The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem A

Carl the Ant

Input File: carl.in

Ants leave small chemical trails on the ground in order to mark paths for other ants to follow. Ordinarily these trails follow rather straight lines. But in one ant colony there is an ant named Carl, and Carl is not an ordinary ant. Carl will often zigzag for no apparent reason, sometimes crossing his own path numerous times in the process. When other ants come to an intersection, they always follow the path with the strongest scent, which is the most recent path that leads away from the intersection point.

Ants are 1 centimeter long, move and burrow at 1 centimeter per second, and follow their paths exactly (bending at right angles when moving around corners). Ants cannot cross or overlap each other. If two ants meet at the exact same instant at an intersection point, the one that has been on Carl's path the longest has the right of way; otherwise, the ant that has been waiting the longest at an intersection will move first.

Carl burrows up from the ground to start at the origin at time 0. He then walks his path and burrows back down into the ground at the endpoint. The rest of the ants follow at regular intervals. Given the description of Carl's path and when the other ants start the path, you are to determine how long it takes the entire set of ants to finish burrowing back into the ground. All the ants are guaranteed to finish.

Input

Input consists of several test cases. The first line of the input file contains a single integer indicating the number of test cases.

The input for each test case starts with a single line containing three positive integers n ($1 \le n \le 50$), m ($1 \le m \le 100$), and d ($1 \le d \le 100$). Here, n is the number of line segments in Carl's path, m is the number of ants traveling the path (including Carl), and d is the time delay before each successive ant's emergence. Carl (who is numbered 0) starts at time 0. The next ant (ant number 1) will emerge at time d, the next at time $2d$, and so on. If the burrow is blocked, the ants will emerge as soon as possible in the correct order.

Each of the next n lines for the test case consists of a unique integer pair $x\ y$ ($-100 \le x, y \le 100$), which is the endpoint of a line segment of Carl's path, in the order that Carl travels. The first line starts at the origin (0,0) and the starting point of every subsequent line is the endpoint of the previous line.

For simplicity, Carl always travels on line segments parallel to the axes, and no endpoints lie on any segment other than the ones which they serve as an endpoint.

Output

The output for each case is described as follows:

```
Case C:
Carl finished the path at time t1
The ants finished in the following order:
a1 a2 a3 ... am
The last ant finished the path at time t2
```

Here, `C` is the case number (starting at 1), `a1`, `a2`, `a3`, … `am` are the ant numbers in the order that they go back underground, and `t1` and `t2` are the times (in seconds) at which Carl and the last ant finish going underground. You should separate consecutive cases with a single blank line.

Sample Input	Output for the Sample Input
2 4 7 4 0 4 2 4 2 2 -2 2 4 7 2 0 4 2 4 2 2 -2 2	Case 1: Carl finished the path at time 13 The ants finished in the following order: 0 2 1 3 4 5 6 The last ant finished the path at time 29 Case 2: Carl finished the path at time 13 The ants finished in the following order: 0 4 1 5 2 6 3 The last ant finished the path at time 19

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

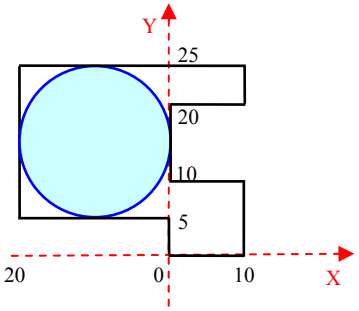
sponsored by IBM

Problem B

Heliport

Input File: heliport.in

In these fast-paced times, companies are investing in heliports to reduce travel time for their busy executives. The heliports are typically circular landing pads, constructed on the roofs of the companies' headquarters.



You must write a program that finds the largest radius for a circular heliport that can be constructed on the flat roof of a building that is in the form of a simple polygon. Since this is merely the design phase of the construction effort, your program must find only the radius of the heliport. The maximum radius for a heliport in the diagram shown is 10.

Input File

The input file contains several test cases. Each test case consists of two lines. The first line consists of an even integer n ($4 \le n \le 20$), which is the number of the sides of the building. The second line consists of n pairs of the form (m, d), where m is an integer ($1 \le m \le 50$) and d is a letter (U, R, D, L). Assuming the roof is drawn on the Cartesian plane, m is the length of a roof boundary segment and d is the direction of that segment as you travel counterclockwise around the roof. U, R, D, and L mean "Up," "Right," "Down," and "Left" respectively. The boundary segments of the roof, which are parallel to the x and y axes, are given in counterclockwise order. The starting position is the origin (0, 0).

Input for the last test case is followed by a line consisting of the number 0.

Output File

For each test case, the output consists of a separate line containing the case number (starting with 1) and a real number (rounded to two digits after the decimal point) representing the radius of the heliport. Print a blank line between cases as shown in the sample output.

Sample Input

```
4
2 R 2 U 2 L 2 D
10
10 R 10 U 10 L 10 U 10 R 5 U 30 L 20 D 20 R 5 D
0
```

Output for the Sample Input

```
Case Number 1 radius is: 1.00

Case Number 2 radius is: 10.00
```

The 2004 ACM Programming Contest World Finals sponsored by IBM

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem C

Image Is Everything

Input File: image.in

Your new company is building a robot that can hold small lightweight objects. The robot will have the intelligence to determine if an object is light enough to hold. It does this by taking pictures of the object from the 6 cardinal directions, and then inferring an upper limit on the object's weight based on those images. You must write a program to do that for the robot.

You can assume that each object is formed from an $N \times N \times N$ lattice of cubes, some of which may be missing. Each $1 \times 1 \times 1$ cube weighs 1 gram, and each cube is painted a single solid color. The object is not necessarily connected.

Input

The input for this problem consists of several test cases representing different objects. Every case begins with a line containing N, which is the size of the object ($1 \leq N \leq 10$). The next N lines are the different $N \times N$ views of the object, in the order front, left, back, right, top, bottom. Each view will be separated by a single space from the view that follows it. The bottom edge of the top view corresponds to the top edge of the front view. Similarly, the top edge of the bottom view corresponds to the bottom edge of the front view. In each view, colors are represented by single, unique capital letters, while a period (.) indicates that the object can be seen through at that location.

Input for the last test case is followed by a line consisting of the number 0.

Output

For each test case, print a line containing the maximum possible weight of the object, using the format shown below.

Sample Input	Output for the Sample Input
3 .R. YYR .Y. RYY .Y. .R. GRB YGR BYG RBY GYB GRB .R. YRR .Y. RRY .R. .Y. 2 ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ 0	Maximum weight: 11 gram(s) Maximum weight: 8 gram(s)

The 2004 ACM Programming Contest World Finals sponsored by IBM

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem D

Insecure in Prague

Input: insecure.in

Prague is a dangerous city for developers of cryptographic schemes. In 2001, a pair of researchers in Prague announced a security flaw in the famous PGP encryption protocol. In Prague in 2003, a flaw was discovered in the SSL/TLS (Secure Sockets Layer and Transport Layer Security) protocols. However, Prague's reputation for being tough on cryptographic protocols hasn't stopped the part-time amateur cryptographer and full-time nutcase, Immanuel Kant-DeWitt (known to his friends as "I. Kant-DeWitt"), from bringing his latest encryption scheme to Prague. Here's how it works:

A plain text message p of length n is to be transmitted. The sender chooses an integer $m \geq 2n$, and integers s, t, i, and j, where $0 \leq s, t, i, j < m$ and $i < j$. The scheme works as follows: m is the length of the transmitted ciphertext string, c. Initially, c contains m empty slots. The first letter of p is placed in position s of c. The kth letter, $k \geq 2$, is placed by skipping over i empty slots in c after the (k-1)st letter, wrapping around to the beginning of c if necessary. Slots already containing letters are not counted as empty. For instance, if the message is PRAGUE, if $s = 1$, $i = 6$, and $m = 15$, then the letters are placed in c as follows:

```
A  P  _  U  _  _  _  _  R  G  _  _  E  _  _
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
```

Starting with the first empty slot in or after position t in string c, the plain text message is entered again, but this time skipping j empty slots between letters. For instance, if $t = 0$ and $j = 8$, the second copy of p is entered as follows (beginning in position 2, the first empty slot starting from $t = 0$):

```
A  P  P  U  R  _  A  U  R  G  E  G  E  _  _
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
```

Finally, any remaining unfilled slots in c are filled in with randomly chosen letters:

```
A  P  P  U  R  A  A  U  R  G  E  G  E  W  E
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
```

Kant-DeWitt believes that the duplication of the message, combined with the use of random letters, will confuse decryption schemes based upon letter frequencies and that, without knowledge of s and i, no one can figure out what the original message is. Your job is to try to prove him wrong. Given a number of ciphertext strings (and no additional information), you will determine the longest possible message that could have been encoded using the Kant-DeWitt method.

Input

A number of ciphertext strings, one per line. Each string will consist only of upper case alphabetic letters, with no leading or trailing blanks; each will have length between 2 and 40.

Input for the last test case is followed by a line consisting of the letter X.

Output

For each input ciphertext string, print the longest string that could be encrypted in the ciphertext. If more than one string has the longest length, then print "`Codeword not unique`". Follow the format of the sample output given below.

Sample Input	Output for the Sample Input
APPURAAURGEGEWE ABABABAB THEACMPROGRAMMINGCONTEST X	Code 1: PRAGUE Code 2: Codeword not unique Code 3: Codeword not unique

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem E

Intersecting Dates

Input File: intersect.in

A research group is developing a computer program that will fetch historical stock market quotes from a service that charges a fixed fee for each day's quotes that it delivers. The group has examined the collection of previously-requested quotes and discovered a lot of duplication, resulting in wasted money. So the new program will maintain a list of all past quotes requested by members of the group. When additional quotes are required, only quotes for those dates not previously obtained will be fetched from the service, thus minimizing the cost.

You are to write a program that determines when new quotes are required. Input for the program consists of the date ranges for which quotes have been requested in the past and the date ranges for which quotes are required. The program will then determine the date ranges for which quotes must be fetched from the service.

Input

There will be multiple input cases. The input for each case begins with two non-negative integers *NX* and *NR*, ($0 \le NX, NR \le 100$). *NX* is the number of existing date ranges for quotes requested in the past. *NR* is the number of date ranges in the incoming requests for quotes. Following these are *NX* + *NR* pairs of dates. The first date in each pair will be less than or equal to the second date in the pair. The first *NX* pairs specify the date ranges of quotes which have been requested and obtained in the past, and the next *NR* pairs specify the date ranges for which quotes are required.

Two zeroes will follow the input data for the last case.

Each input date will be given in the form *YYYYMMDD*. *YYYY* is the year (1700 to 2100), *MM* is the month (01 to 12), and *DD* is the day (in the allowed range for the given month and year). Recall that months 04, 06, 09, and 11 have 30 days, months 01, 03, 05, 07, 08, 10, and 12 have 31 days, and month 02 has 28 days except in leap years, when it has 29 days. A year is a leap year if it is evenly divisible by 4 and is not a century year (a multiple of 100), or if it is divisible by 400.

Output

For each input case, display the case number (1, 2, …) followed by a list of any date ranges for which quotes must be fetched from the service, one date range per output line. Use the American date format shown in the sample output below. Explicitly indicate (as shown) if no additional quotes must be fetched. If two date ranges are contiguous or overlap, then merge them into a single date range. If a date range consists of a single date, print it as a single date, not as a range consisting of two identical dates. Display the date ranges in chronological order, starting with the earliest date range.

Sample Input	Output for the Sample Input

1 1
19900101 19901231
19901201 20000131
0 3
19720101 19720131
19720201 19720228
19720301 19720301
1 1
20010101 20011231
20010515 20010901
0 0
	Case 1:
	    1/1/1991 to 1/31/2000
	
	Case 2:
	    1/1/1972 to 2/28/1972
	    3/1/1972
	
	Case 3:
	    No additional quotes are required.

The 2004 ACM Programming Contest World Finals sponsored by IBM

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem F

Merging Maps

Input File: maps.in

Pictures taken from an airplane or satellite of an area to be mapped are often of sufficiently high resolution to uniquely identify major features. Since a single picture can cover only a small portion of the earth, mapping larger areas requires taking pictures of smaller overlapping areas, and then merging these to produce a map of a larger area.

For this problem you are given several maps of rectangular areas, each represented as an array of single-character cells. A cell contains an uppercase alphabetic character ('A' to 'Z') if its corresponding area contains an identifiable major feature. Different letters correspond to different features, but the same major feature (such as a road) may be identified in multiple cells. A cell contains a hyphen ('-') if no identifiable feature is located in the cell area. Merging two maps means overlaying them so that one or more common major features are aligned. A cell containing a major feature in one map can be overlaid with a cell not containing a major feature in the other. However, different major features (with different letters) cannot be overlaid in the same cell.

```
        --A-C     C----     C----     ----D     -D--C
        ----D     D---F     -----     -E--B     ----G
        ----B     B----     B-A-C     -----     ----B
Map #     1         2         3         4         5
```

Consider the five 3-row, 5-column maps shown above. The rightmost column of map 1 perfectly matches the leftmost column of map 2, so those maps could be overlaid to yield a 3-row, 9-column map. But map 1 could also overlay map 3 as well, since the C and B features in the rightmost column of map 1 match those in the leftmost column of map 3; the D does not perfectly match the '-' in the center of the column, but there is no conflict. In a similar manner, the top row of map 1 could also overlay the bottom row of map 3.

The "score" of a pair of maps indicates the extent to which the two maps match. The score of an overlay of a pair of maps is the number of cells containing major features that coincide in the overlay that gives the best match. The score for the map pair is the maximum score for the possible overlays of the maps. Thus, the score for a pair of maps each having 3 rows and 5 columns must be in the range 0 to 15.

An "offset" is a pair of integers (*r*,*c*) that specifies how two maps, *a* and *b*, are overlaid. The value of *r* gives the offset of rows in *b* relative to rows in *a*; similarly, *c* gives the offset of columns in *b* relative to columns in *a*. For example, the overlay of map 1 and map 2 shown above has the offset (0,4) and a score of 3. The two overlays of map 1 and map 3 yielding scores of 2 have offsets of (0,4) and (-2,0).

The following steps describe how to merge a sequence of maps:

1. Merge the pair of maps in the sequence that yield the highest positive score (resolving ties by choosing pair that has the map with the lowest sequence number).
2. Remove the maps that were merged from the sequence.
3. Add the resulting merged map to the sequence, giving it the next larger sequence number.

In the example above, maps 1 and 2 would be merged to produce map 6, and maps 1 and 2 would be removed from the sequence. Steps 1, 2 and 3 are repeated until only a single map remains in the sequence, or until none of the maps in the sequence can be merged (that is, until the overlay score for each possible map pair is zero).

If two maps can be merged in several ways to yield the same score, then merge them using the smallest row offset. If the result is still ambiguous, use the smallest row offset and the smallest column offset.

Input

The input will contain one or more sets of data, each containing between 2 and 10 maps. Each set of data begins with an integer specifying the number of maps in the sequence. The maps follow, each beginning with a line containing two integers *NR* and *NC* ($1 \le NR, NC \le 10$) that specify the number of rows and columns in the map that immediately follows on the next *NR* lines. The first *NC* characters on each of these *NR* lines are the map data, and any trailing characters on such lines are to be ignored.

Input for the last test case is followed by a line consisting of the number 0.

Output

For each set of data, display the input case number (1, 2, …) and the merged maps, each identified with its sequence number and enclosed by a border. The output should be formatted as shown in the samples below. No merged map will have more than 70 columns.

Sample Input

```
5
3 5
--A-C
----D
----B
3 5
C----
D---F
B----
3 5
C----
-----
B-A-C
3 5
----D
-E--B
-----
3 5
-D--C
----G
----B
2
3 5
----A
----B
----C
3 5
A----
B----
D----
0
```

Output for the Sample Input

```
Case 1
    MAP 9:
    +-------------+
    |-D--C--------|
    |----G--------|
    |----B-A-C----|
    |--------D---F|
    |-----E--B----|
    |-------------|
    +-------------+

Case 2
    MAP 1:
    +-----+
    |----A|
    |----B|
    |----C|
    +-----+

    MAP 2:
    +-----+
    |A----|
    |B----|
    |D----|
    +-----+
```

Problem G
Navigation
Input File: navigation.in

Global Positioning System (GPS) is a navigation system based on a set of satellites orbiting approximately 20,000 kilometers above the earth. Each satellite follows a known orbit and transmits a radio signal that encodes the current time. If a GPS-equipped vehicle has a very accurate clock, it can compare its own local time with the time encoded in the signals received from the satellites. Since radio signals propagate at a known rate, the vehicle can compute the distance between its current location and the location of the satellite when the signal was broadcast. By measuring its distance from several satellites in known orbits, a vehicle can compute its position very accurately.

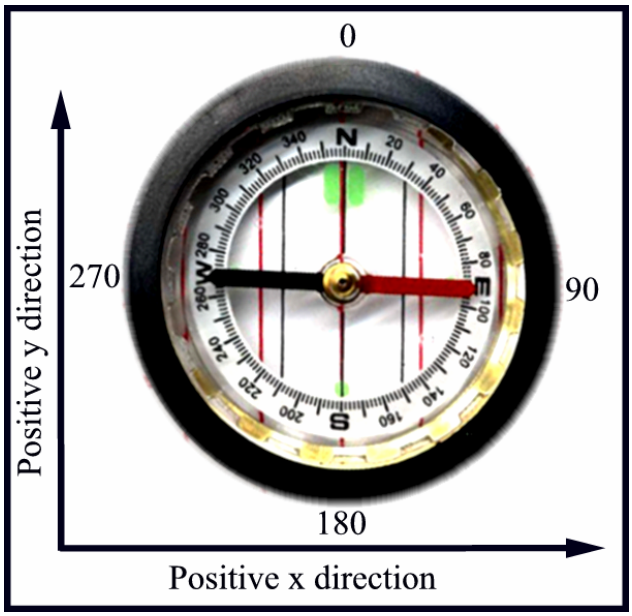


Figure 1: The Compass

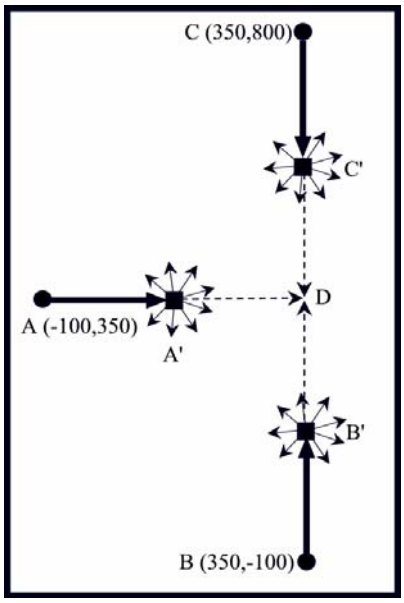


Figure 2: 1st Sample Input

You must write a simple "autopilot" program based on GPS navigation. To make the problem easier, we state it as a two-dimensional problem. In other words, you do not need to take into account the curvature of the earth or the altitude of the satellites. Furthermore, the problem uses speeds that are more appropriate for airplanes and sound waves than for satellites and radio waves.

Given a set of signals from moving sources, your program must compute the receiving position on the Cartesian plane. Then, given a destination point on the plane, your program must compute the compass heading required to go from the receiving position to the destination. All compass headings are stated in degrees. Compass heading 0 (North) corresponds to the positive *y* direction, and compass heading 90 (East) corresponds to the positive *x* direction, as shown in Figure 1.

Input

The input consists of multiple data sets.

The first line of input in each data set contains an integer N ($1 \le N \le 10$), which is the number of signal sources in the set. This is followed by three floating point numbers: t, x, and y. Here, t denotes the exact local time when

all the signals are received, represented in seconds after the reference time (time 0), and *x* and *y* represent the coordinates of the destination point on the Cartesian plane. Each of the next *N* lines contains four floating-point numbers that carry information about one signal source. The first two numbers represent the known position of the signal source on the Cartesian plane at the reference time. The third number represents the direction of travel of the signal source in the form of a compass heading *D* ($0 \le D < 360$). The fourth number is the time that is encoded in the signal—that is, the time when the signal was transmitted, represented in seconds after the reference time. The magnitudes of all numbers in the input file are less than 10000 and no floating-point number has more than 5 digits after the decimal point.

The last data set is followed by a line containing four zeros.

The unit distance in the coordinate space is one meter. Assume that each signal source is moving over the Cartesian plane at a speed of 100 meters per second and that the broadcast signal propagates at a speed of 350 meters per second. Due to inaccuracies in synchronizing clocks, assume that your distance calculations are accurate only to 0.1 meter. That is, if two points are computed to be within 0.1 meter of each other, you should treat them as the same point. There is also the possibility that a signal may have been corrupted in transmission, so the data received from multiple signals may be inconsistent.

Output

For each trial, print the trial number followed by the compass heading from the receiving location to the destination, in degrees rounded to the nearest integer. Use the labeling as shown in the example output. If the signals do not contain enough information to compute the receiving location (that is, more than one position is consistent with the signals), print "Inconclusive." If the signals are inconsistent (that is, no position is consistent with the signals), print "Inconsistent." If the receiving location is within 0.1 meter of the destination, print "Arrived." If the situation is Inconclusive or Inconsistent, then you do not need to consider the case Arrived.

Figure 2 on the previous page corresponds to the first sample input. The locations of the three satellites at time $t = 0$ are *A* (-100,350), *B* (350,-100) and *C* (350,800). The signals received by the GPS unit were transmitted at time $t = 1.75$, when the satellites were at locations *A'*, *B'*, and *C'* (however, in general the signals received by the GPS unit might have been transmitted at different times). The signals from the three satellites converge at *D* at time $t = 2.53571$, which means *D* is the location of the receiving GPS unit. From point *D*, a compass course of 45 degrees leads toward the destination point of (1050, 1050).

Sample Input	Output for the Sample Input
3 2.53571 1050.0 1050.0 -100.0 350.0 90.0 1.75 350.0 -100.0 0.0 1.75 350.0 800.0 180.0 1.75 2 2.0 1050.0 1050.0 -100.0 350.0 90.0 1.0 350.0 -100.0 0.0 1.0 0 0 0 0	Trial 1: 45 degrees Trial 2: Inconclusive

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem H

Tree-Lined Streets

Input File: streets.in

The city council of Greenville recently voted to improve the appearance of inner city streets. To provide more greenery in the scenery, the city council has decided to plant trees along all major streets and avenues. To get an idea of how expensive this urban improvement project will be, the city council wants to determine how many trees will be planted. The planting of trees is limited in two ways:

- Along a street, trees have to be planted at least 50 meters apart. This is to provide adequate growing space, and to keep the cost of the project within reasonable limits.
- Due to safety concerns, no tree should be planted closer than 25 meters along a street to the nearest intersection. This is to ensure that traffic participants can easily see each other approaching an intersection. Traffic safety should not be compromised by reducing visibility.

All streets considered in this project are straight. They have no turns or bends.

The city council needs to know the maximum number of trees that can be planted under these two restrictions.

Input

The input consists of descriptions of several street maps. The first line of each description contains an integer n ($1 \le n \le 100$), which is the number of streets in the map. Each of the following n lines describes a street as a line segment in the Cartesian plane. An input line describing a street contains four integers x_1, y_1, x_2, and y_2. This means that this street goes from point (x_1, y_1) to point (x_2, y_2). The coordinates x_1, y_1, x_2, and y_2 are given in meters, ($0 \le x_1, y_1, x_2, y_2 \le 100000$). Every street has a positive length. Each end point lies on exactly one street.

For each street, the distances between neighboring intersections and/or the end points of the street are not exact multiples of 25 meters. More precisely, the difference of such a distance to the nearest multiple of 25 meters will be at least 0.001 meters. At each intersection, exactly two streets meet.

Input for the last street map description is followed by a line consisting of the number 0.

Output

For each street map described in the input, first print its number in the sequence. Then print the maximum number of trees that can be planted under the restrictions specified above. Follow the format in the sample output given below.

Sample Input	Output for the Sample Input
3 0 40 200 40 40 0 40 200 0 200 200 0 4 0 30 230 30 0 200 230 200 30 0 30 230 200 0 200 230 3 0 1 121 1 0 0 121 4 0 4 121 0 0	Map 1 Trees = 13 Map 2 Trees = 20 Map 3 Trees = 7

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem I

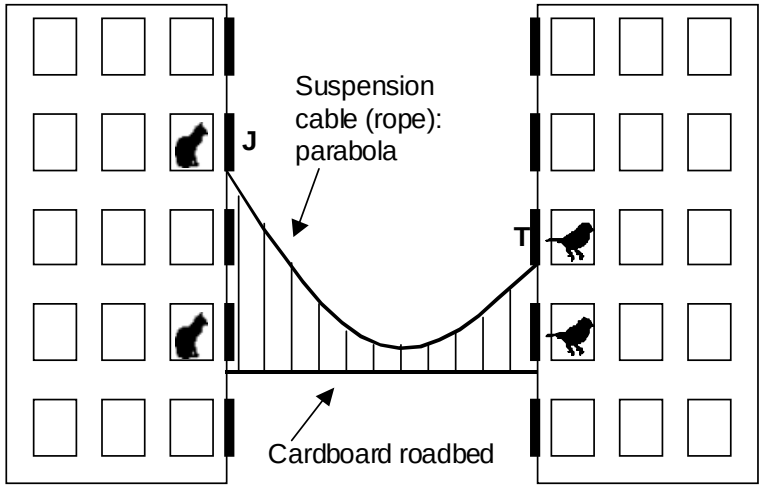
Suspense!

Input File: suspense.in

Jan and Tereza live in adjoining buildings and their apartments face one another. For their school science project, they want to construct a miniature suspension bridge made of rope, string, and cardboard connecting their two buildings. Two pieces of identical-length rope form the main suspension cables, which are attached to the bottoms of their windows. The cardboard "roadbed" of the bridge is held up by numerous strings tied to the main cables. The horizontal bridge roadbed lies exactly one meter below the lowest point of the ropes. For aesthetic reasons, the roadbed should be at least two meters below the lower edge of the lower of the two students' windows. The laws of physics dictate that each suspension rope forms a parabola.

While Jan and Tereza don't plan to walk on this model bridge, there is a serious problem: some of the occupants of the apartment buildings own pet cats, and others own pet birds. Jan and Tereza want to be sure that their bridge doesn't provide a way for a cat to reach a bird. Jan and Tereza have observed that a cat cannot jump as high as 0.5 meters, and will not jump down as far as 3 meters. So as long as the bridge roadbed lies at least 0.5 meters above the bottom of a cat's window, or at least 3 meters below the bottom of a cat's window, the cat will not jump onto it. Likewise, a cat that successfully jumps onto the roadbed will not be able to reach a bird's window if the roadbed lies at least 0.5 meters below the bottom of the bird's window, or at least 3 meters above the bottom of the bird's window. Cats are concerned only with reaching birds, and they do not worry about returning home.

The figure below shows Jan's apartment ("J") and Tereza's apartment ("T") with a rope joining the bottoms of their windows and the cardboard roadbed one meter below the lowest point of the rope. The cat on the second floor can reach the bird on the second floor using the bridge.



You must write a program to determine how much rope Jan and Tereza need to construct each cable for a bridge that won't endanger any of the birds in their two buildings.

Input for your program will be: the distance between the two buildings, in meters; the floor numbers for Jan and Tereza (with the lowest, or ground floor in each building numbered 1), the kinds of pets living in all the floors up through Jan's floor, and the kinds of pets living in all the floors up through Tereza's floor. Your program must determine the length of the longest cable that can be used to suspend a bridge between the two buildings that does not permit any cat to reach a bird by means of the bridge. The roadbed of the bridge must lie at least 1 meter above the ground and must lie exactly one meter below the lowest point of the suspension cables. It must

also lie at least two meters below the lower of the two windows of Jan and Tereza. All rooms in the buildings are exactly 3 meters tall; all windows are exactly 1.5 meters tall and the bottom of each window lies exactly 1 meter above the floor of each room.

Input

The input will describe several cases, each of which has three lines. The first line will contain two positive integers j and t ($2 \le j, t \le 25$) representing Jan's floor and Tereza's floor, and a real value d ($1 \le d \le 25$) representing the distance, in meters, between the buildings. The second line will contain j uppercase letters ℓ_1, ℓ_2, ..., ℓ_j separated by whitespace. Letter ℓ_k is *B* if a bird lives on floor number k of Jan's building, *C* if a cat lives on floor number k, and *N* if neither kind of pet lives on floor number k. The third line similarly contains t uppercase letters representing the same kind of information for the floors in Tereza's building. The last case is followed by a line containing three zeroes.

Output

For each case, print the case number (1, 2, …) and the largest value c such that two cables, each of length c, can be used to suspend a bridge from the lower edges of Jan's and Tereza's windows so that the bridge floor lies one meter below the lowest point in the cable, lies at least 1 meter above the ground, lies at least two meters below Jan and Tereza's windows, and does not allow a cat to reach a bird. The length should be rounded to three places following the decimal point. If no such bridge can be constructed, print "impossible." Print a blank line between the output for consecutive cases. Your output format should imitate the sample output.

Sample Input	Output for the Sample Input
4 3 5.0 N C N C N B B 4 3 5.0 C B C C B C B 0 0 0	Case 1: 14.377 Case 2: impossible

The 2004 28th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem J

Air Traffic Control

Input File: traffic.in

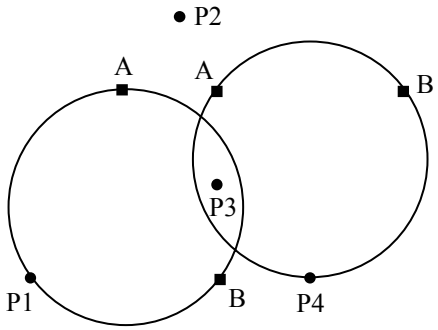
In order to avoid midair collisions, most commercial flights are monitored by ground-based air traffic control centers that track their position using radar. For this problem, you will be given information on a set of airplanes and a set of control centers, and you must compute how monitoring of the airplanes is distributed among the control centers. The position of each airplane is represented by a unique (x, y) coordinate pair. For the purpose of this problem, the height (altitude) of the airplanes can be ignored.

The number of airplanes that can be monitored by a given control center varies from time to time due to changes in staff and equipment. At any given time, each control center monitors as many planes as it can, choosing the airplanes to be monitored according to the following priorities: (1) it will prefer to monitor planes that are closer to the control center rather than ones that are farther away; (2) if two airplanes are equally distant from the center and the center can monitor only one of them, it will choose the one that is farther to the north (positive y-axis); (3) if two airplanes are equally distant and have the same y-coordinate, the center will give preference to the airplane that is farther to the east (positive x-axis).

At any given moment, each control center has a circular "span of control" whose radius is the distance to the farthest airplane being monitored by the control center. All airplanes inside the span of control are monitored by the control center. Airplanes on the boundary of the span of control may or may not be monitored by the control center, depending on its capacity and on the priorities listed above.

You will not be given the positions of the control centers. Instead, for each control center, you will be given the number of airplanes that it is currently monitoring, and two points that are on the boundary of its current span of control. With this information, you can compute the position of the control center and decide which airplanes it is monitoring. If the data is consistent with more than one possible span of control, you should choose the span that includes the airplane that is farthest to the north, breaking ties by choosing the airplane that is farthest to the north then to the east.

The figure below, which shows four airplanes and two control centers, illustrates the problem. Each control center is represented by a circular span of control and by two points on the boundary of this span, labeled *A* and *B*. *P*1, *P*2, *P*3, and *P*4 label the four airplanes. In this example, airplanes *P*1 and *P*4 are each being monitored by a single control center, airplane *P*3 is being monitored by two control centers, and airplane *P*2 is not being monitored by either control center.



Input

The input consists of several trial data sets. The first line of input in each trial data set contains two integers *NP* ($0 < NP < 100$) and *NC* ($0 < NC < 10$), which represent the number of airplanes and the number of control centers, respectively. Each of the next *NP* lines contains two floating-point numbers that represent the (x, y)

coordinates of one airplane. Each of the next *NC* lines describes one control center. Each contains an integer between 0 and *NP* (inclusive) indicating the number of airplanes monitored by the control center, followed by two pairs of floating point numbers that represent the (*x*, *y*) coordinates of two points on the boundary of its span of control (neither of which is the position of an airplane). If two distances differ by less than 0.00001, you should treat them as the same distance.

The last data set is followed by a line containing two zeros.

Output

For each trial, compute the number of airplanes that are monitored by zero control centers, the number of airplanes that are monitored by one control center, and so on up to the number of airplanes that are monitored by *NC* control centers. Print the trial number followed by a sequence of *NC* + 1 integers, where the i^{th} integer in the sequence represents the number of airplanes that are monitored by *i*-1 control centers. If data for one of the control centers is inconsistent, print "Impossible" instead of the sequence of integers for that trial. Use the format shown in the example output, and print a blank line after each trial.

Sample Input	Output for the Sample Input
4 2 3.0 0.0 0.0 0.0 1.6 2.8 2.0 1.0 2 1.0 2.0 2.0 0.0 2 2.0 2.0 4.0 2.0 2 1 0.0 0.5 0.0 -0.5 0 -1.0 0.0 1.0 0.0 0 0	Trial 1: 1 2 1 Trial 2: Impossible