The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem A

Kingdom of Magic

Input File: kingdom.in

Kingdom of Magic has a network of bidirectional magic portals between cities since ancient times. Each portal magically connects a pair of cities and allows fast magical communication and travel between them. Cities that are connected by the magic portal are called neighboring.

Prince Albert and Princess Betty are living in the neighboring cities. Since their childhood Albert and Betty were always in touch with each other using magic communication Orbs, which work via a magic portal between the cities.

Albert and Betty are in love with each other. Their love is so great that they cannot live a minute without each other. They always carry the Orbs with them, so that they can talk to each other at any time. There is something strange about their love — they have never seen each other and they even fear to be in the same city at the same time. People say that the magic of the Orbs have affected them.

Traveling through the Kingdom is a complicated affair for Albert and Betty. They have to travel through magic portals, which is somewhat expensive even for royal families. They can simultaneously use a pair of the portals to move to a different pair of cities, or just one of them can use a portal, while the other one stays where he or she is. At any moment of their travel they have to be in a neighboring cities. They cannot simultaneously move through the same portal.

Write a program that helps Albert and Betty travel from one pair of the cities to another pair. It has to find the cheapest travel plan — with the minimal number of times they have to move though the magic portals. When they move through the portals simultaneously it counts as two moves.

Input

The first line of the input file contains the number of test cases.

The first line of each case contains integer numbers n, m, a_1, b_1, a_2, b_2. Here n ($3 \le n \le 100$) is a number of cities in the Kingdom (cities are numbered from 1 to n); m ($2 \le m \le 1000$) is a number of magic portals; a_1, b_1 ($1 \le a_1, b_1 \le n$, $a_1 \ne b1$) are the neighboring cities where Albert and Betty correspondingly start their travel from; a_2, b_2 ($1 \le a_2, b_2 \le n$, $a_2 \ne b_2$) are the neighboring cities where Albert and Betty correspondingly want to get to ($a_1 \ne a_2$ or $b_1 \ne b2$).

Following m lines describe the portals. Each line contains two numbers p_{i1} and p_{i2} ($1 \le p_{i1}, p_{i2} \le n$, $p_{i1} \ne p_{i2}$) representing cities that are connected by the portal. There is at most one portal connecting two cities.

Output

For each case, on the first line write two numbers c and k. Here c is the minimal number of moves in the travel plan; k is the number of neighboring city pairs that Albert and Betty visit during their travel including a_1, b_1 at the start and a_2, b_2 at the end.

Then write k lines with two integer numbers a_i and b_i on each line — consecutive different pairs of neighboring cities that Albert and Betty visit during their travel. If there are multiple travel plans with the same number of moves, then write any of them. It is guaranteed that solution exists.

Write a blank line between two consecutive cases.

Sample Input	Output for the Sample Input
1 4 5 1 2 2 1 1 2 2 3 3 4 4 1 1 3	3 3 1 2 2 3 2 1

The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem B

Art of War

Input File: art.in

The *Warring States Period* (473–221 BC) refers to the centuries of turmoil following the Spring and Autumn Period. China was divided into many little kingdoms that were constantly fighting with each other. Unlike in previous ages, when chivalry played an important role in battles and the states fought mostly for balance of power or to resolve disputes, in this period the aim of battle was to conquer and completely annihilate the other states. Eventually seven states, known as the "Seven Great Powers" rose to prominence: Qi, Chu, Yan, Han, Zhao, Wei, and Qin. After numerous alliances and counter-alliances, Qin defeated all the other states one by one, putting an end to the Warring States Period.

You are given a map that shows the position of the capital for each state, and the borders between the states as a series of line segments. Your job is to determine which states were fighting with each other. This is pretty easy to determine — if two states had a common border, then they were fighting.

Input

The input contains several blocks of test cases. Each case begins with a line containing two integers: the number $1 \le n \le 600$ of states, and the number $1 \le m \le 4000$ of border segments. The next n lines describe the coordinates of capitals, there are two integers in each line. The next m lines after that describe the m border segments. Each line contains four integers x_1, y_1, x_2 and y_2, meaning that there is a border segment from (x_1, y_1) to (x_2, y_2). It is not given in the input what the two states on the two sides of the border are, but it can be deduced from the way the borders go.

Each state is enclosed by a continuous borderline. The states are surrounded by an infinite wasteland, thus a border segment either separates two states, or a state from the wasteland. It is not possible that the same state is on both sides of a border segment, or the wasteland is on both sides of a border segment. There is exactly one capital in each state, and there is no capital in the wasteland. The border segments do not cross each other, they can meet only at the end points.

The input is terminated by a block with $n = m = 0$.

Output

For each test case, you have to output n lines that describe the enemies of the n states (recall that if two states share a border, then they are enemies). Each line begins with an integer, the number x of enemies the given statet has. This number is followed by x numbers identifying the enemies of the state. These numbers are between 1 and n and number 1 refers to the first capital appearing in the input, number n refers to the last.

Sample Input	Output for the Sample Input
4 12 3 2 11 8 12 17 1 19 0 0 10 0 10 0 20 0 20 0 20 10 20 10 20 20 10 20 0 20 0 20 0 10 0 10 0 0 10 0 10 10 0 10 10 10 20 10 10 10 10 20 10 10 4 16 170 13 24 88 152 49 110 130 60 60 140 60 140 60 140 140 140 140 60 140 60 140 60 60 0 0 200 0 200 0 200 200 200 200 0 200 0 200 0 0 40 40 160 40 160 40 160 160 160 160 40 160 40 160 40 40 20 20 180 20 180 20 180 180 180 180 20 180 20 180 20 20 0 0	2 2 4 2 1 3 2 2 4 2 1 3 1 2 2 1 3 2 2 4 1 3

The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice

on Winter Camp

Problem C

Insecure in Prague

Input File: 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 k-th 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	PRAGUE
ABABABAB	Codeword not unique
THEACMPROGRAMMINGCONTEST	Codeword not unique

The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem D

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 2005 ACM Programming Contest Short Practice on Winter Camp

The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem E

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----|
    +-----+
```

The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem F

Contracting Integers

Input File: integers.in

You are given a sequence of n positive integers $a = [a_1, a_2, \ldots, a_n]$, on which you can perform contraction operations. One *contraction* operation consists of replacing adjacent elements a_i and a_{i+1} by their difference $a_i - a_{i+1}$. More formally, let $\text{con}(a, i)$ denote the $(n-1)$-element sequence obtained from sequence a by the contraction described above, involving elements a_i and a_{i+1}. That is:

$$\text{con}(a, i) = [a_1, \ldots, a_{i-1}, a_i - a_{i+1}, a_{i+2}, \ldots, a_n]$$

Applying $n-1$ contractions to any given sequence of n integers obviously yields a single integer. For example, applying contractions 2, 3, 2 and 1, in that order, to the sequence [12,10,4,3,5] yields the integer 4, since:

$$\begin{aligned} \text{con}([12, 10, 4, 3, 5], 2) &= [12, 6, 3, 5] \\ \text{con}([12, 6, 3, 5], 3) &= [12, 6, -2] \\ \text{con}([12, 6, -2], 2) &= [12, 8] \\ \text{con}([12, 8], 1) &= [4] \end{aligned}$$

Given a sequence $a_1, a_2, \ldots, a_n$ and a target number T, the problem is to find a sequence of $n-1$ contractions which, when applied to the original sequence, yields T. We assume that there is always at least one such sequence of contractions yielding T.

Input

The input file may contain several instances of the problem. Each instance is given as follows:

1. The first line contains two integers, separated by an arbitary number of blank spaces: the integer n, $1 \leq n \leq 100$ (the number of integers in the original sequence), and the target integer T, $-10000 \leq T \leq 10000$.
2. The following n lines contain the starting sequence, i.e., a sequence of n integers a_i, one per line, where $1 \leq a_i \leq 100$.

The input file is terminated by a line containing the integers `0 0`.

Output

For each instance of the problem, your program should print $n-1$ lines, describing a sequence of contractions that transforms the starting sequence into the number T corresponding to that instance. Each of these $n-1$ lines contains just an integer i, denoting the i-th contraction to be applied.

You can assume that at least one such sequence of contractions will exist for a given input. Each instance should be terminated with a blank line.

Sample Input	Output for the Sample Input
4 5 10 2 5 2 5 4 12 10 4 3 5	1 2 1 2 3 2 1

The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem G

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 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem H

Lord of the Ring

Input File: ring.in

Frodo must accomplish a noble and difficult mission, he must destroy a magic and wicked ring. In this quest, he must travel to a dangerous place called Modor and throw the ring into a crevice of fire. He has left home for some time and is currently following a straight and quite long road that has bushes from place to place. Being very tired Frodo thinks he would better have some rest. The only safe place along the road is a bush the position of which can be computed using a magic formula that uses the value *P* that is the product of the distances between pairs of adjacent bushes along the road. Unfortunately what Frodo knows are only the distances between every pair of bushes along the road and the magic formula, but he doesn't know the value of *P*. Can you help him in this respect?

Input

The program input is from a text file. Each data set in the file stands for a particular set of distances between pairs of bushes on the road Frodo is traveling along. Each data set starts with the number of distances followed by the distances in nondecreasing order. White spaces can occur freely in the input.

Output

For each set of data the program computes the value of *P* to the standard output from the beginning of a separate line. If *P* cannot be computed from the data set the output is "`No solution`".

It is known that there are at least two bushes and at most 1000 bushes along the road. Moreover, the value of *P* cannot exceed 10^9.

Sample Input	Output for the Sample Input
6 1 2 2 3 3 5 3 1 2 2	4 No solution

The 2005 ACM Programming Contest Short Practice on Winter Camp

The 2005 29th Annual **acm** International Collegiate

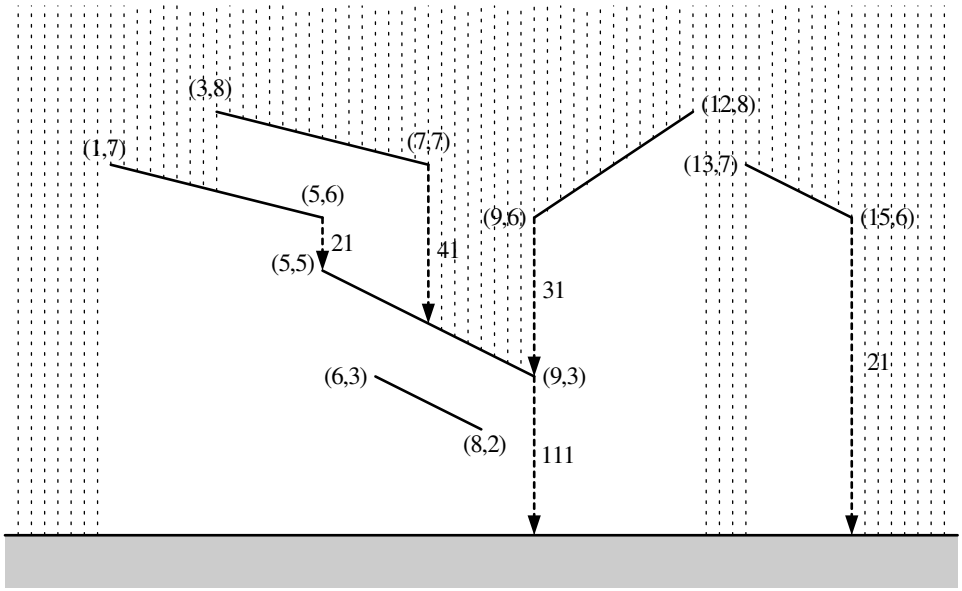
Programming Contest Short Practice on Winter Camp

Problem I

November rain

Input File: rain.in

Contemporary buildings can have very complicated roofs. If we take a vertical section of such a roof it results in a number of sloping segments. When it is raining the drops are falling down on the roof straight from the sky above. Some segments are completely exposed to the rain but there may be some segments partially or even completely shielded by other segments. All the water falling onto a segment as a stream straight down from the lower end of the segment on the ground or possibly onto some other segment. In particular, if a stream of water is falling on an end of a segment then we consider it to be collected by this segment.



For the purpose of designing a piping system it is desired to compute how much water is down from each segment of the roof. To be prepared for a heavy November rain you should count one liter of rain water falling on a meter of the horizontal plane during one second.

Your task is to write a program that reads the description of a roof, computes the amount of water down in one second from each segment of the roof, and writes the results.

Input

The input contains multiple data sets.

The first line of a data set contains one integer n ($1 \le n \le 40000$) being the number of segments of the roof. Each of the next n lines describes one segment of the roof and contains four integers x_1, y_1, x_2, y_2 ($0 \le x_1, y_1, x_2, y_2 \le 1000000$, $x_1 < x_2$, $y_1 \ne y_2$) separated by single spaces. Integers x_1, y_1 are respectively the horizontal position and the height of the left end of the segment. Integers x_2, y_2 are respectively the horizontal position and the height of the right end of the segment. The segments don't have common points and there are no horizontal segments. You can also assume that there are at most 25 segments placed above any point on the ground level.

The input is terminated by $n = 0$.

Output

The output for a data set consists of *n* lines. The *i*-th line should contain the amount of water (in liters) down from the *i*-th segment of the roof in one second. Print a blank line between data sets.

Sample Input	Output for the Sample Input
6 13 7 15 6 3 8 7 7 1 7 5 6 5 5 9 3 6 3 8 2 9 6 12 8 0	2 4 2 11 0 3

The 2005 29th Annual **acm** International Collegiate

Programming Contest Short Practice on Winter Camp

Problem J

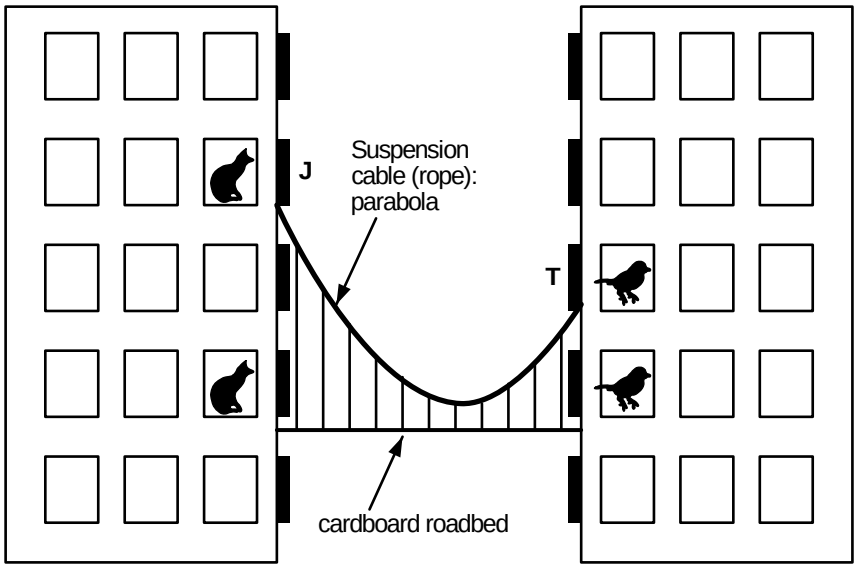
Suspense!

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 ($2 \leq j, t \leq 25$) representing Jan's floor and Tereza's floor, and a real value d ($1 \leq d \leq 25$) 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_j 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 5 3 4.5 B B N N B N C C	Case 1: 14.377 Case 2: impossible