graphic of Robert Abbott's Atlanta maze. Solving it is quite difficult, even when you have an overview of the entire maze. Imagine trying to solve this by actually walking through the maze and only seeing one sign at a time! Robert Abbott himself indicated that the maze is too complex and most people give up before finishing. Among the people that did not give up was Donald Knuth: it took him about thirty minutes to solve the maze.

The 2000 24th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem B

According to Bartjens

Input File: bartjens.in

The wide dissemination of calculators and computers has its disadvantages. Even students in technical disciplines tend to exhibit a surprising lack of calculating ability. Accustomed to the use of calculators and computers, many of them are unable to make calculations like 7* 8 mentally or like 13 * 17 using pencil and paper. We all know, but who cares?

Professor Bartjens[1] cares. Professor Bartjens is a bit old fashioned. He decided to give his students some training in calculating without electronic equipment by creating a collection of calculation problems, (like 2100 – 100 =…). To simplify grading the problems, he constructed them so that *almost* all of them had 2000 as an answer. Not all of them, of course. His students would be smart enough to recognize the pattern, and fill in 2000 everywhere without further thinking.

Unfortunately Professor Bartjens' printer driver turned out to be even more old-fashioned than the professor himself, and it could not interface with his new printer. Inspecting the printed problems, he soon recognized the pattern: *none of the operations was transmitted to the printer.* A problem like:

```
2100-100=
```

was printed as:

```
2100100=
```

Fortunately, all the digits and the equal sign were still printed.

To make this bad situation much worse, Professor Bartjens' source file had disappeared. So Professor Bartjens has another problem: what were his original problems? Given the fact that the answer (most likely) should be 2000, the line `2100100=` could have been any one of the lines:

```
2100-100=
2*100*10+0=
2*100*10-0=
2*10*0100=
2*-100*-10+0=
```

Professor Bartjens does remember a few things about how he wrote the problems:

- He is sure that whenever he wrote down a number (other than 0), it would not start with a zero. So `2*10*0100=` could **not** have been one of his problems.
- He also knows he never wrote the number zero as anything but `0`. So he would **not** have a problem like `2*1000+000=`.
- He used only binary operators, not the unary minus or plus, so `2*-100*-10+0=` was **not** an option either.
- He used the operators `+`, `-` and `*` only, avoiding the operator `/` (after all, they were first year students).
- He knew all problems followed the usual precedence and associativity rules.

You are to help Professor Bartjens recover his problem set by writing a program that when given a row of digits, insert one or more of the operators `+`, `-` and `*` in such a way that the value of the resulting expression equals 2000.

[1] Willem Bartjens (1569-1638) was the author of *Cijferinge*, a much used Dutch textbook on arithmetic. The phrase "…according to Bartjens" (uttered following a calculation) made his name immortal.

Input

The input consists of one or more test cases. Each test case is a single line containing *n* digits ('0'…'9'), 1 ≤ *n* ≤ 9, followed by an equal sign. There will not be any blanks embedded in the input, but there may be some after the equal sign.

The last test case is followed by a line containing only the equal sign. This line should not be processed.

Output

For each test case, print the word `Problem`, then the number of the case, then all possible ways of inserting operators in the row of digits such that the resulting expression has the value 2000, subject to Professor Bartjens' memory of how he wrote the problems. Use the format shown below. If there is more than one possible problem, they may be written in any order, but no problem may appear more than once in the list. Each possible problem should be on a new line, indented 2 spaces. If there is no solution the answer IMPOSSIBLE should be printed, indented 2 spaces.

Sample Input

```
2100100=
77=
=
```

Output for the Sample Input

```
Problem 1
  2100-100=
  2*100*10+0=
  2*100*10-0=
Problem 2
  IMPOSSIBLE
```

Problem C
Cutting Chains
Input File: chains.in

What a find! Anna Locke has just bought several links of chain some of which may be connected. They are made from zorkium, a material that was frequently used to manufacture jewelry in the last century, but is not used for that purpose anymore. It has its very own shine, incomparable to gold or silver, and impossible to describe to anyone who has not seen it first hand.

Anna wants the pieces joined into a single end-to-end strand of chain. She takes the links to a jeweler who tells her that the cost of joining them depends on the number of chain links that must be opened and closed. In order to minimize the cost, she carefully calculates the minimum number of links that have to be opened to rejoin all the links into a single sequence. This turns out to be more difficult than she at first thought. You must solve this problem for her.

Input

The input consists of descriptions of sets of chain links, one set per line. Each set is a list of integers delimited by one or more spaces. Every description starts with an integer *n*, which is the number of chain links in the set, where $1 \le n \le 15$. We will label the links 1, 2,…, *n*. The integers following *n* describe which links are connected to each other. Every connection is specified by a pair of integers *i*,*j* where $1 \le i,j \le n$ and $i \ne j$, indicating that chain links *i* and *j* are connected, i.e., one passes through the other. The description for each set is terminated by the pair `-1 -1`, which should not be processed.

The input is terminated by a description starting with *n* = 0. This description should not be processed and will not contain data for connected links.

Output

For each set of chain links in the input, output a single line which reads

```
Set N: Minimum links to open is M
```

where `N` is the set number and `M` is the minimal number of links that have to be opened and closed such that all links can be joined into one single chain.

Sample Input	Output for the Sample Input
`5  1 2  2 3  4 5  -1 -1`	`Set 1: Minimum links to open is 1`
`7  1 2  2 3  3 1  4 5  5 6  6 7  7 4 -1 -1`	`Set 2: Minimum links to open is 2`
`4  1 2  1 3  1 4  -1 -1`	`Set 3: Minimum links to open is 1`
`3  1 2  2 3  3 1  -1 -1`	`Set 4: Minimum links to open is 1`
`3  1 2  2 1  -1 -1`	`Set 5: Minimum links to open is 1`
`0`	

The 2000 24th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem D

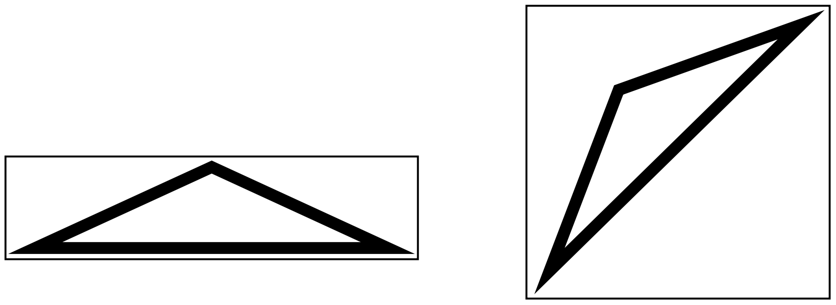
Gifts Large and Small

Input File: gifts.in

WrapIt.com specializes in wrapping gifts. Started several years ago as a service offered to local department stores and malls, today WrapIt serves customers world-wide and boasts that it can package anything from half-carat diamonds to whole apartment blocks.

WrapIt has found that some customers prefer their gifts to be wrapped in the smallest possible packages, whereas others prefer large packages that make their gifts seem larger than they really are. The company needs a program that computes the smallest and largest rectangular package into which a gift can be “tightly” wrapped. Since this is a difficult problem, the company will initially settle for a two-dimensional version of the program.

Each gift is approximated as a simple polygon, and all packages are represented by rectangles. A gift is said to “fit tightly” in a package if the gift touches all four sides of the package. The figure below shows how a triangular gift might fit tightly in two packages of different sizes. For each gift, your program must compute the areas of the smallest and largest packages into which the gift can fit tightly.



Input

The input contains several gift descriptions. Each description begins with a line containing an integer n ($3 \le n \le 100$), which is the number of vertices in the polygon that represents the gift. The following n lines contain pairs of integers that represent the coordinates of the polygon vertices, in clockwise order. Each polygon will have a non-zero area and will not intersect itself.

The input is terminated by a line containing the integer 0.

Output

For each gift, first print the number of the gift. Then on separate lines, print the minimum and maximum areas of the packages into which the gift fits tightly, using the format in the sample output. Print a blank line after each test case. The computed areas should be exact to three digits to the right of the decimal point.

Sample Input	Output for the Sample Input

```
3
-3  5
7   9
17  5
4
10 10
10 20
20 20
20 10
0
```
	```
	Gift 1
	Minimum area = 80.000
	Maximum area = 200.000

	Gift 2
	Minimum area = 100.000
	Maximum area = 200.000
	```

The 2000 24th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

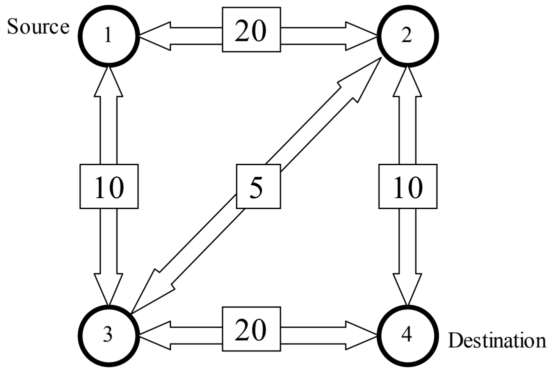
Problem E

Internet Bandwidth

Input File: internet.in

On the Internet, machines (nodes) are richly interconnected, and many paths may exist between a given pair of nodes. The total message-carrying capacity (bandwidth) between two given nodes is the maximal amount of data per unit time that can be transmitted from one node to the other. Using a technique called packet switching, this data can be transmitted along several paths at the same time.

For example, the following figure shows a network with four nodes (shown as circles), with a total of five connections among them. Every connection is labeled with a bandwidth that represents its data-carrying capacity per unit time.



In our example, the bandwidth between node 1 and node 4 is 25, which might be thought of as the sum of the bandwidths 10 along the path 1-2-4, 10 along the path 1-3-4, and 5 along the path 1-2-3-4. No other combination of paths between nodes 1 and 4 provides a larger bandwidth.

You must write a program that computes the bandwidth between two given nodes in a network, given the individual bandwidths of all the connections in the network. In this problem, assume that the bandwidth of a connection is always the same in both directions (which is not necessarily true in the real world).

Input

The input file contains descriptions of several networks. Every description starts with a line containing a single integer *n* ($2 \leq n \leq 100$), which is the number of nodes in the network. The nodes are numbered from 1 to *n*. The next line contains three numbers *s*, *t*, and *c*. The numbers *s* and *t* are the source and destination nodes, and the number *c* is the total number of connections in the network. Following this are *c* lines describing the connections. Each of these lines contains three integers: the first two are the numbers of the connected nodes, and the third number is the bandwidth of the connection. The bandwidth is a non-negative number not greater than 1000.

There might be more than one connection between a pair of nodes, but a node cannot be connected to itself. All connections are bi-directional, i.e. data can be transmitted in both directions along a connection, but the sum of the amount of data transmitted in both directions must be less than the bandwidth.

A line containing the number 0 follows the last network description, and terminates the input.

Output

For each network description, first print the number of the network. Then print the total bandwidth between the source node *s* and the destination node *t*, following the format of the sample output. Print a blank line after each test case.

Sample Input	Output for the Sample Input

4
1 4 5
1 2 20
1 3 10
2 3 5
2 4 10
3 4 20
0
	Network 1
	The bandwidth is 25.

The 2000 24th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

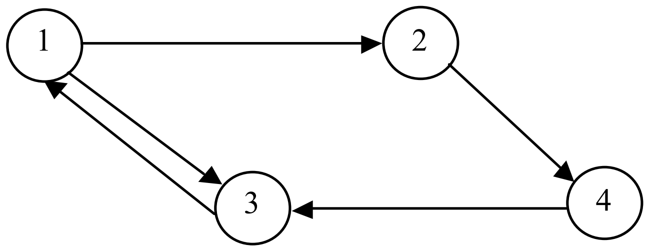
Problem F

Page Hopping

Input File: page.in

It was recently reported that, on the average, only 19 clicks are necessary to move from any page on the World Wide Web to any other page. That is, if the pages on the web are viewed as nodes in a graph, then the average path length between arbitrary pairs of nodes in the graph is 19.

Given a graph in which all nodes can be reached from any starting point, your job is to find the average shortest path length between arbitrary pairs of nodes. For example, consider the following graph. Note that links are shown as directed edges, since a link from page *a* to page *b* does not imply a link from page *b* to page *a*.



The length of the shortest path from node 1 to nodes 2, 3, and 4 is 1,1, and 2 respectively. From node 2 to nodes 1, 3 and 4, the shortest paths have lengths of 3, 2, and 1. From node 3 to nodes 1, 2, and 4, the shortest paths have lengths of 1, 2, and 3. Finally, from node 4 to nodes 1, 2, and 3 the shortest paths have lengths of 2, 3, and 1. The sum of these path lengths is 1 + 1 + 2 + 3 + 2 + 1 + 1 + 2 + 3 + 2 + 3 + 1 = 22. Since there are 12 possible pairs of nodes to consider, we obtain an average path length of 22/12, or 1.833 (accurate to three fractional digits).

Input

The input data will contain multiple test cases. Each test case will consist of an arbitrary number of pairs of integers, *a* and *b*, each representing a link from a page numbered *a* to a page numbered *b*. Page numbers will always be in the range 1 to 100. The input for each test case will be terminated with a pair of zeroes, which are not to be treated as page numbers. An additional pair of zeroes will follow the last test case, effectively representing a test case with no links, which is not to be processed. The graph will not include self-referential links (that is, there will be no direct link from a node to itself), and at least one path will exist from each node in the graph to every other node in the graph.

Output

For each test case, determine the average shortest path length between every pair of nodes, accurate to three fractional digits. Display this length and the test case identifier (they're numbered sequentially starting with 1) in a form similar to that shown in the sample output below.

Sample Input

```
1 2   2 4   1 3   3 1   4 3   0 0
1 2   1 4   4 2   2 7   7 1   0 0
0 0
```

Output for the Sample Input

```
Case 1: average length between pages = 1.833 clicks
Case 2: average length between pages = 1.750 clicks
```

The 2000 24th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem G
Queue and A
Input File: queue.in

The customer support group of Contest.com receives and responds to requests for technical support via e-mail. Requests may begin arriving when the office opens at 8:00 a.m. and all requests must be serviced by the end of the day.

As requests are received, they are classified according to a predetermined list of topics. Each member of the support staff has responsibility for one or more of these topics and each topic has one or more support personnel assigned to it. Because staff members have different levels of expertise, each staff member has a prioritized list of topics that he or she can handle. Staff personnel are not permitted to handle requests outside their specified areas.

As staff members become available, they select from the pool of waiting requests according to their priority list of topics. All requests arriving at time *t* are available for allocation at time *t*. If two staff members are simultaneously available, scheduling preference is given to the one whose most recent job was scheduled earliest. If there is still a tie, scheduling preference is given to the person whose id number appears earlier in the input list of staff people. At the opening of business, all personnel are available to handle requests.

You have been asked to perform a preliminary analysis of the technical support environment based on a number of different scenarios. For each scenario, information will be given about the mix of requests and the division of labor among the staff. For each topic, you will given the average number of requests per day for that topic, the average elapsed time before the first of these requests is received, the average time between requests for this topic, and the average time needed to service the request. All times are given in minutes. You will also be given a list of support personnel and, for each one, a list of the topics for which he or she has responsibility. (Since data are based on estimates, factors such as coffee breaks, lunch, computer failures, etc., can be ignored.)

Input

Input consists of a number of scenarios. Each scenario begins with the number of request topics, a positive integer no larger than 20. This is followed by a description of each topic. Each description consists of five integer values: a unique topic identifier, the number of requests for that topic, the elapsed time before the first request for that topic is received, the time needed to service a request, and the time between successive requests. All but the third of these values are positive integers; the elapsed time until the first request could be zero. Following this, the number of personnel is given. This will be a positive integer not to exceed 5. Finally, a description of each person is given in the form of three or more positive integer values: a unique identifying number for the person, the number of topics covered by this person, and a list of the topic identifiers arranged from highest priority to lowest priority for that person. A zero follows the last scenario.

Output

For each scenario, the output consists of the scenario number followed by the statement, "All requests are serviced within *m* minutes," where *m* is the number of minutes from the start of the business day until the last request is serviced.

Sample Input	Output for the Sample Input

3
128 20 0 5 10
134 25 5 6 7
153 30 10 4 5
4
10 2 128 134
11 1 134
12 2 128 153
13 1 153
0
	Scenario 1: All requests are serviced within 195 minutes.

The 2000 24th Annual **acm** International Collegiate
Programming Contest World Finals
sponsored by IBM

Problem H
Stopper Stumper
Input File: stopper.in

Stephen Stepper's Supply Store sells stoppers—rubber corks for sealing jars, bottles, and other containers that have round openings. A stopper is shaped like two concentric cylinders, each of height 1.5 centimeters, glued together. Figure 1 shows two stoppers of different sizes.

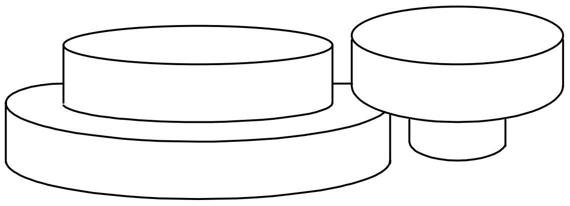


Figure 1. Two stoppers.

When Stephen packages an order to be shipped by mail, he tries to use space efficiently to conserve packing material. Since stoppers are the smallest items in his store, he packs them last, and finds that he must often pack several stoppers into small triangular gaps at the top of the shipping carton. Stoppers must inserted in one of the two orientations shown in Figure 1. The triangular spaces are only 3 cm deep, so stoppers cannot be placed on top of one another; however, the large cylinder of one stopper is permitted to overlap the large cylinder of another inverted stopper as shown in Figures 1 and 2. Your job is to help Stephen decide what collections of stoppers will fit into a triangular space.

For instance, suppose a triangular space with side lengths 8, 7, and 10 were available, and we had to fit three stoppers in it with inside/outside diameters of 2cm/3cm, 1.5cm/3cm, and 1cm/3cm. One way to pack them is as shown in Figure 2. (The dotted circle indicates that the smaller cylinder of one of the stoppers is underneath the larger one.) The only packing Stephen will consider has the larger cylinder of each stopper touching two sides of the triangle, with no two larger cylinders touching the same pair of sides.

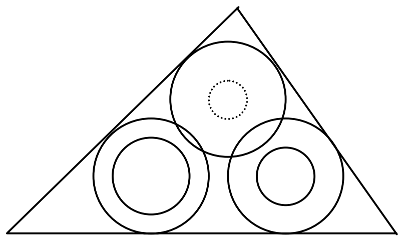


Figure 2: Three stoppers packed inside a triangle.

Input

The input consists of a sequence of triangle specifications and descriptions of three stoppers for each triangle. Each triangular space is specified by three positive integers representing the lengths of the three sides; only valid triangles will appear in the input. A pair of positive real numbers represents each stopper. The first number in the pair represents the diameter of the smaller cylinder, and the second represents the diameter of the larger cylinder. The final line of the input file contains zeros for all the data values.

Output

For each triangle, print a line identifying its sequence number in the input data and a line indicating whether or not the stoppers can be packed into the triangular space. Separate the output for each triangle with a blank line. Do not print anything for the final line of zeros in the input. Imitate the sample output as closely as possible.

Sample Input

```
 6  6  6   0.5  1.0    0.3  2.0    0.4  1.0
10 10 10   2.0  3.0    1.0  2.0    1.5  3.5
20  6 20   3.0  4.5    0.5  1.0    4.0  5.0
 8  7 10   2.0  3.0    1.5  3.0    1.0  3.0
 8  7 10   2.0  3.0    2.5  3.0    2.0  3.0
 0  0  0   0.0  0.0    0.0  0.0    0.0  0.0
```

Output for the Sample Input

```
Triangle number 1:
All three stoppers will fit in the triangular space

Triangle number 2:
All three stoppers will fit in the triangular space

Triangle number 3:
Stoppers will not fit in the triangular space

Triangle number 4:
All three stoppers will fit in the triangular space

Triangle number 5:
Stoppers will not fit in the triangular space
```