Problem A
Eyeball Benders
Input File: eyeball.in

"Eyeball benders" are a popular kind of puzzle in which the reader must identify a common object based on a close-up view of a part of that object. For instance, an image that looks like a regular array of colored cones might be a view of an open box of new crayons. Figure 1 shows an example where the puzzle is on the left and the solution is on the right.

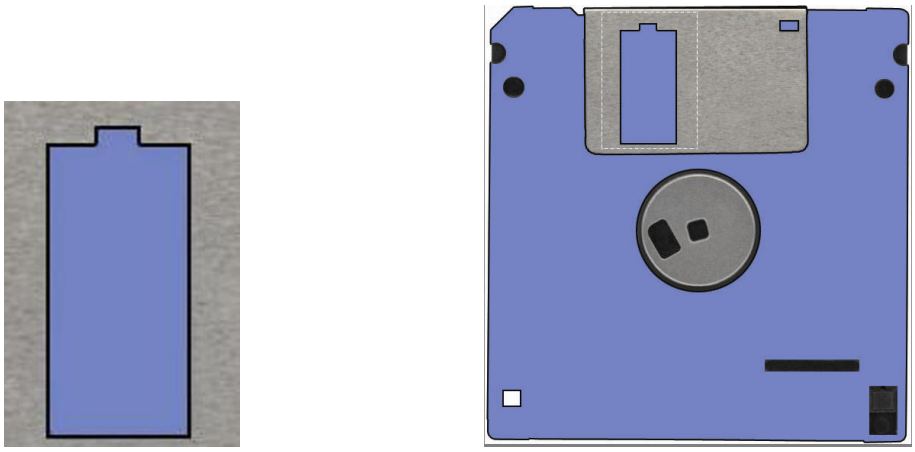


Figure 1. A sample eyeball-bender puzzle and solution image (a floppy disk).

You must verify solutions to a simplified version of the "eyeball bender" puzzle. You will be given a number of pairs of images, each one a collection of line segments. All line segments will be either horizontal or vertical, and they include their endpoints. Figure 2 shows an example.

You must determine whether the images form a valid pair in which the first image is a magnified view of some portion of the second image. Lines are assumed to have zero thickness in both images. At least one endpoint in the puzzle image of a valid pair must be an endpoint of a line segment in the solution image.

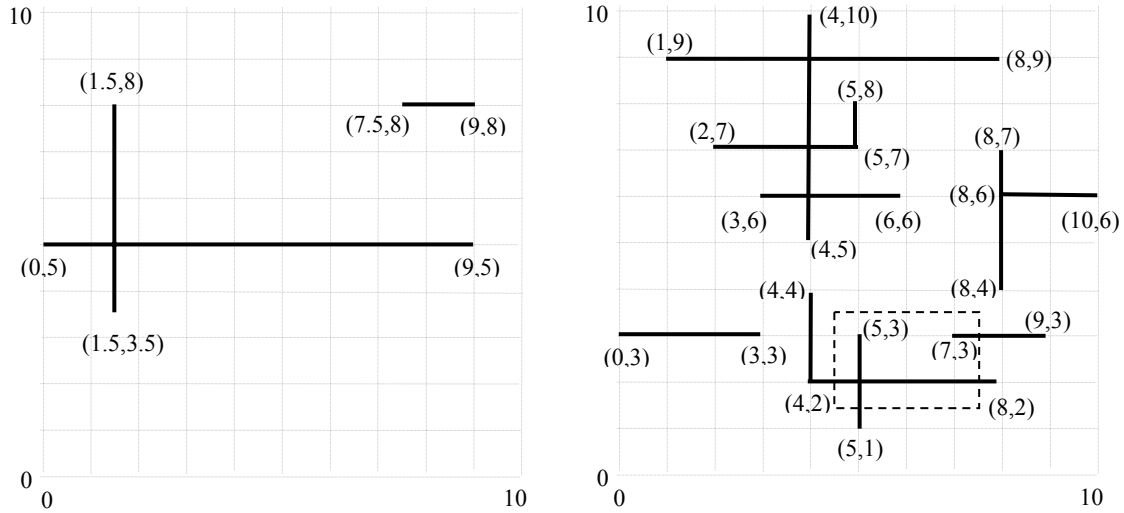


Figure 2. The left image is the portion of the right image inside the dotted rectangle, magnified 3 times.

Coordinates describe *relative* positions and scale within a single image. The coordinates in one image do not necessarily use the same origin or scale as those in the other image. The magnification of the puzzle image relative to the solution image is required to be greater than or equal to 1. For Figure 2, your program should determine that this is a valid puzzle/solution image pair.

Input

The input consists of multiple cases. The input for each case begins with two positive integers M and N, ($1 \le M,N \le 50$). M is the number of line segments in the puzzle image. N is the number of line segments in the proposed solution image. The following lines contain $M + N$ pairs of points. The first M pairs of points are the endpoints of the line segments in the puzzle image; the remaining N pairs are the endpoints of the line segments in the proposed solution image. The x and y coordinates for each pair satisfy $-100 \le x, y \le 100$ and are given to at most three decimal places of precision. All input values are separated by white space (blanks or new line characters).

No pair of distinct points in a given image will be closer than .005 to another (relative to the scale of the image) and all segments will have length at least .005. No two horizontal segments overlap and no two vertical segments overlap. However, horizontal segments may intersect vertical segments either internally or at segment endpoints.

The input data for the last case is followed by a line consisting of the integers 0 0.

Output

For each input case, display the case number (1, 2, …) followed by the words "`valid puzzle`" if the proposed solution image matches a closed rectangular sub-region of the puzzle image (including at least one endpoint), magnified by a factor of one or greater, and possibly translated by some amount. Line segments that are not included in the puzzle image will be at least 0.005 distant from the rectangle.

If the match condition fails to hold, print "`impossible`". Follow the format of the sample output.

Sample Input	Output for the Sample Input
3 12 9 8 7.5 8 1.5 8 1.5 3.5 0 5 9 5 4 2 8 2 5 7 2 7 10 6 8 6 8 7 8 4 1 9 8 9 9 3 7 3 4 10 4 5 4 2 4 4 5 8 5 7 3 6 6 6 0 3 3 3 5 1 5 3 4 12 -50 -5 50 -5 0 10 0 -10 50 5 -50 5 -50 0 50 0 4 2 8 2 5 7 2 7 10 6 8 6 8 7 8 4 1 9 8 9 9 3 7 3 4 10 4 5 4 2 4 4 5 8 5 7 3 6 6 6 0 3 3 3 5 1 5 3 0 0	Case 1: valid puzzle Case 2: impossible

Problem B
Simplified GSM Network
Input File: gsm.in

Mobile phones have changed our lifestyle dramatically in the last decade. Mobile phones have a variety of protocols to connect with one another. One of the most popular networks for mobile phones is the GSM (Global System for Mobile Communication) network.

In a typical GSM network, a mobile phone connects with the nearest BTS (Base Transceiver Station). A BSC (Base Station Center) controls several BTSs. Several BSCs are controlled by one MSC (Mobile Services Switching Center), and this MSC maintains a connection with several other MSCs, a PSTN (Public Switched Telecom Network) and an ISDN (Integrated Services Digital Network).

This problem uses a simplified model of the conventional GSM network. Our simplified network is composed of up to fifty BTS towers. When in use, a mobile phone always connects to its nearest BTS tower. The area covered by a single BTS tower is called a cell. When an active mobile phone is in motion, as it crosses cell boundaries it must seamlessly switch from one BTS to another. Given the description of a map consisting of cities, roads and BTS towers, you must determine the minimum number of BTS switches required to go from one city to another.

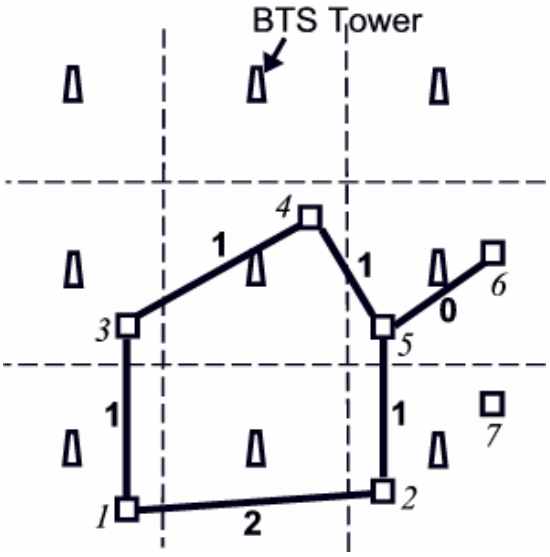


Figure: Cities here are represented by squares and BTS towers by trapezoids. Solid lines are roads. The dotted lines show 9 different cells. The minimum number of switches required to go from city 1 to city 6 is (2+1+0)=3. Note that city 7 is isolated and cannot be reached.

Each tower and each city location is to be considered as a single point in a two-dimensional Cartesian coordinate system. If there is a road between two cities, assume that the road is a straight line segment connecting these two cities. For example, in the figure, traveling on the road from city 1 to city 2 will cross two cell boundaries and thus requires two switches. Traveling from city 2 to city 5 crosses one cell boundary and traveling from city 5 to city 6 requires no switch. Traveling this route from city 1 to city 6 requires three total switches. Note than any other path from city 1 to city 6 requires more than three switches. If there is more than one possible way to get from one city to another, your program must find the optimal route.

Input

The input file contains several test cases. The first line of each test case contains four integers: B ($1 \le B \le 50$), the number of BTS towers; C ($1 \le C \le 50$), the number of cities; R ($0 \le R \le 250$), the number of roads; and Q ($1 \le Q \le 10$), the number of queries. Each of the next B lines contains two floating-point numbers x and y, the

Cartesian coordinates of a BTS tower. Each of the next C lines contains two floating-point numbers x_i, y_i that indicate the Cartesian coordinates of the ith city ($1 \le i \le C$). Each of the next R lines contains two integers m and n ($1 \le m,n \le C$), which indicate that there is a road between the mth and the nth city. Each of the next Q lines contains two integers s and d ($1 \le s,d \le C$), the source and destination cities.

No coordinate will have an absolute value greater than 1000. No two towers will be at the same location. No two cities will be at the same location, and no city will lie on a cell boundary. No road will be coincident with a cell boundary, nor contain a point lying on the boundary of three or more cells.

The input will end with a line containing four zeros.

Output

For each input set, you should produce Q+1 lines of output, as shown below. The first line should contain the number of the test case. Q lines should follow, one for each query, each containing an integer indicating the minimum number of switches required to go from city s to city d. If it is not possible to go from city s to city d, print the line "`Impossible`" instead.

Sample Input	Output for the Sample Input
9 7 6 2 5 5 15 5 25 5 5 15 15 15 25 15 5 25 15 25 25 25 8 2 22 3 8 12 18 18 22 12 28 16 28 8 1 2 1 3 2 5 3 4 4 5 5 6 1 6 1 7 0 0 0 0	Case 1: 3 Impossible

Problem C
The Traveling Judges Problem
Input File: judges.in

A group of judges must get together to judge a contest held in a particular city, and they need to figure out the cheapest way of renting cars in order to get everyone to the contest. They observed that it might be cheaper if several judges share a rental car during all or part of the trip, thus reducing the overall cost. Your task is to identify the routes the judges should take in order to minimize the total cost of their car rentals.

We will make the following assumptions:

- The cost of a rental car is proportional to the distance it is driven. There are no charges for more than one occupant in the car, fuel, insurance, or leaving the car in a city other than that in which it was rented.
- All rental cars are billed at the same rate per mile.
- A rental car can accommodate any number of passengers.
- At most one road directly connects any pair of cities. Each road is two-way and has an integer length greater than zero.
- There is at least one route from every judge's starting city to the city in which the contest is held.
- All judges whose routes to the contest take them from or through the same city travel from that city to the contest together. (A judge might arrive at a city in one car and leave that city in a different car.)

Input

The input contains several test cases. Each test case includes a route map, the destination city where the contest is being held, and the cities where the judges are initially located.

Each case appears in the input as a list of integers separated by blanks and/or ends of lines. The order and interpretation of the integers in each case is as follows:

- *NC* – the number of cities that appear in the route map; this will be no larger than 20.
- *DC* – the number of the destination city, assuming the cities are numbered 1 to *NC*.
- *NR* – the number of roads in the route map. Each road connects a distinct pair of cities.
- For each of the *NR* roads, three integers *C1*, *C2*, and *DIST*. *C1* and *C2* identify two cities connected by a road, and *DIST* gives the distance between these cities along that road.
- *NJ* – the number of judges. This number will be no larger than 10.
- *NJ* integers, one for each judge – each of these is a city number identifying the initial location of that judge.

The data for the last case will be followed by a line consisting of the integer –1.

Output

For each test case, display the case number (1, 2, …) and the shortest total distance traveled by the rental cars conveying the judges to the contest. Then display the list of routes used by the judges, each route on a separate line, in the same order as the ordering of starting cities given in the input. Each route consists of the cities that the corresponding judge must visit, listed in the order in which they are visited, starting with the judge's starting city and ending with the contest city. Any other cities along the route are listed in the order in which they are visited during the judge's travels. Separate the numbers in the route with "–", and precede each route by three spaces.

If multiple sets of routes have the same minimum distance, choose a set that requires the fewest number of cities. If several sets of cities of the same cardinality may be used, choose the set that comes lexicographically first when ordered by city number (e.g., {2, 3, 6} rather than {2, 10, 5}). If multiple routes are still available, output any set of routes that meets the requirements.

Follow the format of the sample output.

Sample Input	Output for the Sample Input
5 3 5 1 2 1 2 3 2 3 4 3 4 5 1 2 4 2 2 5 1 4 4 3 1 3 1 2 3 2 3 4 2 2 1 2 3 3 3 1 2 2 1 3 3 2 3 1 2 2 1 -1	Case 1: distance = 6 5-4-2-3 1-2-3 Case 2: distance = 5 1-3-4 2-3-4 Case 3: distance = 3 2-3 1-2-3

Problem D

cNteSahruPfefrlefe

Input File: shuffle.in

Preston Digitation is a magician who specializes in card tricks. One thing Preston cannot get just right is perfect in-shuffles. A perfect in-shuffle is one where a deck of 52 cards is divided in half and then the two halves are perfectly interleaved so that the top card of the lower half of the deck becomes the top card of the shuffled deck. If we number the cards 0 (top) to 51 (bottom), the resulting deck after a perfect in-shuffle will look like the following:

26 0 27 1 28 2 29 3 30 4 31 5 32 6 ... 51 25

Preston finds that he makes at most one mistake per shuffle. For example, cards 2 and 28 might end up interchanged, resulting in a shuffled deck that looks like this:

26 0 27 1 2 28 29 3 30 4 31 5 32 6 ... 51 25

These exchanges of two adjacent cards are the only mistakes Preston makes. After one shuffle, it is easy for him to see if and where he has made a mistake, but after several shuffles this becomes increasingly difficult. He would like you to write a program that can determine his mistakes (if any).

Input

Input will consist of multiple problem instances. The first line will be a single integer indicating the number of problem instances. Each problem instance will consist of a single line containing the cards of a deck which has been shuffled between 1 and 10 times. All decks will be of size 52.

Note: The sample input below shows multiple lines for a problem instance. The actual input data for a problem instance is contained on a single line.

Output

For each problem instance, output the case number (starting at 1), followed by the number of shuffles that were used for that instance. If there were no mistakes made during the shuffling, output the line

```
No error in any shuffle
```

Otherwise, output a set of lines of the form

```
Error in shuffle n at location m
```

where *n* is a shuffle where an error occurred and *m* is the location of the error. Shuffles are numbered starting with 1 and the location value should indicate the first location of the two cards that were swapped in that shuffle (where the top of the deck is position 0). In the example described above, the cards in positions 4 and 5 (the cards numbered 2 and 28) are incorrect, so *m* would be 4 in this case. List all errors in order of increasing *n*. If one or more shuffles have no errors, do not print any line for them. If there are multiple solutions, pick the solution with the fewest number of errors (all test cases will have a unique solution of minimum size).

Sample Input

```
3
26 0 27 1 2 28 29 3 30 4 31 5 32 6 33 7 34
8 35 9 36 10 37 11 38 12 39 13 40 14 41 15
42 16 43 17 44 18 45 19 46 20 47 21 48 22
49 23 50 24 51 25
26 0 27 1 28 2 29 3 30 4 31 5 32 6 33 7 34
8 35 9 36 10 37 11 38 12 39 13 40 14 41 15
42 16 43 17 44 18 45 19 46 20 47 21 48 22
49 23 50 24 51 25
49 26 43 40 37 34 31 28 25 22 19 16 13 10
7 4 1 51 48 45 42 39 36 33 24 27 30 21 18
15 12 9 6 3 0 50 47 44 41 38 35 32 29 46
23 20 17 2 11 8 5 14
```

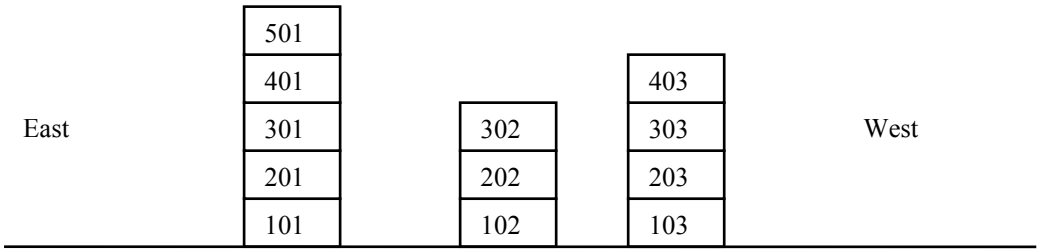
Output for the Sample Input

```
Case 1
Number of shuffles = 1
Error in shuffle 1 at location 4

Case 2
Number of shuffles = 1
No error in any shuffle

Case 3
Number of shuffles = 9
Error in shuffle 3 at location 3
Error in shuffle 7 at location 11
Error in shuffle 8 at location 38
```

Problem E
Lots of Sunlight
Input File: sunlight.in

The Apartment Construction Management (ACM) has several new high-rise apartment buildings in suburban Shanghai. With the booming economy, ACM expects a considerable profit on apartment leases. Because their apartments receive more direct sunlight, the company claims that these are nicer than others in the area. No other buildings obstruct the sunlight path to apartments in ACM's tall buildings.

ACM wants to verify this claim by telling potential residents exactly how much sunlight a given apartment receives. To offer customers a representative sample of sunlight hours, the company wants to advertise the sunlight hours for April 6, 2005. On that day in Shanghai, the sun rises at 5:37 am, and sets at 6:17 pm.



As shown above, apartments are in a series of buildings aligned east to west. The last two digits of the apartment number identify the building, starting with 01 for the east-most building. The other digits encode the apartment floor, with 1 as the ground floor.

The sun rises in the east and travels at a constant radial speed across the sky, until setting in the west. The only shadows are created by buildings (i.e. each building can cast a shadow on one or more other buildings). An apartment is considered to receive sunlight when either its eastern or western exterior wall is fully covered in sunlight or when the sun is directly overhead.

Input

The input file contains a series of descriptions of apartment complexes. Each description starts with a line containing a single integer n ($1 \le n < 100$) that is the number of apartment buildings in the complex. The next line has two integers w, the width (in east-west direction), and h, each apartment's height in meters. Next is a list of integers $m(1)$, $d(1)$, $m(2)$, $d(2)$, ..., $d(n-1)$, $m(n)$, where $m(i)$ is the number of apartments in apartment building i, and $d(i)$ is the distance, in meters, between the apartment building i and apartment building $i+1$.

The apartment complex description is followed by an integer list of apartments to query for sunlight hours and is terminated by a zero. The input file is terminated by a line consisting of the integer zero.

Output

For each apartment complex description, output its number in the sequence of descriptions. Then for each query, output the corresponding sunlight hours, using the 24-hour time format. Truncate all times down to the nearest second. If the query refers to an apartment that does not exist, indicate that the apartment does not exist. Follow the format shown in the sample output.

Sample Input	Output for the Sample Input
3 6 4 5 6 3 3 4 302 401 601 303 0 4 5 3 4 5 7 8 5 4 3 101 302 503 0 0	Apartment Complex: 1 Apartment 302: 10:04:50 - 13:23:47 Apartment 401: 05:37:00 - 17:13:57 Apartment 601: Does not exist Apartment 303: 09:21:19 - 18:17:00 Apartment Complex: 2 Apartment 101: 05:37:00 - 12:53:32 Apartment 302: 09:08:55 - 14:52:47 Apartment 503: 09:01:12 - 18:17:00

Problem F
Crossing Streets
Input File: streets.in

Peter Longfoot is a student at the university of Suburbia. Every morning, Peter leaves home to walk to the university. Many other students are driving their cars or riding their bicycles to campus, but Peter prefers to walk, avoiding the chaotic traffic of the city as much as possible.

Unfortunately, Peter cannot avoid the traffic entirely, since he has to cross streets in order to reach the university. Recently, Peter has wondered how to minimize the number of streets he has to cross. For example, in the following map, streets are drawn as horizontal and vertical lines. To reach the university starting at his home, Peter has to cross at least two streets. Peter cannot cross a street pair at an intersection and Peter cannot walk along a street.

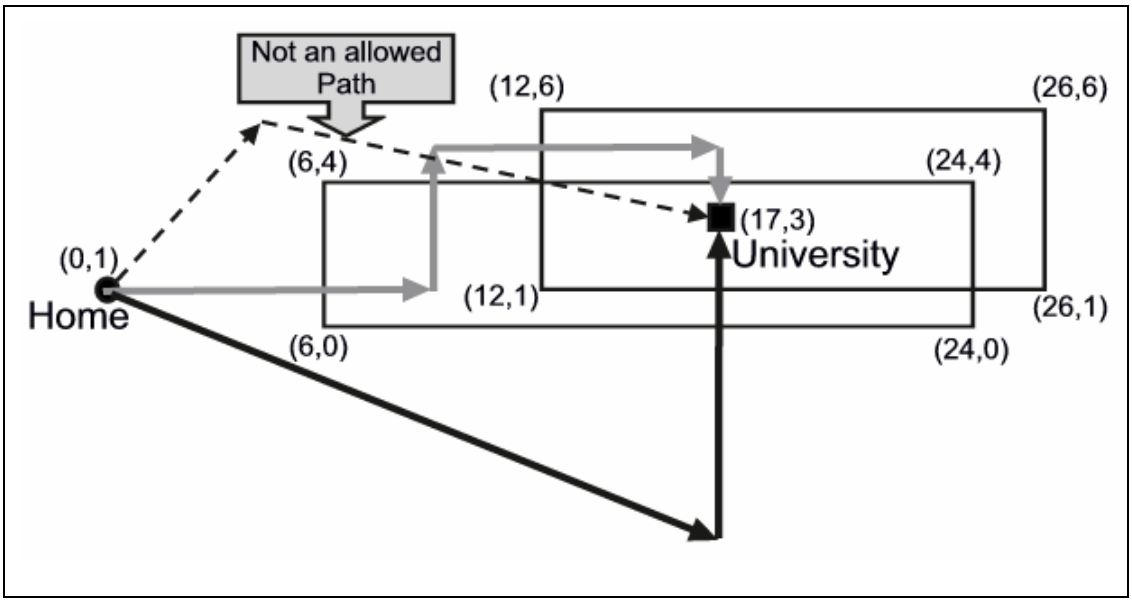


Figure: Streets are shown as straight lines and the arrows show possible walking paths for Peter. The black arrows show one possible path for Peter where he only has to cross two streets. The gray arrows show a path for Peter where he needs to cross four streets. The path shown by the dotted arrows is not legal because it crosses two streets at an intersection. The figure above corresponds to the first sample input.

Peter knows the locations of all streets in the city, but he has trouble figuring out the best way to the university. So you must write a program to help him.

Input

The input consists of several descriptions of cities. Each description starts with a line containing a single integer n ($1 \le n \le 500$), the number of streets in the city. This is followed by n lines, each containing four integers x_1, y_1, x_2, y_2, indicating that there is a street from coordinates (x_1, y_1) to (x_2, y_2). All streets are parallel to either the x- or y-axis, since the city is built on a square grid. Streets can overlap, in which case they are counted as only one street. The city description is concluded by a line containing four integers x_h, y_h, x_u, y_u, the coordinates (x_h, y_h) of Peter's home, and the coordinates (x_u, y_u) of the university, respectively. Neither Peter's home nor the university will be located on a street. You should consider the streets as straight-line segments, so the streets have no width. Although the endpoints of the streets are integers, Peter doesn't need to walk along the grids. He can walk in any direction he likes. The magnitudes of all integers in the input file are less than 2×10^9.

The input is terminated by a line consisting of the integer zero.

Output

For each city description, first output its number in the sequence of descriptions. Then output the minimum number of streets that Peter has to cross to go from his home to the university.

Follow the format of the sample output.

Sample Input	Output for the Sample Input
8 6 0 24 0 24 0 24 4 24 4 6 4 6 4 6 0 12 1 26 1 26 1 26 6 26 6 12 6 12 6 12 1 0 1 17 3 1 10 10 20 10 1 1 30 30 0	City 1 Peter has to cross 2 streets City 2 Peter has to cross 0 streets

Problem G
Tiling the Plane
Input File: tiling.in

A polygon is said to "tile the plane" if a collection of identical copies of the polygon can be assembled to fill an unbounded two-dimensional plane without any gaps or overlap. For example, Figure 1 shows an L-shaped polygon, and Figure 2 shows how a portion of the plane can be tiled with copies of the polygon. You must write a program to determine whether a given polygon can tile the plane.

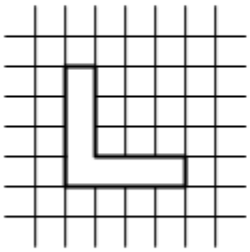


Figure 1: A test polygon shown against a grid of unit squares

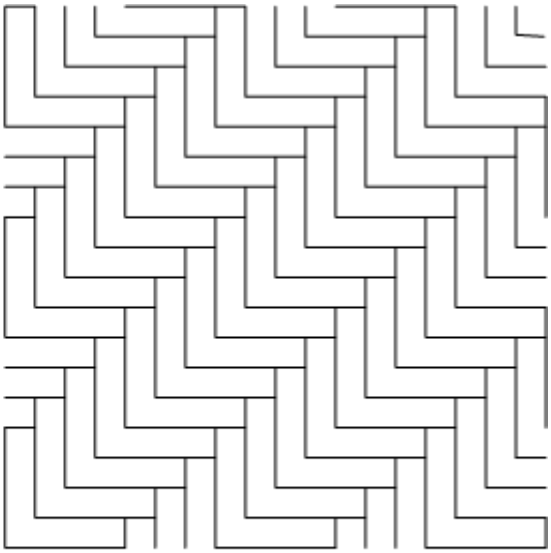


Figure 2: A portion of the plane tiled with the test polygon

Each test case consists of a closed polygon in which every vertex is at a right angle and the length of every side is an integer multiple of a unit length. You may make as many copies of the polygon as you like, and you may move them over the plane, but you may not rotate or reflect any polygon.

You might find the following information useful: It is known that there are only two fundamentally different tilings of the plane, the regular tiling by squares (chessboard tiling) and the tiling by regular hexagons (honeycomb tiling). A polygon can therefore tile the plane if and only if it satisfies one of the following two conditions:

1. There are points A, B, C, D in order on the polygon boundary (the points are not necessarily vertices of the polygon) such that the polygon boundaries from A to B and from D to C are congruent and the boundaries from B to C and from A to D are congruent. This leads to a tiling equivalent to the square tiling.
2. There are points A, B, C, D, E, F in order on the polygon boundary, such that the boundary pairs AB and ED, BC and FE, CD and AF are congruent. This leads to a tiling equivalent to the hexagon tiling.

Input

The input contains the descriptions of several polygons, each description consisting of one input line. Each description begins with an integer n ($4 \le n \le 50$) that represents the number of sides of the polygon. This number is followed by descriptions of n line segments which (taken in order) form a counterclockwise traversal of the perimeter of the polygon. Each line segment description consists of a letter followed by an integer. The letter is "N", "E", "S", or "W", representing the direction of the line segment as North, East, South, or West, respectively. The integer represents the length of the line segment as a multiple of the unit length. The described polygon will not touch or intersect itself.

The input is terminated by a line consisting of the integer zero.

Output

For each polygon in the input, print one output line. Print the number of the polygon in the input, followed by the word "`Possible`" if it is possible to tile the plane with the test polygon, or "`Impossible`" otherwise. Follow the format of the sample output.

Sample Input	Output for the Sample Input
6 N 3 W 1 S 4 E 4 N 1 W 3 8 E 5 N 1 W 3 N 3 E 2 N 1 W 4 S 5 0	Polygon 1: Possible Polygon 2: Impossible

Problem H

The Great Wall Game

Input File: wall.in

Hua and Shen have invented a simple solitaire board game that they call "The Great Wall Game." The game is played with n stones on an $n \times n$ grid. The stones are placed at random in the squares of the grid, at most one stone per square. In a single move, any single stone can move into an unoccupied location one unit horizontally or vertically in the grid. The goal of the puzzle is to create a "wall," i.e., to line up all n stones in a straight line either horizontally, vertically, or diagonally using the fewest number of moves. An example for the case $n = 5$ is shown in Figure 1(a). In Figure 1(b) it is shown that with six moves we can line all the stones up diagonally. No smaller number of moves suffices to create a line of five stones. (However, there are other solutions using six moves, e.g., we can line up all five stones in the third column using six moves.)

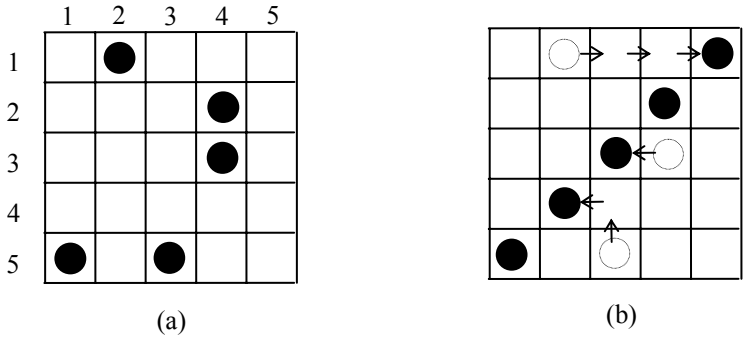


Figure 1. Starting board (a) and a 6-move solution (b) for $n = 5$

There is just one problem – Hua and Shen have no idea what the minimum number of moves is for any given starting board. They would like you to write a program that can take any starting configuration and determine the fewest number of moves needed to create a wall.

Input

The input consists of multiple cases. Each case begins with a line containing an integer n, $1 \le n \le 15$. The next line contains the row and column numbers of the first stone, followed by the row and column numbers of the second stone, and so on. Rows and columns are numbered as in the above diagram. The input data for the last case will be followed by a line containing a single zero.

Output

For each input case, display the case number (1, 2, …) followed by the minimum number of moves needed to line up the n stones into a straight-line wall. Follow the format shown in the sample output.

Sample Input	Output for the Sample Input
5 1 2 2 4 3 4 5 1 5 3 2 1 1 1 2 3 3 1 1 2 2 2 0	Board 1: 6 moves required. Board 2: 0 moves required. Board 3: 1 moves required.

Problem I
Workshops
Input File: workshops.in

The first Californian Conference on Holism took place back in 1979 in San Francisco. The term "Californian" was a slight exaggeration, as all 23 participants actually lived in San Francisco. Several years later, in 1987, the conference was truly Californian, with 337 participants from all over the state. Since then, the number of participants has been growing like the size of memory chips. In 1993 the conference was renamed the American Conference on Holism (2549 participants), and a second renaming (World Conference on Holism) followed in 1997, when the number of participants from all over the world had grown to 9973. The conference obtained its present name (Galactic Conference on Holism) in 2003 after some discussion as to whether or not the word Galactic was intended to exclude extragalactic life forms. Still the next year, all registered participants were terrestrial—though a few participants positively reported to have sensed extraterrestrial presence.

The number of workshops grew with the number of participants. For the upcoming conference, the organization has to face some down to earth but very nasty scheduling problems. For the 2005 conference the board has decided to have no more than 1000 workshops concurrently. Nevertheless they had to rent every hall or classroom they could lay their hands on. Some of these rooms are available for a restricted time only.

In the morning of the first day the opening meeting takes place in a football stadium, and in the afternoon the participants attend workshops. Before lunch each participant has to indicate which workshop he or she wants to join that afternoon. The organizing staff then has a list of all workshops, including the duration and the number of participants for each workshop. They also have a list of all available rooms, with the capacity of each room, and the time this specific room must be cleared. With this information the staff must schedule each workshop in a room of sufficient capacity and sufficient availability in time. As this problem is not necessarily solvable, some overflow capacity is supplied by tents in the football stadium. These tents have plenty of capacity, but they are unpleasantly warm and noisy. So the organizing staff wants the schedule to minimize the number of tent workshops—that is, workshops that are not assigned to a room. If there are multiple solutions that minimize the number of tent workshops, the staff wants to minimize the number of participants attending tent workshops.

We ask you to supply such a schedule (preferably before lunch is over).

Input

The input file contains several trials. Each trial consists of two parts: the list of workshops, and the list of rented rooms.

The list of workshops starts with a line containing the number of workshops w ($0 < w \leq 1000$). Each of the next w lines contains two numbers, describing a workshop. The first number is the number p of participants ($0 < p \leq 100$), and the second number is the duration d of the workshop in minutes ($0 < d \leq 300$). For your convenience, other details of the workshops are omitted. All workshops start at 14:00.

The list of rented rooms starts with a line containing the number of rented rooms r ($0 < r \leq 1000$). Each of the following r lines contains the description of a rented room. A line describing a rented room contains the number s of seats in the room ($0 < s \leq 100$), followed by the time when the room must be cleared, in the format *hh:mm* where *hh* represents the hour and *mm* represents the minute, using a 24-hour clock. All the rooms are available starting at 14:00. All times when rooms must be cleared are between 14:01 and 23:59, inclusive.

The input is terminated by a line consisting of the integer zero.

Output

For each trial in the input file the output must contain a line consisting of the trial number, the number of tent workshops, and the number of participants attending tent workshops. Follow the format shown in the sample output.

Sample Input	Output for the Sample Input
1 20 60 1 30 16:00 2 20 60 50 30 1 30 14:50 0	Trial 1: 0 0 Trial 2: 2 70

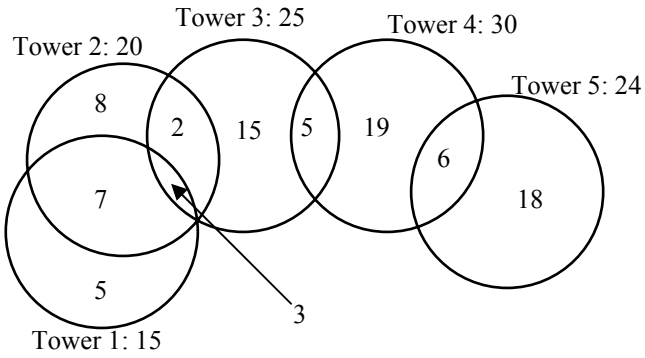
Problem J
Zones
Input File: zones.in

Telephone poles are part of an outdated technology. Cell phones nowadays allow us to call whoever we want, wherever we want, independent of any wire. There is one problem — without a service tower nearby a cell phone is useless.

In the absence of hills and mountains, a service tower will provide coverage for a circular area. Instead of planning where to place the wires, a wireless telephone company has to plan where to build its service towers. Building the towers too far apart causes holes in the coverage and increases complaints. Building the towers too close to each other results in large areas covered by more than one service tower, which is redundant and inefficient.

International Cell Phone Company is developing a network strategy to determine the optimal placement of service towers. Since most customers have replaced their old wired home phones by cell phones, the strategy for planning service towers is therefore to cover as many homes of customers as possible.

The figure below shows the service areas for the five towers ICPC's strategic department planned to build this year. Tower 5 will serve 24 customers, 6 of which are also served by tower 4. Towers 1, 2 and 3 have a common service area containing 3 customers.



Shortly after the plans for these five towers had been published, higher management issued a stop on all tower building. Protesting customers forced them to weaken this decree, and allow the building of three towers out of the five planned. These three towers should serve as many customers as possible, of course. Finding the best possible choice for the towers to build is not trivial (the best choice in this case is towers 2, 4 and 5, serving 68 customers).

You must write a program to help ICPC choose which towers to build in cases like these. If several choices of towers serve the same number of customers, choices including tower 1 are preferable. If this rule still leaves room for more than one solution, solutions including tower 2 are preferable, and so on.

Input

The input file contains several test cases. The first line of each test case contains two positive integers: the number n ($n \leq 20$) of towers planned, and the number of towers to be actually built. The next line contains n numbers, each giving the number of customers served by a planned tower. (The first number refers to the first tower, and so on.) No tower serves more than a million people. The next line contains the number m ($m \leq 10$) of common service areas. Each of the next m lines contains the description of a common service area. Such a line starts with the number t ($t > 1$) of towers for which this is a common service area, followed by the t numbers of the towers. The last number on the line is the number of customers in the common service area. The last line of the input file consists of the two integers 0 0.

Output

For each test case, display the number of customers served and the best choice for the location of the towers. Follow the format of the sample output.

Sample Input	Output for the Sample Input
5 3 15 20 25 30 24 5 2 1 2 7 3 1 2 3 3 2 2 3 2 2 3 4 5 2 4 5 6 5 3 25 25 25 25 25 4 2 1 2 5 2 2 3 5 2 3 4 5 2 4 5 5 5 3 25 25 25 25 25 0 0 0	Case Number 1 Number of Customers: 68 Locations recommended: 2 4 5 Case Number 2 Number of Customers: 75 Locations recommended: 1 3 5 Case Number 3 Number of Customers: 75 Locations recommended: 1 2 3