Problem A: Finding Nemo

Input File: nemo.in

Nemo is a naughty boy. One day he went into the deep sea all by himself. Unfortunately, he became lost and couldn't find his way home. Therefore, he sent a signal to his father, Marlin, to ask for help.

After checking the map, Marlin found that the sea is like a labyrinth with walls and doors. All the walls are parallel to the X-axis or to the Y-axis. The thickness of the walls are assumed to be zero. All the doors are opened on the walls and have a length of 1. Marlin cannot go through a wall unless there is a door on the wall. Because going through a door is dangerous (there may be some virulent medusas near the doors), Marlin wants to go through as few doors as he could to find Nemo.

Figure-1 shows an example of the labyrinth and the path Marlin went through to find Nemo.

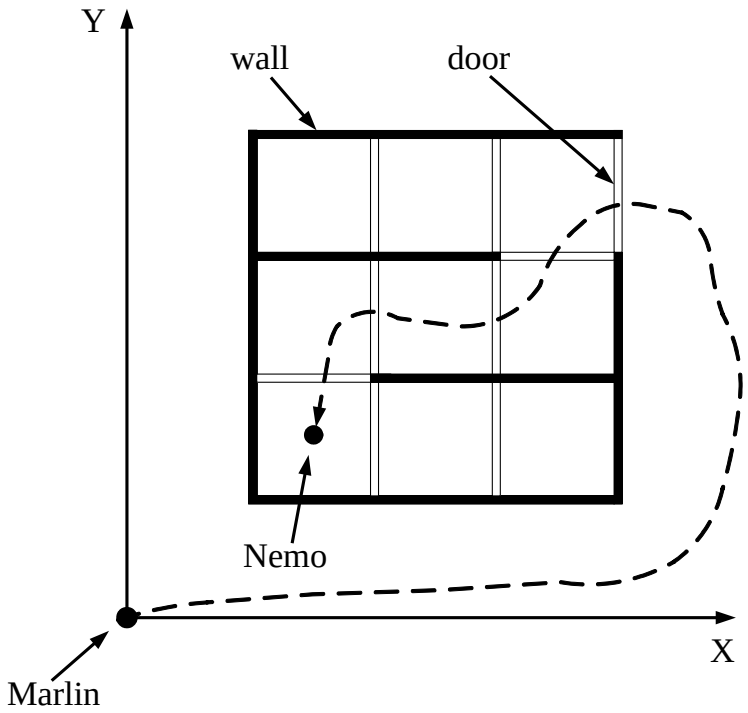

Figure-1. Labyrinth and Path

We assume Marlin's initial position is at (0, 0). Given the position of Nemo and the configuration of walls and doors, please write a program to calculate the minimum number of doors Marlin has to go through in order to reach Nemo.

Input

The input consists of several test cases. Each test case is started by two non-negative integers *M* and *N*. *M* represents the number of walls in the labyrinth and *N* represents the number of doors. Then follow *M* lines, each containing four integers that describe a wall in the following format:

x y d t

(*x*, *y*) indicates the lower-left point of the wall, *d* is the direction of the wall — 0 means it's parallel to the X-axis and 1 means that it's parallel to the Y-axis, and *t* gives the length of the wall. The coordinates of two ends of any wall will be in the range of [1,199].

Then there are *N* lines that give the description of the doors:

x y d

x, *y*, *d* have the same meaning as the walls. As the doors have fixed length of 1, *t* is omitted.

The last line of each case contains two positive float numbers:

f_1 f_2

(f_1, f_2) gives the position of Nemo. And it will not lie within any wall or door.

A test case of *M* = -1 and *N* = -1 indicates the end of input, and should not be processed.

Output

For each test case, in a separate line, please output the minimum number of doors Marlin has to go through in order to rescue his son. If he can't reach Nemo, output -1.

Sample Input	Output for the Sample Input
8 9 1 1 1 3 2 1 1 3 3 1 1 3 4 1 1 3 1 1 0 3 1 2 0 3 1 3 0 3 1 4 0 3 2 1 1 2 2 1 2 3 1 3 1 1 3 2 1 3 3 1 1 2 0 3 3 0 4 3 1 1.5 1.5 4 0 1 1 0 1 1 1 1 1 2 1 1 1 1 2 0 1 1.5 1.7 -1 -1	5 -1

Problem B: Searching the Web

Input File: web.in

The word "search engine" may not be strange to you. Generally speaking, a search engine searches the web pages available in the Internet, extracts and organizes the information and responds to users' queries with the most relevant pages. World famous search engines, like GOOGLE, have become very important tools for us to use when we visit the web. Such conversations are now common in our daily life:

"What does the word like ****** mean?"
"Um… I am not sure, just google it."

In this problem, you are required to construct a small search engine. Sounds impossible, does it? Don't worry, here is a tutorial teaching you how to organize large collection of texts efficiently and respond to queries quickly step by step. You don't need to worry about the fetching process of web pages, all the web pages are provided to you in text format as the input data. Besides, a lot of queries are also provided to validate your system.

Modern search engines use a technique called *inversion* for dealing with very large sets of documents. The method relies on the construction of a data structure, called an *inverted index*, which associates *terms* (words) to their occurrences in the collection of documents. The set of terms of interest is called the *vocabulary*, denoted as *V*. In its simplest form, an inverted index is a dictionary where each search key is a term $\omega \in V$. The associated value $b(\omega)$ is a pointer to an additional intermediate data structure, called a *bucket*. The bucket associated with a certain term ω is essentially a list of pointers marking all the occurrences of ω in the text collection. Each entry in each bucket simply consists of the *document identifier (DID)*, the ordinal number of the document within the collection and the ordinal line number of the term's occurrence within the document.

Let's take Figure-1 for an example, which describes the general structure. Assuming that we only have three documents to handle, shown at the right part in Figure-1; first we need to tokenize the text for words (blank, punctuations and other non-alphabetic characters are used to separate words) and construct our vocabulary from terms occurring in the documents. For simplicity, we don't need to consider any phrases, only a single word as a term. Furthermore, the terms are case-insensitive (e.g. we consider "book" and "Book" to be the same term) and we don't consider any morphological variants (e.g. we consider "books" and "book", "protected" and "protect" to be different terms) and hyphenated words (e.g. "middle-class" is not a single term, but separated into 2 terms "middle" and "class" by the hyphen). The vocabulary is shown at the left part in Figure-1. Each term of the vocabulary has a pointer to its bucket. The collection of the buckets is shown at the middle part in Figure-1. Each item in a bucket records the DID of the term's occurrence.

After constructing the whole inverted index structure, we may apply it to the queries. The query is in any of the following formats:

term
term AND *term*
term OR *term*
NOT *term*

A single term can be combined by Boolean operators: AND, OR and NOT ("*term1* AND *term2*" means to query the documents including term1 and term2; "*term1* OR *term2*" means to query the documents including term1 or term2; "NOT *term1*" means to query the documents not including term1). Terms are single words as defined above. You are guaranteed that no non-alphabetic characters appear in a term, and all the terms are in lowercase. Furthermore, some meaningless stop words (common words such as articles, prepositions, and adverbs, specified to be "the, a, to, and, or, not" in our problem) will not appear in the query, either.

For each query, the engine based on the constructed inverted index searches the term in the vocabulary, compares the terms' bucket information, and then gives the result to user. Now can you construct the engine?

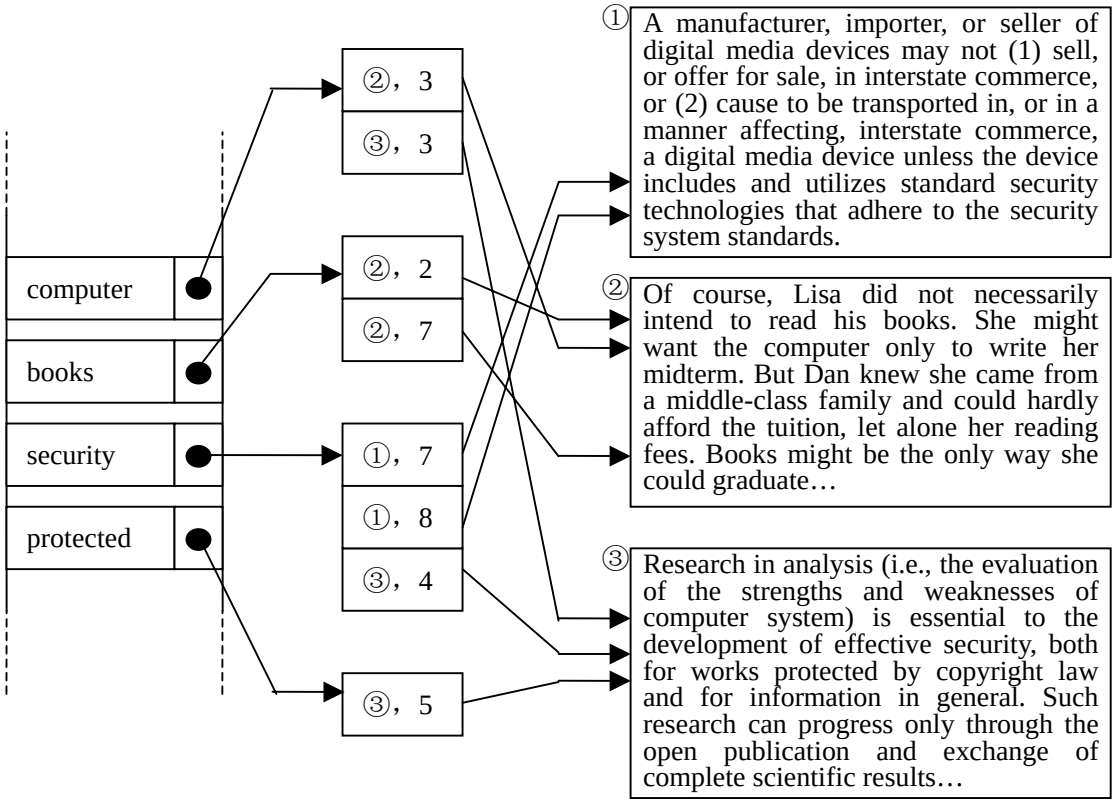

Figure-1. Inverted Index

Input

The input starts with integer *N* (0 < *N* < 100) representing *N* documents provided. Then the next *N* sections are *N* documents. Each section contains the document content and ends with a single line of ten asterisks.

****

You may assume that each line contains no more than 80 characters and the total number of lines in the *N* documents will not exceed 1500.

Next, integer *M* (0 < *M* <= 50000) is given representing the number of queries, followed by *M* lines, each query in one line. All the queries correspond to the format described above.

Output

For each query, you need to find the document satisfying the query, and output just the lines within the documents that include the search term (For a NOT query, you need to output the whole document). You should print the lines in the same order as they appear in the input. Separate different documents with a single line of 10 dashes.

If no documents matching the query are found, just output a single line: “Sorry, I found nothing.” The output of each query ends with a single line of 10 equal signs.

==========

Sample Input	Output for the Sample Input
4 A manufacturer, importer, or seller of digital media devices may not (1) sell, or offer for sale, in interstate commerce, or (2) cause to be transported in, or in a manner affecting, interstate commerce, a digital media device unless the device includes and utilizes standard security technologies that adhere to the security system standards. ********** Of course, Lisa did not necessarily intend to read his books. She might want the computer only to write her midterm. But Dan knew she came from a middle-class family and could hardly afford the tuition, let alone her reading fees. Books might be the only way she could graduate ********** Research in analysis (i.e., the evaluation of the strengths and weaknesses of computer system) is essential to the development of effective security, both for works protected by copyright law and for information in general. Such research can progress only through the open publication and exchange of complete scientific results ********** I am very very very happy! What about you? ********** 6 computer books AND computer books OR protected NOT security very slick	want the computer only to write her ---------- computer system) is essential to the ========== intend to read his books. She might want the computer only to write her fees. Books might be the only way she ========== intend to read his books. She might fees. Books might be the only way she ---------- for works protected by copyright law ========== Of course, Lisa did not necessarily intend to read his books. She might want the computer only to write her midterm. But Dan knew she came from a middle-class family and could hardly afford the tuition, let alone her reading fees. Books might be the only way she could graduate ---------- I am very very very happy! What about you? ========== I am very very very happy! ========== Sorry, I found nothing. ==========

Problem C: Argus

Input File: argus.in

A data stream is a real-time, continuous, ordered sequence of items. Some examples include sensor data, Internet traffic, financial tickers, on-line auctions, and transaction logs such as Web usage logs and telephone call records. Likewise, queries over streams run continuously over a period of time and incrementally return new results as new data arrives. For example, a temperature detection system of a factory warehouse may run queries like the following.

> Query-1: "Every five minutes, retrieve the maximum temperature over the past five minutes."
> Query-2: "Return the average temperature measured on each floor over the past 10 minutes."

We have developed a Data Stream Management System called *Argus*, which processes the queries over the data streams. Users can register queries to the Argus. Argus will keep the queries running over the changing data and return the results to the corresponding user with the desired frequency.

For the Argus, we use the following instruction to register a query:

> Register *Q_num Period*

Q_num (0 < *Q_num* <= 3000) is query ID-number, and *Period* (0 < *Period* <= 3000) is the interval between two consecutive returns of the result. After *Period* seconds of register, the result will be returned for the first time, and after that, the result will be returned every *Period* seconds.

Here we have several different queries registered in Argus at once. It is confirmed that all the queries have different *Q_num*. Your task is to tell the first *K* queries to return the results. If two or more queries are to return the results at the same time, they will return the results one by one in the ascending order of *Q_num*.

Input

The first part of the input are the register instructions to Argus, one instruction per line. You can assume the number of the instructions will not exceed 1000, and all these instructions are executed at the same time. This part is ended with a line of "#".

The second part is your task. This part contains only one line, which is one positive integer *K* (<= 10000).

Output

You should output the *Q_num* of the first *K* queries to return the results, one number per line.

Sample Input	Output for the Sample Input
Register 2004 200 Register 2005 300 # 5	2004 2005 2004 2004 2005

Problem D: Fun Game

Input File: fun.in

A few kids are standing around an old tree playing a game. The tree is so huge that each kid can only see the kids close to him/her.

The game consists many 'turns'. At the beginning of each turn of the game, a piece of paper is given to a randomly chosen kid. This kid writes the letter "B" if he is a boy or the letter "G" if a girl. Then he chooses a direction to pass the paper (clockwise or counter-clockwise), and gives the paper to his neighbor in that direction. The kid getting the paper writes down his sex too, and gives the paper to his neighbor in the same direction. In this way, the paper goes through the kids one by one, until one kid stops passing the paper and announces the end of this turn.

For example, there are five kids around the tree, and their genders are shown in Figure-1. The paper first goes to Kid1, after writing a "B" he passes it to Kid2, and Kid2 to Kid3. After Kid3 writes down a "G", she ends up this turn, and we get the paper with a string "BBG".

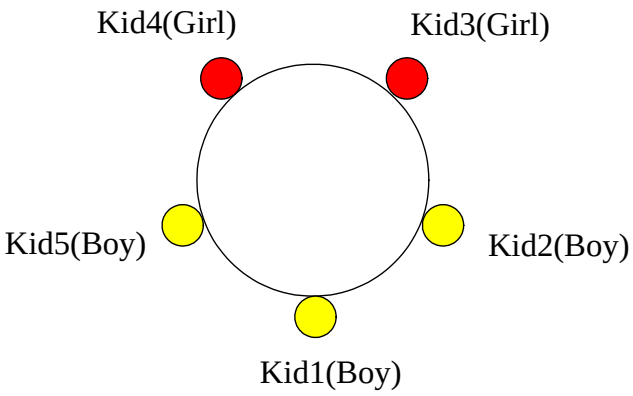

Figure-1. Five kids around the tree

After *N* turns, we get *N* pieces of paper with strings of "B"s and/or "G"s. One of the kids will get all these papers, and has to figure out at least how many kids are around the tree playing the game. It's known that there are at least two kids. Please write a program to help him.

Input

There are several test cases. Each case starts with a line containing an integer *N*, the number of papers ($2 <= N <= 16$). Each of the following *N* lines contains a string on a paper, which is a nonempty string of letter "B"s and/or "G"s. Each string has no more than 100 letters.

A test case of $N = 0$ indicates the end of input, and should not be processed.

Output

For each test case, output the least possible number of kids in a line.

Sample Input

```
3
BGGB
BGBGG
GGGBGB
2
BGGGBBBGG
GBBBG
0
```

Output for the Sample Input

```
9
6
```

Problem E: Square

Input File: square.in

Given a square at [0, 1] * [0, 1] that has N points (P_1, P_2, ..., P_N) in the square (you may assume that different points can be at the same position), we can connect the N points and the four corners of the square with some line segments so that through these segments any two of the N+4 points can reach each other (directly or indirectly). The *graph length* is defined as the total length of the line segments. When N points' positions are fixed, there must exist a way of connecting them, such that it will make the shortest graph length. We can use $LEN\ (P_1, P_2, ..., P_N)$ to record the graph length using this way of connecting.

In this situation, $LEN\ (P_1, P_2, ..., P_N)$ is a function of $P_1, P_2, ..., P_N$. When $P_1, P_2, ..., P_N$ change their positions, $LEN\ (P_1, P_2, ..., P_N)$ also changes. It's easy to prove that there exist some P_1', P_2', ..., P_N' in the square such that $LEN\ (P_1', P_2', ..., P_N')$ is at its minimum.

Given the initial positions of N points, your task is to find out N points $P_1"$, $P_2"$, ..., $P_N"$ in the square such that $|P_1P_1"| + |P_2P_2"| + ... + |P_NP_N"|$ is minimum and $LEN\ (P_1", P_2", ..., P_N") = LEN\ (P_1', P_2', ..., P_N')$. You are requested to output the value of $|P_1P_1"| + |P_2P_2"| + ... + |P_NP_N"|$, where $|P_iP_i"|$ is the distance between P_i and $P_i"$.

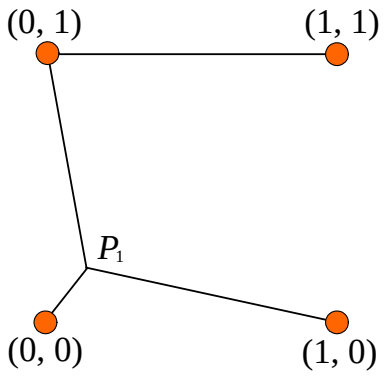

Figure-1

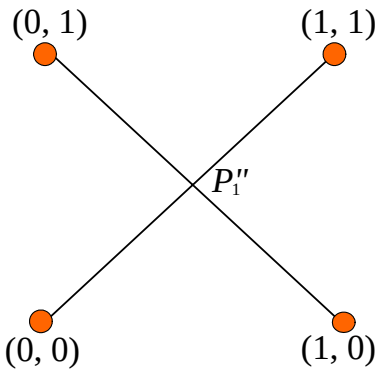

Figure-2

For example, Figure-1 gives the initial position of P_1 and the way of connecting to obtain $LEN\ (P_1)$. In Figure-2, it gives the position of $P_1"$, which is at the center of the square, and the way of connecting to obtain $LEN\ (P_1")$. It can be proved that $LEN\ (P_1") = LEN\ (P_1')$; your job is to output the distance between P_1 and $P_1"$.

Input

The input consists of several test cases. For each test case, the first line consists of one integer N ($1 <= N <= 100$), the number of points, and N lines follow to give the coordinates for every point in the following format:

x y

Here, *x* and *y* are float numbers within the value [0, 1].

A test case of $N = 0$ indicates the end of input, and should not be processed.

Output

For each test case, output the value of $|P_1P_1''| + |P_2P_2''| + ... + |P_NP_N''|$. The value should be rounded to three digits after the decimal point.

Sample Input	Output for the Sample Input
1 0.2 0.5 2 0 0.5 0.5 0.5 0	0.300 0.500

Problem F: Color a Tree

Input File: color.in

Bob is very interested in the data structure of a tree. A tree is a directed graph in which a special node is singled out, called the "root" of the tree, and there is a unique path from the root to each of the other nodes.

Bob intends to color all the nodes of a tree with a pen. A tree has *N* nodes, these nodes are numbered 1, 2, ..., *N*. Suppose coloring a node takes 1 unit of time, and after finishing coloring one node, he is allowed to color another. Additionally, he is allowed to color a node only when its father node has been colored. Obviously, Bob is only allowed to color the root in the first try.

Each node has a “coloring cost factor”, C_i. The coloring cost of each node depends both on C_i and the time at which Bob finishes the coloring of this node. At the beginning, the time is set to 0. If the finishing time of coloring node *i* is F_i, then the coloring cost of node *i* is $C_i * F_i$.

For example, a tree with five nodes is shown in Figure-1. The coloring cost factors of each node are 1, 2, 1, 2 and 4. Bob can color the tree in the order 1, 3, 5, 2, 4, with the minimum total coloring cost of 33.

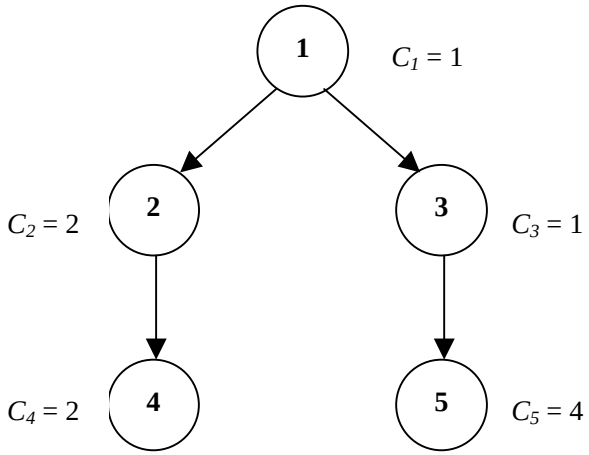

Figure-1. A tree with five nodes

Given a tree and the coloring cost factor of each node, please help Bob to find the minimum possible total coloring cost for coloring all the nodes.

Input

The input consists of several test cases. The first line of each case contains two integers *N* and *R* ($1 <= N <= 1000$, $1 <= R <= N$), where *N* is the number of nodes in the tree and *R* is the node number of the root node. The second line contains *N* integers, the *i*-th of which is C_i ($1 <= C_i <=$

500), the coloring cost factor of node *i*. Each of the next *N*-1 lines contains two space-separated node numbers V_1 and V_2, which are the endpoints of an edge in the tree, denoting that V_1 is the father node of V_2. No edge will be listed twice, and all edges will be listed.

A test case of *N* = 0 and *R* = 0 indicates the end of input, and should not be processed.

Output

For each test case, output a line containing the minimum total coloring cost required for Bob to color all the nodes.

Sample Input

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

Output for the Sample Input

```
33
```

Problem G: Kid's Problem

Input File: kid.in

Kid is a famous thief and he is known for his unique habit. Before his attempt, he will always inform the person who he is going to rob beforehand. Though the people have paid much attention to him, he has never failed. This time Kid informed a billionaire, Jack, that he is going to enter Jack's home to take away his expensive treasure. Jack is very afraid and he asked a brilliant boy, Conan, to help him. Conan designed a special lock for Jack's home. However, Kid is very tricky and he stole the structure map and the password of the lock.

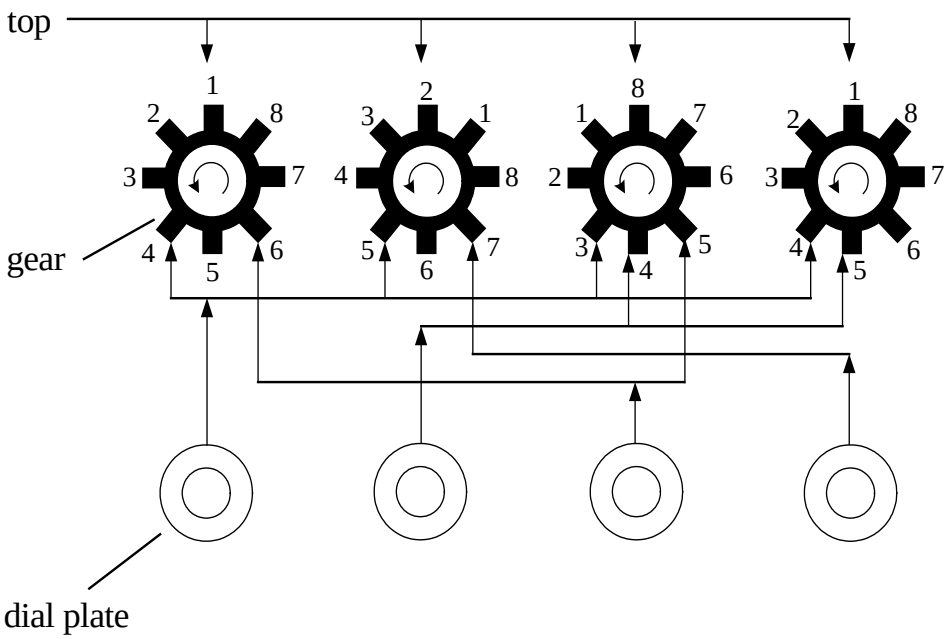

Figure-1

From the structure map (see Figure-1), Kid knows that the lock contains *K* dial plates and *K* gears, and every dial plate controls several gears. There are *N* teeth on each gear, which is numbered counter-clockwise from 1 to *N*. When a certain dial plate is dialed, the gears that refer to the plate will rotate counter-clockwise by several teeth (different gears may rotate different numbers of teeth). A dial plate can be dialed more than once. At the beginning, the numbers of the top teeth of the gears are all "1". To open the lock, the number of the top tooth of every gear must be a certain number. These *K* numbers form the password. Take Figure-1 for an example where *N*=8, *K*=4; if the password is 1-2-8-1, the lock will open.

With the password in hand, Kid wants to know whether he can open the lock; and if he can, he wants to know the least number of dials he has to use to open the lock. You may assume that the lock is always locked at the beginning, which means that the password cannot be *K* "1"s.

Input

There are several test cases. In the first line of each case there are two integers K ($1 <= K <= 20$) and N ($2 <= N <= 10$). Then follows a line with K integers expressing the password. Each number in the password is between 1 and N. Then come K lines, the i-th of which describes how the i-th dial plate controls the referred gears. These K lines have the following format:

$p\ a_1\ b_1\ a_2\ b_2\ ...\ a_p\ b_p$

Integer p ($0 <= p <= K$) expresses the number of gears that are referred to the dial plate. a_i ($1 <= i <= p$) is an integer between 1 and K which tells that the a_i-th gear is under the control of this dial plate. b_i is an integer between 1 and N-1 which tells that when the dial plate is dialed once, the a_i-th gear will rotate across by b_i gears.

A test case of $K = 0$ and $N = 0$ indicates the end of input, and should not be processed.

Output

For each test case, there is a single line. If the lock can be opened, the line contains the least number of times Kid should dial the dial plates; otherwise, output “No solution”.

Sample Input	Output for the Sample Input
1 4 2 1 1 2 4 8 8 8 8 8 4 1 1 2 2 4 1 3 1 2 4 7 3 2 2 1 5 3 5 1 2 7 0 0	No solution 2

Problem H: The Separator in Grid

Input File: grid.in

Given a connected, undirected graph G = (V, E), where V is the vertex set consisting a collection of nodes, and E is the set of edges, each of which connects two nodes from V. A vertex subset S is a *separator* if the subgraph induced by the vertices in V, but not in S, has two connected components. We shall use the notation [S, W, B] to represent the partition, where the removal of the separator S will give two connected components W and B.

In this problem, we consider the separators in grids. Each node in a grid is connected to its eight neighbors (if they exist). In Figure-1, we illustrate a partition of a 6*6 grid with a 9-point separator (gray nodes in the figure). The nodes on the left of the separator are in set W (white nodes), and the nodes on the right of the separator are in set B (black nodes).

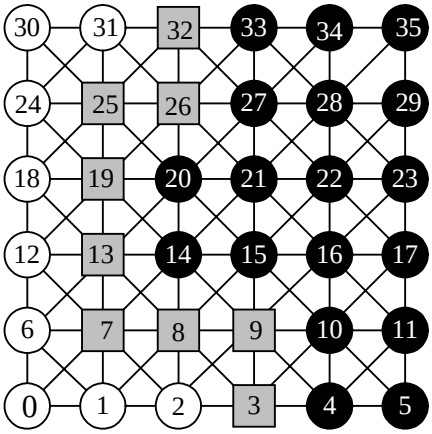

Figure-1. Partition (S, W, B)

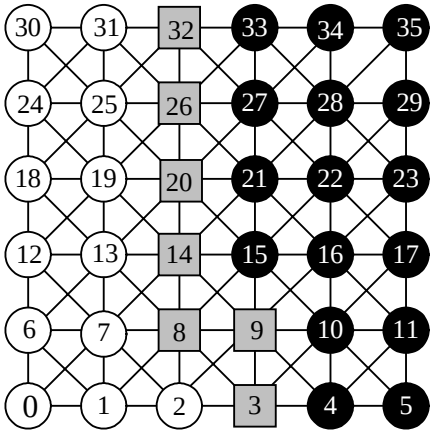

Figure-2. New Partition

To simplify the problem, you can assume that all the separators referred in this problem satisfy the following restrictions:

1) It's a minimal separator. A separator is *minimal* if no subset of it forms a separator.
2) It begins from a node on the top line of the grid, except the corner (i.e. 30 and 35 in the figures), and ends with a node on the bottom line of the grid, also except the corner (i.e. 0 and 5 in the figures).
3) On its way from top to bottom, it can go left, right or down, but never go up.

Now we describe a method to improve a given partition on a grid, through which we can reduce the number of nodes in the separator. This method contains two steps:

1) Select several nodes from B and add them into S. Any of the selected nodes must have a left neighbor which is in S.
2) Remove several nodes from S (excluding the nodes added in the former step), and add them into W.

After the improvement, we should ensure S is still a separator, and make the number of nodes in S as small as possible. As for Figure-1, we should add 14 and 20 into S, and remove 7, 13, 19 and 25 from S. After that, we obtain a new partition with a 7-point separator shown in Figure-2.

Your task is, given a partition on a grid, to determine the least number of nodes in the separator after the improvement.

Input

There are several test cases. Each case begins with a line containing two integers, N and M (3 <= M, N <= 200). In each of the following N lines, there are M characters, describing the initial partition of the M*N grid. Every character is 'S', 'W' or 'B'. It is confirmed that each of these three characters appears at least once in each line, and 'W's are always on the left of 'S's.

A test case of $N = 0$ and $M = 0$ indicates the end of input, and should not be processed.

Output

For each test case, you should output one line containing one integer, which is the least number of nodes in the separator after the improvement.

Sample Input	Output for the Sample Input
6 6 WWSBBB WSSBBB WSBBBB WSBBBB WSSSBB WWWSBB 0 0	7

Problem I: The Lost House

Input File: snail.in

One day a snail climbed up to a big tree and finally came to the end of a branch. What a different feeling to look down from such a high place he had never been to before! However, he was very tired due to the long time of climbing, and fell asleep. An unbelievable thing happened when he woke up —— he found himself lying in a meadow and his house originally on his back disappeared! Immediately he realized that he fell off the branch when he was sleeping! He was sure that his house must still be on the branch he had been sleeping on. The snail began to climb the tree again, since he could not live without his house.

When reaching the first fork of the tree, he sadly found that he could not remember the route that he climbed before. In order to find his lovely house, the snail decided to go to the end of every branch. It was dangerous to walk without the protection of the house, so he wished to search the tree in the best way.

Fortunately, there lived many warm-hearted worms in the tree that could accurately tell the snail whether he had ever passed their places or not before he fell off.

Now our job is to help the snail. We pay most of our attention to two parts of the tree —— the forks of the branches and the ends of the branches, which we call them *key points* because key events always happen there, such as choosing a path, getting the help from a worm and arriving at the house he is searching for.

Assume all worms live at key points, and all the branches between two neighboring key points have the same distance of 1. The snail is now at the first fork of the tree.

Our purpose is to find a proper route along which he can find his house as soon as possible, through the analysis of the structure of the tree and the locations of the worms. The only restriction on the route is that he must not go down from a fork until he has reached all the ends grown from this fork.

The house may be left at the end of any branches in an equal probability. We focus on the mathematical expectation of the distance the snail has to cover before arriving his house. We wish the value to be as small as possible.

As illustrated in Figure-1, the snail is at the key point 1 and his house is at a certain point among 2, 4 and 5. A worm lives at point 3, who can tell the snail whether his house is at one of point 4 and 5 or not. Therefore, the snail can choose two strategies. He can go to point 2 first. If he cannot find the house there, he should go back to point 1, and then reaches point 4 (or 5) by point 3. If still not, he has to return point 3, then go to point 5 (or 4), where he will undoubtedly find his house. In this choice, the snail covers distances of 1, 4, 6 corresponding to the circumstances under which the house is located at point 2, 4 (or 5), 5 (or 4) respectively. So the expectation value is (1 + 4 + 6) / 3

= 11 / 3. Obviously, this strategy does not make full use of the information from the worm. If the snail goes to point 3 and gets useful information from the worm first, and then chooses to go back to point 1 then towards point 2, or go to point 4 or 5 to take his chance, the distances he covers will be 2, 3, 4 corresponding to the different locations of the house. In such a strategy, the mathematical expectation will be (2 + 3 + 4) / 3 = 3, and it is the very route along which the snail should search the tree.

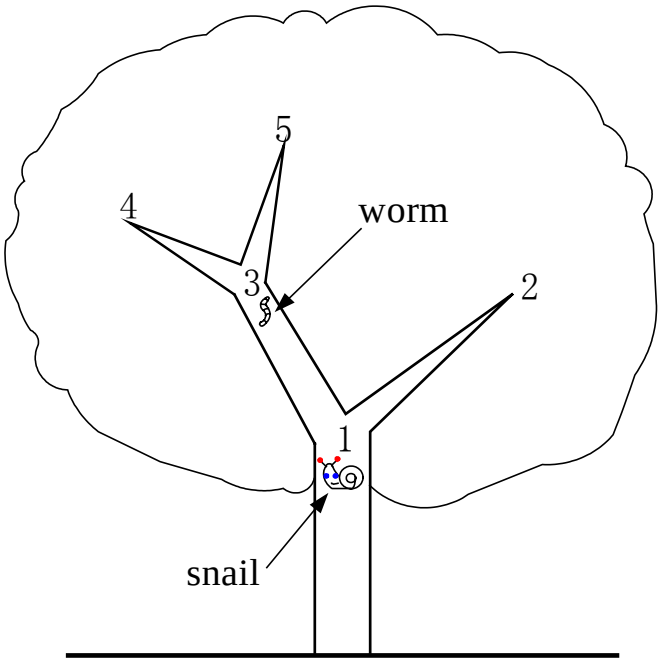

Figure-1

Input

The input contains several sets of test data. Each set begins with a line containing one integer *N*, no more than 1000, which indicates the number of key points in the tree. Then follow *N* lines describing the *N* key points. For convenience, we number all the key points from 1 to *N*. The key point numbered with 1 is always the first fork of the tree. Other numbers may be any key points in the tree except the first fork. The *i*-th line in these *N* lines describes the key point with number *i*. Each line consists of one integer and one uppercase character 'Y' or 'N' separated by a single space, which represents the number of the previous key point and whether there lives a worm ('Y' means lives and 'N' means not). The *previous key point* means the neighboring key point in the shortest path between this key point and the key point numbered 1. In the above illustration, the previous key point of point 2 or 3 is point 1, while the previous key point of point 4 or 5 is point 3. This integer is -1 for the key point 1, means it has no previous key point. You can assume a fork has at most eight branches. The first set in the sample input describes the above illustration.

A test case of *N* = 0 indicates the end of input, and should not be processed.

Output

Output one line for each set of input data. The line contains one float number with exactly four digits after the decimal point, which is the mathematical expectation value.

Sample Input	Output for the Sample Input
5 -1 N 1 N 1 Y 3 N 3 N 10 -1 N 1 Y 1 N 2 N 2 N 2 N 3 N 3 Y 8 N 8 N 6 -1 N 1 N 1 Y 1 N 3 N 3 N 0	3.0000 5.0000 3.5000