Problem A
Gift from the Goddess of Programming

Input: Standard Input
Time Limit: 30 seconds

The goddess of programming is reviewing a thick logbook, which is a yearly record of visitors to her holy altar of programming. The logbook also records her visits at the altar.

The altar attracts programmers from all over the world because one visitor is chosen every year and endowed with a gift of miracle programming power by the goddess. The endowed programmer is chosen from those programmers who spent the longest time at the altar *during the goddess's presence*. There have been enthusiastic visitors who spent very long time at the altar but failed to receive the gift because the goddess was absent during their visits.

Now, your mission is to write a program that finds how long the programmer to be endowed stayed at the altar during the goddess's presence.

Input

The input is a sequence of datasets. The number of datasets is less than 100. Each dataset is formatted as follows.

n
M_1/D_1 h_1:m_1 e_1 p_1
M_2/D_2 h_2:m_2 e_2 p_2
$\vdots$
M_n/D_n h_n:m_n e_n p_n

The first line of a dataset contains a positive even integer, $n \leq 1000$, which denotes the number of lines of the logbook. This line is followed by n lines of space-separated data, where M_i/D_i identifies the month and the day of the visit, $h_i : m_i$ represents the time of either the entrance to or exit from the altar, e_i is either `I` for entrance, or `O` for exit, and p_i identifies the visitor.

All the lines in the logbook are formatted in a fixed-column format. Both the month and the day in the month are represented by two digits. Therefore April 1 is represented by `04/01` and not by `4/1`. The time is described in the 24-hour system, taking two digits for the hour, followed by a colon and two digits for minutes, `09:13` for instance and not like `9:13`. A programmer is identified by an ID, a unique number using three digits. The same format is used to indicate entrance and exit of the goddess, whose ID is `000`.

All the lines in the logbook are sorted in ascending order with respect to date and time. Because the altar is closed at midnight, the altar is emptied at `00:00`. You may assume that each time in the input is between `00:01` and `23:59`, inclusive.

A programmer may leave the altar just after entering it. In this case, the entrance and exit time are the same and the length of such a visit is considered 0 minute. You may assume for such entrance and exit records, the line that corresponds to the entrance appears earlier in the input than the line that corresponds to the exit. You may assume that at least one programmer appears in the logbook.

The end of the input is indicated by a line containing a single zero.

Output

For each dataset, output the total sum of the *blessed time* of the endowed programmer. The blessed time of a programmer is the length of his/her stay at the altar during the presence of the goddess. The endowed programmer is the one whose total blessed time is the longest among all the programmers. The output should be represented in minutes. Note that the goddess of programming is not a programmer.

Sample Input

```
14
04/21 09:00 I 000
04/21 09:00 I 001
04/21 09:15 I 002
04/21 09:30 O 001
04/21 09:45 O 000
04/21 10:00 O 002
04/28 09:00 I 003
04/28 09:15 I 000
04/28 09:30 I 004
04/28 09:45 O 004
04/28 10:00 O 000
04/28 10:15 O 003
04/29 20:00 I 002
04/29 21:30 O 002
20
06/01 09:00 I 001
06/01 09:15 I 002
06/01 09:15 I 003
06/01 09:30 O 002
06/01 10:00 I 000
06/01 10:15 O 001
06/01 10:30 I 002
06/01 10:45 O 002
```

```
06/01 11:00 I 001
06/01 11:15 O 000
06/01 11:30 I 002
06/01 11:45 O 001
06/01 12:00 O 002
06/01 12:15 I 000
06/01 12:30 I 002
06/01 12:45 O 000
06/01 13:00 I 000
06/01 13:15 O 000
06/01 13:30 O 002
06/01 13:45 O 003
0
```

Output for the Sample Input

```
45
120
```

Problem B
The Sorcerer's Donut

Input: Standard Input
Time Limit: 30 seconds

Your master went to the town for a day. You could have a relaxed day without hearing his scolding. But he ordered you to make donuts dough by the evening. Loving donuts so much, he can't live without eating tens of donuts everyday. What a chore for such a beautiful day.

But last week, you overheard a magic spell that your master was using. It was the time to try. You casted the spell on a broomstick sitting on a corner of the kitchen. With a flash of lights, the broom sprouted two arms and two legs, and became alive. You ordered him, then he brought flour from the storage, and started kneading dough. The spell worked, and how fast he kneaded it!

A few minutes later, there was a tall pile of dough on the kitchen table. That was enough for the next week. "OK, stop now." You ordered. But he didn't stop. Help! You didn't know the spell to stop him! Soon the kitchen table was filled with hundreds of pieces of dough, and he still worked as fast as he could. If you could not stop him now, you would be choked in the kitchen filled with pieces of dough.

Wait, didn't your master write his spells on his notebooks? You went to his den, and found the notebook that recorded the spell of cessation.

But it was not the end of the story. The spell written in the notebook is not easily read by others. He used a plastic model of a donut as a notebook for recording the spell. He split the surface of the donut-shaped model into square mesh (Figure B.1), and filled with the letters (Figure B.2). He hid the spell so carefully that the pattern on the surface looked meaningless. But you knew that he wrote the pattern so that the spell "appears" more than once (see the next paragraph for the precise conditions). The spell was not necessarily written in the left-to-right direction, but any of the 8 directions, namely left-to-right, right-to-left, top-down, bottom-up, and the 4 diagonal directions.

You should be able to find the spell as the longest string that appears more than once. Here, a string is considered to appear more than once if there are square sequences having the string on the donut that satisfy the following conditions.

- Each square sequence does not overlap itself. (Two square sequences can share some squares.)
- The square sequences start from different squares, and/or go to different directions.

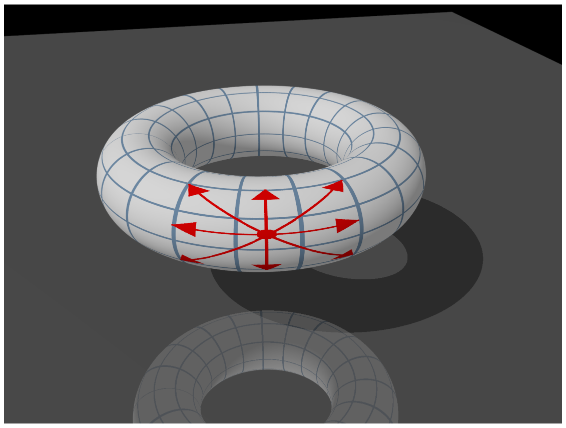

Figure B.1: The Sorcerer's Donut Before Filled with Letters, Showing the Mesh and 8 Possible Spell Directions

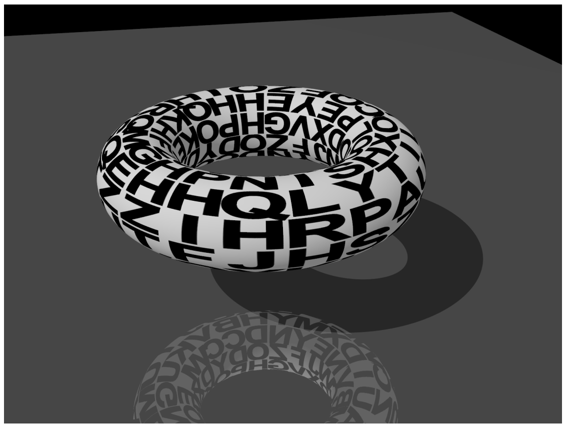

Figure B.2: The Sorcerer's Donut After Filled with Letters

Note that a palindrome (i.e., a string that is the same whether you read it backwards or forwards) that satisfies the first condition "appears" twice.

The pattern on the donut is given as a matrix of letters as follows.

```
ABCD
EFGH
IJKL
```

Note that the surface of the donut has no ends; the top and bottom sides, and the left and right sides of the pattern are respectively connected. There can be square sequences longer than both the vertical and horizontal lengths of the pattern. For example, from the letter `F` in the above pattern, the strings in the longest non-self-overlapping sequences towards the 8 directions are as follows.

```
FGHE
FKDEJCHIBGLA
FJB
FIDGJAHKBELC
FEHG
FALGBIHCJEDK
FBJ
FCLEBKHAJGDI
```

Please write a program that finds the magic spell before you will be choked with pieces of donuts dough.

Input

The input is a sequence of datasets. Each dataset begins with a line of two integers h and w, which denote the size of the pattern, followed by h lines of w uppercase letters from `A` to `Z`, inclusive, which denote the pattern on the donut. You may assume $3 \leq h \leq 10$ and $3 \leq w \leq 20$.

The end of the input is indicated by a line containing two zeros.

Output

For each dataset, output the magic spell. If there is more than one longest string of the same length, the first one in the dictionary order must be the spell. The spell is known to be at least two letters long. When no spell is found, output `0` (zero).

Sample Input

```
5 7
RRCABXT
AABMFAB
RROMJAC
APTADAB
YABADAO
3 13
ABCDEFGHIJKLM
XMADAMIMADAMY
ACEGIKMOQSUWY
3 4
DEFG
ACAB
HIJK
3 6
ABCDEF
GHIAKL
MNOPQR
10 19
JFZODYDXMZZPEYTRNCW
XVGHPOKEYNZTQFZJKOD
EYEHHQKHFZOVNRGOOLP
QFZOIHRQMGHPNISHXOC
DRGILJHSQEHHQLYTILL
NCSHQMKHTZZIHRPAUJA
NCCTINCLAUTFJHSZBVK
LPBAUJIUMBVQYKHTZCW
XMYHBVKUGNCWTLLAUID
EYNDCCWLEOODXYUMBVN
0 0
```

Output for the Sample Input

```
ABRACADABRA
MADAMIMADAM
ABAC
0
ABCDEFGHIJKLMNOPQRSTUVWXYZHHHHHABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Problem C
Weaker than Planned

Input: Standard Input
Time Limit: 30 seconds

The committee members of the Kitoshima programming contest had decided to use cryptographic software for their secret communication. They had asked a company, Kodai Software, to develop cryptographic software that employed a cipher based on highly sophisticated mathematics.

According to reports on IT projects, many projects are not delivered on time, on budget, with required features and functions. This applied to this case. Kodai Software failed to implement the cipher by the appointed date of delivery, and asked to use a simpler version that employed a type of substitution cipher for the moment. The committee members got angry and strongly requested to deliver the full specification product, but they unwillingly decided to use this inferior product for the moment.

In what follows, we call the text before encryption, *plaintext,* and the text after encryption, *ciphertext.*

This simple cipher substitutes letters in the plaintext, and its substitution rule is specified with a set of pairs. A pair consists of two letters and is unordered, that is, the order of the letters in the pair does not matter. A pair (`A`, `B`) and a pair (`B`, `A`) have the same meaning. In one substitution rule, one letter can appear in at most one single pair. When a letter in a pair appears in the plaintext, the letter is replaced with the other letter in the pair. Letters not specified in any pairs are left as they are.

For example, by substituting the plaintext

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

with the substitution rule {(`A`, `Z`), (`B`, `Y`)} results in the following ciphertext.

```
ZYCDEFGHIJKLMNOPQRSTUVWXBA
```

This may be a big chance for us, because the substitution rule seems weak against cracking. We may be able to know communications between committee members. The mission here is to develop a deciphering program that finds the plaintext messages from given ciphertext messages.

A ciphertext message is composed of one or more ciphertext words. A ciphertext word is generated from a plaintext word with a substitution rule. You have a list of candidate words containing the words that can appear in the plaintext; no other words may appear. Some words in the list may not actually be used in the plaintext.

There always exists at least one sequence of candidate words from which the given ciphertext is obtained by some substitution rule. There may be cases where it is impossible to uniquely identify the plaintext from a given ciphertext and the list of candidate words.

Input

The input consists of multiple datasets, each of which contains a ciphertext message and a list of candidate words in the following format.

> n
> $word_1$
> $\vdots$
> $word_n$
> *sequence*

n in the first line is a positive integer, representing the number of candidate words. Each of the next n lines represents one of the candidate words. The last line, *sequence*, is a sequence of one or more ciphertext words separated by a single space and terminated with a period.

You may assume the number of characters in each *sequence* is more than 1 and less than or equal to 80 including spaces and the period. The number of candidate words in the list, n, does not exceed 20. Only 26 uppercase letters, `A` to `Z`, are used in the words and the length of each word is from 1 to 20, inclusive.

A line of a single zero indicates the end of the input.

Output

For each dataset, your program should print the deciphered message in a line. Two adjacent words in an output line should be separated by a single space and the last word should be followed by a single period. When it is impossible to uniquely identify the plaintext, the output line should be a single hyphen followed by a single period.

Sample Input

```
4
A
AND
CAT
DOG
Z XUW ZVX Z YZT.
```

```
2
AZ
AY
ZA.
2
AA
BB
CC.
16
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
ABCDEFGHIJKLMNO
A B C D E F G H I J K L M N O ABCDEFGHIJKLMNO.
0
```

Output for the Sample Input

```
A DOG AND A CAT.
AZ.
-.
A B C D E F G H I J K L M N O ABCDEFGHIJKLMNO.
```

Problem D
Long Distance Taxi

Input: Standard Input
Time Limit: 30 seconds

A taxi driver, Nakamura, was so delighted because he got a passenger who wanted to go to a city thousands of kilometers away. However, he had a problem. As you may know, most taxis in Japan run on liquefied petroleum gas (LPG) because it is cheaper than gasoline. There are more than 50,000 gas stations in the country, but less than one percent of them sell LPG. Although the LPG tank of his car was full, the tank capacity is limited and his car runs 10 kilometer per liter, so he may not be able to get to the destination without filling the tank on the way. He knew all the locations of LPG stations.

Your task is to write a program that finds the best way from the current location to the destination without running out of gas.

Input

The input consists of several datasets, and each dataset is in the following format.

$N \quad M \quad cap$
$src \quad dest$
$c_{1,1} \quad c_{1,2} \quad d_1$
$c_{2,1} \quad c_{2,2} \quad d_2$
$\vdots$
$c_{N,1} \quad c_{N,2} \quad d_N$
s_1
s_2
$\vdots$
s_M

The first line of a dataset contains three integers (N, M, cap), where N is the number of roads $(1 \leq N \leq 3000)$, M is the number of LPG stations $(1 \leq M \leq 300)$, and cap is the tank capacity $(1 \leq cap \leq 200)$ in liter. The next line contains the name of the current city (src) and the name of the destination city $(dest)$. The destination city is always different from the current city. The following N lines describe roads that connect cities. The road i $(1 \leq i \leq N)$ connects two different cities $c_{i,1}$ and $c_{i,2}$ with an integer distance d_i $(0 < d_i \leq 2000)$ in kilometer, and he can go from either city to the other. You can assume that no two different roads connect the same pair of cities. The columns are separated by a single space. The next M lines $(s_1, s_2, \ldots, s_M)$

indicate the names of the cities with LPG station. You can assume that a city with LPG station has at least one road.

The name of a city has no more than 15 characters. Only English alphabet ('A' to 'Z' and 'a' to 'z', case sensitive) is allowed for the name.

A line with three zeros terminates the input.

Output

For each dataset, output a line containing the length (in kilometer) of the shortest possible journey from the current city to the destination city. If Nakamura cannot reach the destination, output "`-1`" (without quotation marks). You must not output any other characters.

The actual tank capacity is usually a little bit larger than that on the specification sheet, so you can assume that he can reach a city even when the remaining amount of the gas becomes exactly zero. In addition, you can always fill the tank at the destination so you do not have to worry about the return trip.

Sample Input

```
6 3 34
Tokyo Kyoto
Tokyo Niigata 335
Tokyo Shizuoka 174
Shizuoka Nagoya 176
Nagoya Kyoto 195
Toyama Niigata 215
Toyama Kyoto 296
Nagoya
Niigata
Toyama
6 3 30
Tokyo Kyoto
Tokyo Niigata 335
Tokyo Shizuoka 174
Shizuoka Nagoya 176
Nagoya Kyoto 195
Toyama Niigata 215
Toyama Kyoto 296
Nagoya
Niigata
Toyama
0 0 0
```

Output for the Sample Input

```
846
-1
```

Problem E
Driving an Icosahedral Rover

Input: Standard Input
Time Limit: 60 seconds

After decades of fruitless efforts, one of the expedition teams of ITO (Intersolar Tourism Organization) finally found a planet that would surely provide one of the best tourist attractions within a ten light-year radius from our solar system. The most attractive feature of the planet, besides its comfortable gravity and calm weather, is the area called *Mare Triangularis.* Despite the name, the area is not covered with water but is a great plane. Its unique feature is that it is divided into equilateral triangular sections of the same size, called *trigons.* The *trigons* provide a unique impressive landscape, a must for tourism. It is no wonder the board of ITO decided to invest a vast amount on the planet.

Despite the expected secrecy of the staff, the Society of Astrogeology caught this information in no time, as always. They immediately sent their president's letter to the Institute of Science and Education of the Commonwealth Galactica claiming that authoritative academic inspections were to be completed before any commercial exploitation might damage the nature.

Fortunately, astrogeologists do not plan to practice all the possible inspections on all of the *trigons;* there are far too many of them. Inspections are planned only on some characteristic *trigons* and, for each of them, in one of twenty different scientific aspects.

To accelerate building this new tourist resort, ITO's construction machinery team has already succeeded in putting their brand-new invention in practical use. It is a rover vehicle of the shape of an *icosahedron,* a regular polyhedron with twenty faces of equilateral triangles. The machine is customized so that each of the twenty faces exactly fits each of the *trigons*. Controlling the high-tech *gyromotor* installed inside its body, the rover can roll onto one of the three *trigons* neighboring the one its bottom is on.

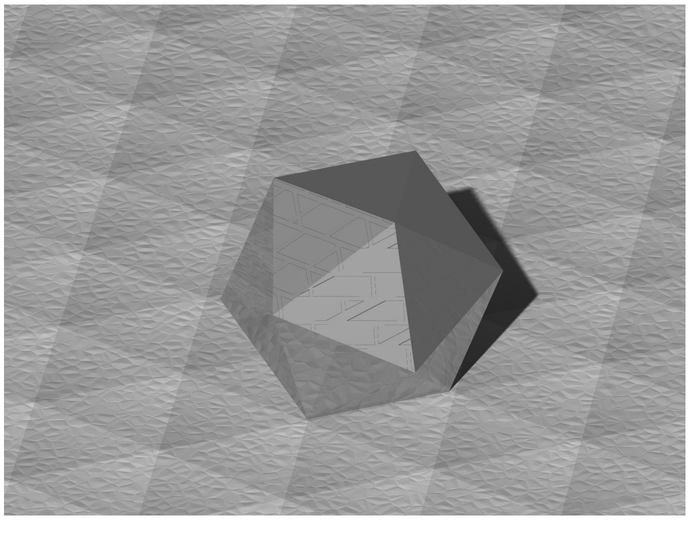
Figure E.1: The Rover on Mare Triangularis

Each of the twenty faces has its own function. The set of equipments installed on the bottom face touching the ground can be applied to the *trigon* it is on. Of course, the rover was meant

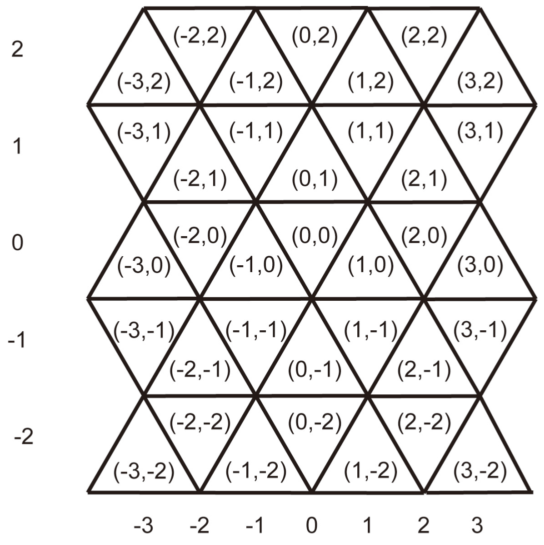

Figure E.2: The Coordinate System

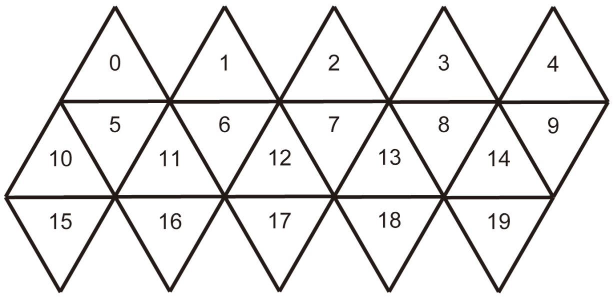

Figure E.3: Face Numbering

to accelerate construction of the luxury hotels to host rich interstellar travelers, but, changing the installed equipment sets, it can also be used to accelerate academic inspections.

You are the driver of this rover and are asked to move the vehicle onto the *trigon* specified by the leader of the scientific commission with the smallest possible steps. What makes your task more difficult is that the designated face installed with the appropriate set of equipments has to be the bottom. The direction of the rover does not matter.

The *trigons* of *Mare Triangularis* are given two-dimensional coordinates as shown in Figure E.2. Like maps used for the Earth, the x axis is from the west to the east, and the y axis is from the south to the north. Note that all the *trigons* with its coordinates (x, y) has neighboring *trigons* with coordinates $(x - 1, y)$ and $(x + 1, y)$. In addition to these, when $x + y$ is even, it has a neighbor $(x, y + 1)$; otherwise, that is, when $x + y$ is odd, it has a neighbor $(x, y - 1)$.

Figure E.3 shows a development of the skin of the rover. The top face of the development makes the exterior. That is, if the numbers on faces of the development were actually marked on the faces of the rover, they should been readable from its outside. These numbers are used to identify the faces.

When you start the rover, it is on the *trigon* $(0, 0)$ and the face 0 is touching the ground. The rover is placed so that rolling towards north onto the *trigon* $(0, 1)$ makes the face numbered 5 to be at the bottom.

As your first step, you can choose one of the three adjacent *trigons,* namely those with coordinates $(-1, 0)$, $(1, 0)$, and $(0, 1)$, to visit. The bottom will be the face numbered 4, 1, and 5, respectively. If you choose to go to $(1, 0)$ in the first rolling step, the second step can bring the rover to either of $(0, 0)$, $(2, 0)$, or $(1, -1)$. The bottom face will be either of 0, 6, or 2, correspondingly. The rover may visit any of the *trigons* twice or more, including the start and the goal *trigons,* when appropriate.

The theoretical design section of ITO showed that the rover can reach any goal *trigon* on the specified bottom face within a finite number of steps.

Input

The input consists of a number of datasets. The number of datasets does not exceed 50.

Each of the datasets has three integers x, y, and n in one line, separated by a space. Here, (x, y) specifies the coordinates of the *trigon* to which you have to move the rover, and n specifies the face that should be at the bottom.

The end of the input is indicated by a line containing three zeros.

Output

The output for each dataset should be a line containing a single integer that gives the *minimum number* of steps required to set the rover on the specified *trigon* with the specified face touching the ground. No other characters should appear in the output.

You can assume that the maximum number of required steps does not exceed 100. *Mare Triangularis* is broad enough so that any of its edges cannot be reached within that number of steps.

Sample Input

```
0 0 1
3 5 2
-4 1 3
13 -13 2
-32 15 9
-50 50 0
0 0 0
```

Output for the Sample Input

```
6
10
9
30
47
100
```

Problem F
City Merger

Input: Standard Input
Time Limit: 60 seconds

Recent improvements in information and communication technology have made it possible to provide municipal service to a wider area more quickly and with less costs. Stimulated by this, and probably for saving their not sufficient funds, mayors of many cities started to discuss on mergers of their cities.

There are, of course, many obstacles to actually put the planned mergers in practice. Each city has its own culture of which citizens are proud. One of the largest sources of friction is with the name of the new city. All citizens would insist that the name of the new city should have the original name of their own city at least as a part of it. Simply concatenating all the original names would, however, make the name too long for everyday use.

You are asked by a group of mayors to write a program that finds the shortest possible name for the new city that includes all the original names of the merged cities. If two or more cities have common parts, they can be overlapped. For example, if “`FUKUOKA`”, “`OKAYAMA`”, and “`YAMAGUCHI`” cities are to be merged, “`FUKUOKAYAMAGUCHI`” is such a name that include all three of the original city names. Although this includes all the characters of the city name “`FUKUYAMA`” in this order, it does not appear as a consecutive substring, and thus “`FUKUYAMA`” is not considered to be included in the name.

Input

The input is a sequence of datasets. Each dataset begins with a line containing a positive integer n ($n \leq 14$), which denotes the number of cities to be merged. The following n lines contain the names of the cities in uppercase alphabetical letters, one in each line. You may assume that none of the original city names has more than 20 characters. Of course, no two cities have the same name.

The end of the input is indicated by a line consisting of a zero.

Output

For each dataset, output the length of the shortest possible name of the new city in one line. The output should not contain any other characters.

Sample Input

```
3
FUKUOKA
OKAYAMA
YAMAGUCHI
3
FUKUOKA
FUKUYAMA
OKAYAMA
2
ABCDE
EDCBA
4
GA
DEFG
CDDE
ABCD
2
ABCDE
C
14
AAAAA
BBBBB
CCCCC
DDDDD
EEEEE
FFFFF
GGGGG
HHHHH
IIIII
JJJJJ
KKKKK
LLLLL
MMMMM
NNNNN
0
```

Output for the Sample Input

```
16
19
9
9
5
70
```

Problem G
Captain Q's Treasure

Input: Standard Input
Time Limit: 30 seconds

You got an old map, which turned out to be drawn by the infamous pirate "Captain Q". It shows the locations of a lot of treasure chests buried in an island.

The map is divided into square sections, each of which has a digit on it or has no digit. The digit represents the number of chests in its 9 neighboring sections (the section itself and its 8 neighbors). You may assume that there is at most one chest in each section.

Although you have the map, you can't determine the sections where the chests are buried. Even the total number of chests buried in the island is unknown. However, it is possible to calculate the minimum number of chests buried in the island. Your mission in this problem is to write a program that calculates it.

Input

The input is a sequence of datasets. Each dataset is formatted as follows.

> h w
> *map*

The first line of a dataset consists of two positive integers h and w. h is the height of the map and w is the width of the map. You may assume $1 \leq h \leq 15$ and $1 \leq w \leq 15$.

The following h lines give the map. Each line consists of w characters and corresponds to a horizontal strip of the map. Each of the characters in the line represents the state of a section as follows.

'`.`': The section is not a part of the island (water). No chest is here.

'`*`': The section is a part of the island, and the number of chests in its 9 neighbors is not known.

'`0`'–'`9`': The section is a part of the island, and the digit represents the number of chests in its 9 neighbors.

You may assume that the map is not self-contradicting, i.e., there is at least one arrangement of chests. You may also assume the number of sections with digits is at least one and at most 15.

A line containing two zeros indicates the end of the input.

Output

For each dataset, output a line that contains the minimum number of chests. The output should not contain any other character.

Sample Input

```
5 6
*2.2**
..*...
..2...
..*...
*2.2**
6 5
.*2*.
..*..
..*..
..2..
..*..
.*2*.
5 6
.1111.
**...*
33....
**...0
.*2**.
6 9
....1....
...1.1...
....1....
.1..*..1.
1.1***1.1
.1..*..1.
9 9
*********
*4*4*4*4*
*********
*4*4*4*4*
*********
*4*4*4*4*
*********
*4*4*4***
*********
0 0
```

Output for the Sample Input

```
6
5
5
6
23
```

Problem H
ASCII Expression

Input: Standard Input
Time Limit: 30 seconds

Mathematical expressions appearing in old papers and old technical articles are printed with typewriter in several lines, where a fixed-width or monospaced font is required to print characters (digits, symbols and spaces). Let us consider the following mathematical expression.

$$\left(1 - \frac{4}{3^2}\right)^2 \times -5 + 6$$

It is printed in the following four lines:

```
        4   2
( 1 - ---- )  * - 5 + 6
        2
       3
```

where "`- 5`" indicates unary minus followed by 5. We call such an expression of lines "ASCII expression".

For helping those who want to evaluate ASCII expressions obtained through optical character recognition (OCR) from old papers, your job is to write a program that recognizes the structure of ASCII expressions and computes their values.

For the sake of simplicity, you may assume that ASCII expressions are constructed by the following rules. Its syntax is shown in Table H.1.

(1) Terminal symbols are '`0`', '`1`', '`2`', '`3`', '`4`', '`5`', '`6`', '`7`', '`8`', '`9`', '`+`', '`-`', '`*`', '`(`', '`)`', and '` `'.

(2) Nonterminal symbols are *expr*, *term*, *factor*, *powexpr*, *primary*, *fraction* and *digit*. The start symbol is *expr*.

(3) A "cell" is a rectangular region of characters that corresponds to a terminal or nonterminal symbol (Figure H.1). In the cell, there are no redundant rows and columns that consist only of space characters. A cell corresponding to a terminal symbol consists of a single character. A cell corresponding to a nonterminal symbol contains cell(s) corresponding to its descendant(s) but never partially overlaps others.

(4) Each cell has a base-line, a top-line, and a bottom-line. The base-lines of child cells of the right-hand side of rules I, II, III, and V should be aligned. Their vertical position defines the base-line position of their left-hand side cell.

Table H.1: Rules for constructing ASCII expressions (similar to Backus-Naur Form)
The box indicates the cell of the terminal or nonterminal symbol that corresponds to a rectangular region of characters. Note that each syntactically-needed **space character** is explicitly indicated by the **period character** denoted by $\boxed{\texttt{.}}$, here.

(I)	$\boxed{\texttt{expr}}\ \texttt{::=}\ \boxed{\texttt{term}}\ \texttt{\|}\ \boxed{\texttt{expr}}\ \boxed{\texttt{.}}\ \boxed{\texttt{+}}\ \boxed{\texttt{.}}\ \boxed{\texttt{term}}\ \texttt{\|}\ \boxed{\texttt{expr}}\ \boxed{\texttt{.}}\ \boxed{\texttt{-}}\ \boxed{\texttt{.}}\ \boxed{\texttt{term}}$
(II)	$\boxed{\texttt{term}}\ \texttt{::=}\ \boxed{\texttt{factor}}\ \texttt{\|}\ \boxed{\texttt{term}}\ \boxed{\texttt{.}}\ \texttt{*}\ \boxed{\texttt{.}}\ \boxed{\texttt{factor}}$
(III)	$\boxed{\texttt{factor}}\ \texttt{::=}\ \boxed{\texttt{powexpr}}\ \texttt{\|}\ \boxed{\texttt{fraction}}\ \texttt{\|}\ \boxed{\texttt{-}}\ \boxed{\texttt{.}}\ \boxed{\texttt{factor}}$
(IV)	$\boxed{\texttt{powexpr}}\ \texttt{::=}\ \boxed{\texttt{primary}}\ \texttt{\|}\ \boxed{\texttt{primary}}^{\boxed{\texttt{digit}}}$
(V)	$\boxed{\texttt{primary}}\ \texttt{::=}\ \boxed{\texttt{digit}}\ \texttt{\|}\ \boxed{\texttt{(}}\ \boxed{\texttt{.}}\ \boxed{\texttt{expr}}\ \boxed{\texttt{.}}\ \boxed{\texttt{)}}$
(VI)	$\boxed{\texttt{fraction}}\ \texttt{::=}\ \begin{array}{c}\boxed{\texttt{expr}}\\ \texttt{---------}\\ \boxed{\texttt{expr}}\end{array}$
(VII)	$\boxed{\texttt{digit}}\ \texttt{::=}\ \boxed{\texttt{0}}\ \texttt{\|}\ \boxed{\texttt{1}}\ \texttt{\|}\ \boxed{\texttt{2}}\ \texttt{\|}\ \boxed{\texttt{3}}\ \texttt{\|}\ \boxed{\texttt{4}}\ \texttt{\|}\ \boxed{\texttt{5}}\ \texttt{\|}\ \boxed{\texttt{6}}\ \texttt{\|}\ \boxed{\texttt{7}}\ \texttt{\|}\ \boxed{\texttt{8}}\ \texttt{\|}\ \boxed{\texttt{9}}$

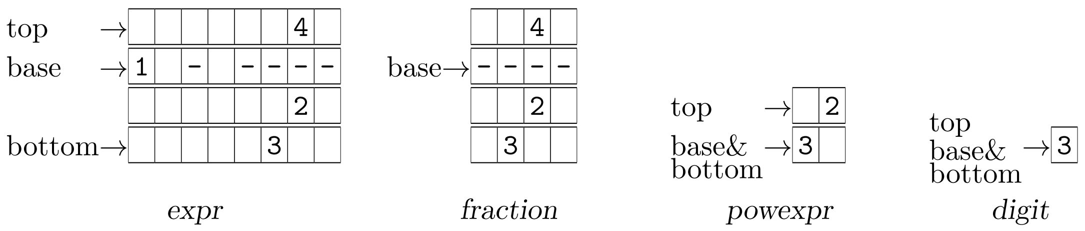

Figure H.1: Top, base, bottom lines: *expr* $1 - \frac{4}{3^2}$, *fraction* $\frac{4}{3^2}$, *powexpr* 3^2, and *digit* 3.

(5) *powexpr* consists of a *primary* and an optional *digit*. The *digit* is placed one line above the base-line of the *primary* cell. They are horizontally adjacent to each other. The base-line of a *powexpr* is that of the *primary*.

(6) *fraction* is indicated by three or more consecutive hyphens called "vinculum". Its dividend *expr* is placed just above the vinculum, and its divisor *expr* is placed just beneath it. The number of the hyphens of the vinculum, denoted by w_h, is equal to $2 + \max(w_1, w_2)$, where w_1 and w_2 indicate the width of the cell of the dividend and that of the divisor, respectively. These cells are centered, where there are $\lceil (w_h - w_k)/2 \rceil$ space characters to the left and $\lfloor (w_h - w_k)/2 \rfloor$ space characters to the right, $(k = 1, 2)$. The base-line of a *fraction* is at the position of the vinculum.

(7) *digit* consists of one character.

For example, the negative fraction $-\frac{3}{4}$ is represented in three lines:

```
   3
- ---
   4
```

where the left-most hyphen means a unary minus operator. One space character is required between the unary minus and the vinculum of the fraction.

The fraction $\frac{3+4\times-2}{-1-2^2}$ is represented in four lines:

```
 3 + 4 * - 2
-------------
          2
   - 1 - 2
```

where the widths of the cells of the dividend and divisor are 11 and 8 respectively. Hence the number of hyphens of the vinculum is $2 + \max(11, 8) = 13$. The divisor is centered by $\lceil(13 - 8)/2\rceil = 3$ space characters (hyphens) to the left and $\lfloor(13 - 8)/2\rfloor = 2$ to the right.

The *powexpr* $(4^2)^3$ is represented in two lines:

```
   2  3
( 4  )
```

where the cell for 2 is placed one line above the base-line of the cell for 4, and the cell for 3 is placed one line above the base-line of the cell for a *primary* (4^2).

Input

The input consists of multiple datasets, followed by a line containing a zero. Each dataset has the following format.

n
str_1
str_2
$\vdots$
str_n

n is a positive integer, which indicates the number of the following lines with the same length that represent the cell of an ASCII expression. str_k is the k-th line of the cell where each space character is replaced with a period.

You may assume that $n \leq 20$ and that the length of the lines is no more than 80.

Output

For each dataset, one line containing a non-negative integer less than 2011 should be output. The integer indicates the value of the ASCII expression in modular arithmetic under modulo 2011. The output should not contain any other characters.

There is no *fraction* with the divisor that is equal to zero or a multiple of 2011.

Note that *powexpr* x^0 is defined as 1, and x^y (y is a positive integer) is defined as the product $x \times x \times \cdots \times x$ where the number of x's is equal to y.

A fraction $\frac{x}{y}$ is computed as the multiplication of x and the inverse of y, i.e., $x \times \mathrm{inv}(y)$, under modulo 2011. The inverse of y ($1 \leq y < 2011$) is uniquely defined as the integer z ($1 \leq z < 2011$) that satisfies $z \times y \equiv 1 \pmod{2011}$, since 2011 is a prime number.

Sample Input

```
4
........4...2..........
(.1.-.----.)..*.-.5.+.6
........2..............
.......3...............
3
...3.
-.---
...4.
4
.3.+.4.*.-.2.
-------------
..........2..
...-.1.-.2...
2
...2..3
(.4..).
1
2.+.3.*.5.-.7.+.9
1
(.2.+.3.).*.(.5.-.7.).+.9
3
.2....3.
4..+.---
......5.
3
.2......-.-.3.
4..-.-.-------
..........5...
```

```
9
............1............
-------------------------
..............1..........
.1.+.-------------------.
................1........
......1.+.-------------..
..................1......
...........1.+.-------...
................1.+.2....
15
.................2......
................---.....
.......2.........5....3.
.(.---------.+.-----.)..
.....7...........3......
....---.+.1.............
.....4..................
------------------------
.......2................
......---...............
.......5.......2....2...
...(.-----.+.-----.)....
.......3.......3........
..............---.......
...............4........
2
.0....2....
3..+.4..*.5
20
............2............................2......................................
...........3............................3.......................................
..........----.........................----.....................................
............4............................4......................................
.....2.+.------.+.1...............2.+.------.+.1................................
............2............................2......................................
...........2............................2........................2..............
..........----.........................----.....................3...............
............2............................2.....................----.............
...........3............................3........................4..............
(.(.----------------.+.2.+.3.).*.----------------.+.2.).*.2.+.------.+.1.+.2.*.5
............2............................2.......................2..............
...........5............................5.......................2...............
..........----.........................----....................----.............
............6............................6.......................2..............
.........------.......................------....................3...............
............3............................3......................................
```

```
..........----.........................----.....................................
............2............................2......................................
...........7............................7.......................................
0
```

Output for the Sample Input

```
501
502
1
74
19
2010
821
821
1646
148
81
1933
```

Problem I
Encircling Circles

Input: Standard Input
Time Limit: 30 seconds

You are given a set of circles C of a variety of radii (radiuses) placed at a variety of positions, possibly overlapping one another. Given a circle with radius r, that circle may be placed so that it encircles all of the circles in the set C if r is large enough.

There may be more than one possible position of the circle of radius r to encircle all the member circles of C. We define the region U as the union of the areas of encircling circles at all such positions. In other words, for each point in U, there exists a circle of radius r that encircles that point and all the members of C. Your task is to calculate the length of the periphery of that region U.

Figure I.1 shows an example of the set of circles C and the region U. In the figure, three circles contained in C are expressed by circles of solid circumference, some possible positions of the encircling circles are expressed by circles of dashed circumference, and the area U is expressed by a thick dashed closed curve.

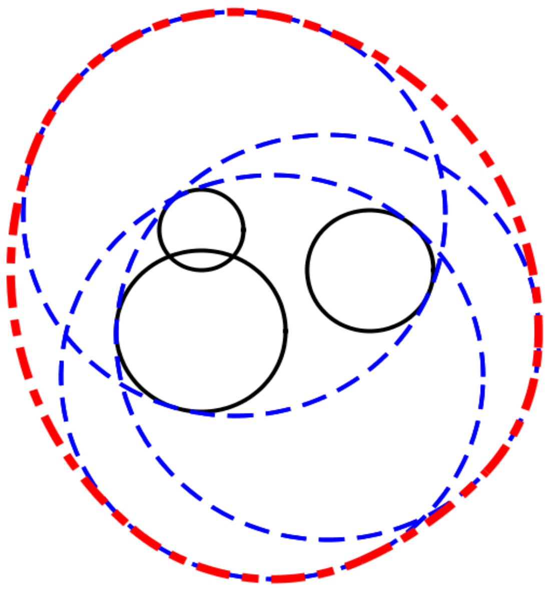

Figure I.1: Example of the Circle Set

Input

The input is a sequence of datasets. The number of datasets is less than 100.

Each dataset is formatted as follows.

$$\begin{array}{lll} n & r & \\ x_1 & y_1 & r_1 \\ x_2 & y_2 & r_2 \\ \vdots & & \\ x_n & y_n & r_n \end{array}$$

The first line of a dataset contains two positive integers, n and r, separated by a single space. n means the number of the circles in the set C and does not exceed 100. r means the radius of the encircling circle and does not exceed 1000.

Each of the n lines following the first line contains three integers separated by a single space. (x_i, y_i) means the center position of the i-th circle of the set C and r_i means its radius.

You may assume $-500 \leq x_i \leq 500$, $-500 \leq y_i \leq 500$, and $1 \leq r_i \leq 500$.

The end of the input is indicated by a line containing two zeros separated by a single space.

Output

For each dataset, output a line containing a decimal fraction which means the length of the periphery (circumferential length) of the region U.

The output should not contain an error greater than 0.01. You can assume that, when r changes by ϵ ($|\epsilon| < 0.0000001$), the length of the periphery of the region U will not change more than 0.001.

If r is too small to cover all of the circles in C, output a line containing only 0.0.

No other characters should be contained in the output.

Sample Input

```
1 10
5 5 7
2 12
5 5 7
8 6 3
3 10
3 11 2
```

```
2 1 1
2 16 3
3 15
-5 2 5
9 2 9
5 8 6
3 38
-25 -10 8
30 5 7
-3 35 11
3 39
-25 -10 8
30 5 7
-3 35 11
3 800
-400 400 2
300 300 1
300 302 1
3 800
400 -400 2
300 300 1
307 300 3
8 147
130 80 12
130 -40 12
-110 80 12
-110 -40 12
70 140 12
70 -100 12
-50 140 12
-50 -100 12
3 493
345 154 10
291 111 75
-275 -301 46
4 55
54 0 1
40 30 5
27 36 10
0 48 7
3 30
0 3 3
-3 0 4
400 0 3
3 7
2 3 2
-5 -4 2
```

```
-4 3 2
3 10
-5 -4 5
2 3 5
-4 3 5
4 6
4 6 1
5 5 1
1 7 1
0 1 1
3 493
345 154 10
291 111 75
-275 -301 46
5 20
-9 12 5
0 15 5
3 -3 3
12 9 5
-12 9 5
0 0
```

Output for the Sample Input

```
81.68140899333463
106.81415022205297
74.11215318612639
108.92086846105579
0.0
254.85616536128433
8576.936716409238
8569.462129048667
929.1977057481128
4181.124698202453
505.09134735536804
0.0
46.82023824234038
65.66979416387915
50.990642291793506
4181.124698202453
158.87951420768937
```

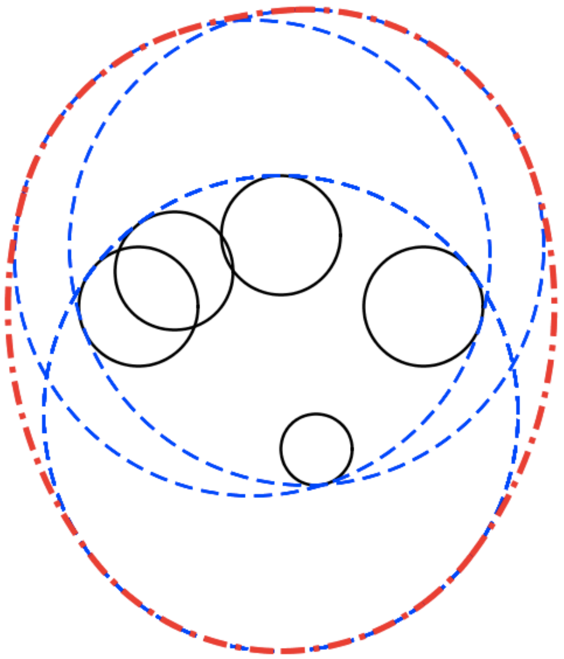
Figure I.2: Last Dataset of the Sample Input

Problem J
Round Trip

Input: Standard Input
Time Limit: 30 seconds

Jim is planning to visit one of his best friends in a town in the mountain area. First, he leaves his hometown and goes to the destination town. This is called the go phase. Then, he comes back to his hometown. This is called the return phase. You are expected to write a program to find the minimum total cost of this trip, which is the sum of the costs of the go phase and the return phase.

There is a network of towns including these two towns. Every road in this network is one-way, i.e., can only be used towards the specified direction. Each road requires a certain cost to travel.

In addition to the cost of roads, it is necessary to pay a specified fee to go through each town on the way. However, since this is the visa fee for the town, it is not necessary to pay the fee on the second or later visit to the same town.

The altitude (height) of each town is given. On the go phase, the use of descending roads is inhibited. That is, when going from town a to b, the altitude of a should not be greater than that of b. On the return phase, the use of ascending roads is inhibited in a similar manner. If the altitudes of a and b are equal, the road from a to b can be used on both phases.

Input

The input consists of multiple datasets, each in the following format.

$$\begin{array}{lll}
n & m & \\
d_2 & e_2 & \\
d_3 & e_3 & \\
\vdots & & \\
d_{n-1} & e_{n-1} & \\
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
 & \vdots & \\
a_m & b_m & c_m
\end{array}$$

Every input item in a dataset is a non-negative integer. Input items in a line are separated by a space.

n is the number of towns in the network. m is the number of (one-way) roads. You can assume the inequalities $2 \leq n \leq 50$ and $0 \leq m \leq n(n-1)$ hold. Towns are numbered from 1 to n, inclusive. The town 1 is Jim's hometown, and the town n is the destination town.

d_i is the visa fee of the town i, and e_i is its altitude. You can assume $1 \leq d_i \leq 1000$ and $1 \leq e_i \leq 999$ for $2 \leq i \leq n-1$. The towns 1 and n do not impose visa fee. The altitude of the town 1 is 0, and that of the town n is 1000. Multiple towns may have the same altitude, but you can assume that there are no more than 10 towns with the same altitude.

The j-th road is from the town a_j to b_j with the cost c_j $(1 \leq j \leq m)$. You can assume $1 \leq a_j \leq n$, $1 \leq b_j \leq n$, and $1 \leq c_j \leq 1000$. You can directly go from a_j to b_j, but not from b_j to a_j unless a road from b_j to a_j is separately given. There are no two roads connecting the same pair of towns towards the same direction, that is, for any i and j such that $i \neq j$, $a_i \neq a_j$ or $b_i \neq b_j$. There are no roads connecting a town to itself, that is, for any j, $a_j \neq b_j$.

The last dataset is followed by a line containing two zeros (separated by a space).

Output

For each dataset in the input, a line containing the minimum total cost, including the visa fees, of the trip should be output. If such a trip is not possible, output "`-1`".

Sample Input

```
3 6
3 1
1 2 1
2 3 1
3 2 1
2 1 1
1 3 4
3 1 4
3 6
5 1
1 2 1
2 3 1
3 2 1
2 1 1
1 3 4
3 1 4
4 5
3 1
3 1
1 2 5
2 3 5
3 4 5
4 2 5
```

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

Output for the Sample Input

```
7
8
36
-1
```