



CTU Open Contest 2007

Catch the Bus!

`bus.c`, `bus.C`, `bus.p`, `bus.java`

ACM needs to deliver marketing materials to one of their clients. Both ACM and the client employ students to make such deliveries. And these students use public buses to move throughout the city. Sometimes, it is necessary to pass the materials as fast as possible.

You are given bus timetables and your task is to find the fastest way for two students to meet at some stop. The place of the meeting is not important, they only need to meet as early as possible.

Students may change between any two bus routes at stops that are common for both routes. At least two minutes are needed for every such change. No additional time is necessary to get on the first bus or to meet the other student in the target stop.

Input Specification

The input contains a sequence of several scenarios, the sequence is terminated by a line containing negative number.

Each scenario begins with a non-negative integer L , the number of bus routes that operate in the city ($L \leq 1000$). Every route is then described by two lines. The first line contains names of stops that the bus runs through. Between each consecutive stops, there is a non-negative integer specifying the number of minutes required to travel between these stops with the given bus. The last stop is followed by a negative number.

The second line of each bus route contains non-negative integers separated with spaces. The first integer H gives the number of buses that depart the initial stop in every hour ($H \leq 60$). The remaining H integers are always distinct and sorted ascendingly, they list the minutes of departure (between 0 and 59). The timetable repeats every hour. For example, if the second line says “2 00 30”, the buses leave the initial stop at 12:00, 12:30, 13:00, 13:30, 14:00, etc.

After the description of routes, there are two lines that specify the initial position of students. Each of them will contain time in a standard 24-hour format (one or two digits for hours, colon, and two digits for minutes) and a stop name.

All numbers, times, and stop names will be separated with a single space. Stop names are case-sensitive and may be composed only from lower-case and upper-case letters, their length will not exceed 30 characters. The total number of stops will be at most 1000, the number of stops on a single route will not exceed 100. The time between any two consecutive stops will be at most one hour. The routes are considered one-way, if they operate in both directions, they will be given as two separate routes. A route may run through the same stop several times.

Output Specification

For each test scenario, output a single line containing the earliest possible time the students can meet at any of the stops. The time must appear in the standard 24-hour format, hours given as a number between 0 and 23, then colon and minutes between 0 and 59. For hour values less than 10, only one digit must be used. Be aware of the fact that both students start their trip on the same day but may meet on another day, if the required time exceeds midnight.

If the students are not able to meet in some scenario, output the words “No connection” instead of the time.

Sample Input

```
4
Hradcanska 2 Malostranska 2 Staromestska 2 Mustek 1 Muzeum -1
10 00 06 12 18 24 30 36 42 48 54
Muzeum 1 Mustek 2 Staromestska 2 Malostranska 2 Hradcanska -1
10 03 09 15 21 27 33 39 45 51 57
Andel 2 Karlovo 1 Narodni 2 Mustek 2 Florenc -1
6 00 10 20 30 40 50
Florenc 2 Mustek 2 Narodni 3 Karlovo 1 Andel -1
6 02 12 22 32 42 52
12:00 Hradcanska
12:11 Andel
1
Hradcanska 2 Malostranska 2 Staromestska 2 Mustek 1 Muzeum 2 Hradcanska -1
10 00 06 12 18 24 30 36 42 48 54
12:00 Mustek
12:00 Andel
-1
```

Output for Sample Input

```
12:20
No connection
```



CTU Open Contest 2007

Calendar of Events

`calendar.c`, `calendar.C`, `calendar.p`, `calendar.java`

Marketing people are very creative and must regularly attend many creative meetings. In the past, the meetings used to repeat every week. Now, there are so many of them, that the period must have been changed to one month.

The monthly schedule of ACM now contains one meeting for each of the first N days in every month (including weekends). Meetings are identified by non-negative integer numbers. Recently, ACM found out that the schedule must be reorganized and the order of meetings changed. The new order has already been chosen. The only problem is that the meeting-planning department has very limited privileges and must follow many directives that are in effect.

It fact, the only change that the department is allowed to do, is to issue a request to reverse the order of meetings in the first D days of the month (D may be selected arbitrarily). For example, if the current schedule is “1 2 3 4 5” and the department issues a request to reverse the first 3 days, the new schedule will be “3 2 1 4 5”.

Another limitation is that the department is allowed to issue only one request per week. Since they need the whole reorganization to finish in one year, the number of requests may not exceed 52. As you can see, their situation is not easy. Will you help?

Input Specification

The input contains a sequence of several reorganizations, the last one followed by a line with zero.

Each reorganization consists of three lines: the first line contains a positive integer N — the number of days in month that are considered, $1 \leq N \leq 30$. The second line describes the old schedule and the third line the new schedule, each of them contains N meeting identifiers, separated with spaces. Some meetings may appear more than once.

Output Specification

For each reorganization, output a single line containing at most 52 positive integer numbers $D_1, D_2, \dots, D_k$ separated by a space ($1 \leq D_i \leq N$). The numbers must form a valid sequence of requests that will be issued to transform the old schedule to the new one.

You are guaranteed that such a sequence will exist. If there are more valid sequences, feel free to output any of them.

Sample Input

```
5
1 4 8 9 10
4 8 9 10 1
0
```

Output for Sample Input

```
5 4
```



CTU Open Contest 2007

Roman Expressions

`expr.c`, `expr.C`, `expr.p`, `expr.java`

As any other marketing company, ACM produces a lot of funky advertising items that may contain a logo and be given to customers and business partners as small gifts. A unique specialty of ACM is a calculator that uses roman numbers.

Roman numbers are able to express any non-negative integer using uppercase letters:

Letter	Value
O	0
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Numbers are created by appending several letters together, the letter representing a higher value must always precede letters with lower values. The only exception are the letters “I”, “X”, and “C”, they may be used before higher letters to form values expressed by digits 4 and 9. The only possible combinations are:

Pair of Letters	Value
IV	4
IX	9
XL	40
XC	90
CD	400
CM	900

Any roman number must first express thousands, then hundreds, tens, and ones. Therefore, 499 must always be written as “CDXCIX”, not “ID”.

Although not very practical, this gift is considered extremely “coooooool”. Your task is to write software for that calculator.

Input Specification

The input will consist from commands, each written on a separate line. Possible commands are assignments, “RESET”, and “QUIT”.

An assignment command starts with a single digit representing one of ten registers that the calculator has. The register number is followed by an equal sign (“=”) and an expression. The expression will consist only from valid roman numbers, register names (digits), plus (“+”) and minus (“-”) signs. You may assume that the expression will always be valid and no longer than 10000 characters.

Output Specification

For each command, output a single line. For assignments, print the register name, equal sign, and the value that is being assigned to that register. Instead of it, print the word “Error”, if the expression contains a reference to a register that has not been assigned before, or if the result is negative or larger than 10000. In such cases, no change to register values is made.

For RESET commands, invalidate all previous register assignments and print the word “Ready”.

The QUIT command will be the last one. Print the word “Bye” and terminate the program.

Sample Input

```
1=MC+IV-X
1=1+1
RESET
1=1+X
1=MM
1=1+1+1+1+1
2=1+1
QUIT
```

Output for Sample Input

```
1=MXCIV
1=MMCLXXXVIII
Ready
Error
1=MM
1=MMMMMMMMMM
Error
Bye
```



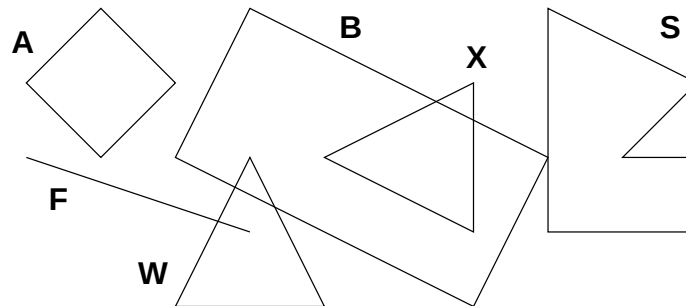
CTU Open Contest 2007

Geometric Shapes

`geometry.c`, `geometry.C`, `geometry.p`, `geometry.java`

While creating a customer logo, ACM uses graphical utilities to draw a picture that can later be cut into special fluorescent materials. To ensure proper processing, the shapes in the picture cannot intersect. However, some logos contain such intersecting shapes. It is necessary to detect them and decide how to change the picture.

Given a set of geometric shapes, you are to determine all of their intersections. Only outlines are considered, if a shape is completely inside another one, it is not counted as an intersection.



Input Specification

Input contains several pictures. Each picture describes at most 26 shapes, each specified on a separate line. The line begins with an uppercase letter that uniquely identifies the shape inside the corresponding picture. Then there is a kind of the shape and two or more points, everything separated by at least one space. Possible shape kinds are:

- **square**: Followed by two distinct points giving the opposite corners of the square.
- **rectangle**: Three points are given, there will always be a right angle between the lines connecting the first point with the second and the second with the third.
- **line**: Specifies a line segment, two distinct end points are given.
- **triangle**: Three points are given, they are guaranteed not to be co-linear.
- **polygon**: Followed by an integer number N ($3 \leq N \leq 20$) and N points specifying vertices of the polygon in either clockwise or anti-clockwise order. The polygon will never intersect itself and its sides will have non-zero length.

All points are always given as two integer coordinates X and Y separated with a comma and enclosed in parentheses. You may assume that $|X|, |Y| \leq 10000$.

The picture description is terminated by a line containing a single dash (“-”). After the last picture, there is a line with one dot (“.”).

Output Specification

For each picture, output one line for each of the shapes, sorted alphabetically by its identifier (X). The line must be one of the following:

- “ X has no intersections”, if X does not intersect with any other shapes.
- “ X intersects with A ”, if X intersects with exactly 1 other shape.
- “ X intersects with A and B ”, if X intersects with exactly 2 other shapes.
- “ X intersects with A , B , ..., and Z ”, if X intersects with more than 2 other shapes.

Please note that there is an additional comma for more than two intersections. A , B , etc. are all intersecting shapes, sorted alphabetically.

Print one empty line after each picture, including the last one.

Sample Input

```
A square (1,2) (3,2)
F line (1,3) (4,4)
W triangle (3,5) (5,5) (4,3)
X triangle (7,2) (7,4) (5,3)
S polygon 6 (9,3) (10,3) (10,4) (8,4) (8,1) (10,2)
B rectangle (3,3) (7,5) (8,3)
-
B square (1,1) (2,2)
A square (3,3) (4,4)
-
.
```

Output for Sample Input

```
A has no intersections
B intersects with S, W, and X
F intersects with W
S intersects with B
W intersects with B and F
X intersects with B

A has no intersections
B has no intersections
```



CTU Open Contest 2007

Corporate Identity

`identity.c`, `identity.C`, `identity.p`, `identity.java`

Beside other services, ACM helps companies to clearly state their “corporate identity”, which includes company logo but also other signs, like trademarks. One of such companies is Internet Building Masters (IBM), which has recently asked ACM for a help with their new identity. IBM do not want to change their existing logos and trademarks completely, because their customers are used to the old ones. Therefore, ACM will only change existing trademarks instead of creating new ones.

After several other proposals, it was decided to take all existing trademarks and find the longest common sequence of letters that is contained in all of them. This sequence will be graphically emphasized to form a new logo. Then, the old trademarks may still be used while showing the new identity.

Your task is to find such a sequence.

Input Specification

The input contains several tasks. Each task begins with a line containing a positive integer N , the number of trademarks ($2 \leq N \leq 4000$). The number is followed by N lines, each containing one trademark. Trademarks will be composed only from lowercase letters, the length of each trademark will be at least 1 and at most 200 characters.

After the last trademark, the next task begins. The last task is followed by a line containing zero.

Output Specification

For each task, output a single line containing the longest string contained as a substring in all trademarks. If there are several strings of the same length, print the one that is lexicographically smallest. If there is no such non-empty string, output the words “IDENTITY LOST” instead.

Sample Input

```
3
aabbbaabb
abbababb
bbbbbbabb
2
xyz
abc
0
```

Output for Sample Input

```
abb
IDENTITY LOST
```



CTU Open Contest 2007

Money Money Money, Must Be Funny

`money.c`, `money.C`, `money.p`, `money.java`

- **Shopkeeper:** “100 crowns and 80 hellers, please.”
- **Customer:** “Here you are.” (handing a 200-crown banknote)
- **Shopkeeper:** “Would you have 80 hellers?” (passing back a 100-crown note)
- **Customer:** “No, but here you are.” (adding a 1-crown coin)
- **Shopkeeper:** “Sorry, but have only this.” (showing a little shiny 50-heller coin)
- **Customer:** “So I need to give you another 30 heller. But I have only 40.”
- **Shopkeeper:** “That’s ok, here are the remaining 10.”

Have you ever experienced similar situations? Paying a precise amount can sometimes be difficult, if the set of available coins and banknotes (“tenders”) is limited. The situation above was finally solved: The customer paid $200+1$ crowns, got $100+0.50$ back, paid another $0.20+0.20$, and finally got 0.10 back. This means, 7 tenders had to be exchanged. Sometimes, it may be even more complicated. Your task is to write a program that solves situations like this.

Input Specification

Input contains several tasks to be solved. Each task begins with a line containing one non-negative number: the amount to be paid. Then there is a list of tenders possessed by the customer (the one who pays). Each line in the list contains the tender nominal value (non-negative number), one space, number of tenders of that value (non-negative integer), and the lowercase letter “x”. The list is terminated by a line containing number “-1”.

After the first list, there is a list of tenders possessed by the shopkeeper (the one who gets paid). The second list has the exactly same format as the first one.

Then the next task begins. The last task is followed by one more line containing “-1”.

You may assume that each list will contain at most 100 lines, and nobody will have more than 10 000 units and/or 500 tenders. Nominal values can be arbitrary, they do not need to follow any existing scheme valid in known countries. All numbers that allow non-integer values will be given either as integers or as decimal numbers with one or two digits after the decimal point.

Output Specification

For each task, output one line containing the sentence “*X* tenders must be exchanged.”, with *X* replaced by the minimal number of tenders that are to be used to pay the required amount. If this is not possible at all, output the sentence “The payment is impossible.” instead.

Sample Input

100.80
500 1x
200 3x
1.00 10x
0.20 2x
-1
500 10x
200 12x
100 8x
0.10 1x
0.20 0x
0.50 100x
20 2x
-1
200
10 19x
-1
200 1x
-1
-1

Output for Sample Input

7 tenders must be exchanged.
The payment is impossible.

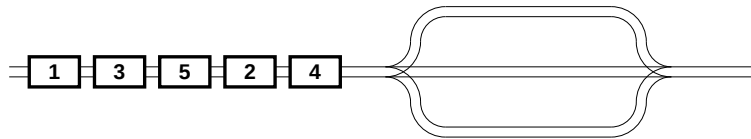


CTU Open Contest 2007

Railway Transportation

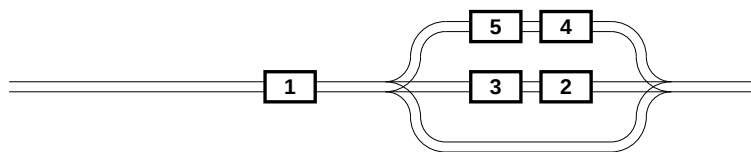
`railway.c`, `railway.C`, `railway.p`, `railway.java`

ACM has created a new series of huge billboards. Now, they have to be transported using a train. Unfortunately, the carriages of the train get somehow mixed, but ACM wants the billboards to be delivered in an appropriate order.



In a small example, we have 5 carriages approaching a railroad station with 3 parallel tracks. Each carriage may enter any track, but once it is there, it cannot go back anymore. At the other end, the tracks are merged back to a single railroad and cannot go back, again. We want the carriages to leave their tracks in the correct order.

Possible solution is illustrated in the second figure. We may send carriage 4 to the first track, then the carriage number 2 to the second one, number 5 to the first, number 3 to the second, and finally, number 1 to the third track, which may immediately leave it to its correct position. The other carriages will follow as needed.



Your task is to write a program that sends carriages to correct tracks and merges them at the other end. You may assume that the tracks are long enough to hold any number of carriages.

Input Specification

The input consists of several scenarios, each of them described on two lines. End of the input is signalized by two zeros.

The first line of each scenario contains two positive integers N and M , separated with a space, $N \leq 200000$, $M \leq 200000$. N is the number of carriages, M is the number of tracks.

The second line will contain N non-negative integers representing carriages and their intended order on the other end of the station. Some carriages may have identical numbers, in such a case, their mutual order is not significant.

Output Specification

For each scenario, output two lines. The first line will contain N numbers, each of them between 1 and M inclusive. Each number corresponds to one carriage (in the order they were given in the input) and determines a track the carriage is to be moved to. The second line also contains N numbers, they specify from what tracks and in which order are the carriages sent to the other end of the station.

If carriages are manipulated according to your answer, they must leave the station in a non-descending order. When there are more solutions, you may print any of them. If there is no solution, print only one line containing the words “**Transportation failed**”.

Sample Input

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

Output for Sample Input

```
1 2 1 2 3
3 2 2 1 1
Transportation failed
```



CTU Open Contest 2007

Soccer Tournament

`soccer.c`, `soccer.C`, `soccer.p`, `soccer.java`

ACM decided to run a big advertising campaign. Beside others, it will sponsor soccer tournaments by providing them with a system for computing and displaying game results. You are to write that system.

Input Specification

The input contains description of several tournaments. Each tournament begins with a line containing single positive integer N : the number of participating teams, $N \leq 80$. Then there are N lines, each containing a unique name of one team. The name is case-sensitive and may be composed only from letters, digits, dots (“.”), and dashes (“-”). No name will be longer than 100 characters.

After the team names, there will be a line with a single non-negative integer M : the number of games that have already been played. Each of the next M lines describes one game and contains the host team name, space, dash, space, guest team name, space, and the game result. The result will always appear as a single digit (0-9), colon (“:”), and another single digit (0-9). The digits specify the number of goals scored by the host and guest teams.

Then, the next tournament is described. The last tournament is followed by a zero on a separate line. No output should be produced for this zero.

You may assume that all team names have been listed among the N teams. All games will have distinct opponent pairs. This also means that any two teams may play at most two games with each other — in such a case, each of the two teams will be a host in one game and a guest in the other.

Output Specification

For each tournament, output a line containing the word “RESULTS” and a colon (“:”), then a result table, empty line, a line with the word “STANDINGS” and a colon, a line with 10 dashes (“-”), then current standings, and another empty line.

The result table is “drawn” using dashes (“-”), pipes (“|”), and plus signs (“+”). The table has $N + 1$ rows and $N + 1$ columns. The upper left cell is always empty (filled with spaces). Otherwise, the first column contains full team names, the width of the column must exactly fit the longest name. The first row contains first three characters of team names (followed by one or two spaces if the name was shorter than 3 characters). All teams must be given in the same order as they appeared in the input list of teams.

Diagonal cells (except the first row and column) contain centered uppercase letter “X”. All other cells are either empty (three spaces) or contain game results for the games that have already been played. Hosting team is in the row, guest team in the column.

The standings consist of N output lines, each of them representing one team. The lines form 8 aligned columns separated with a single space character. The columns must be perfectly aligned, which means that the column occupies the same number of characters in each row. The number of characters must be equal to the longest data in that column. Other rows will be padded with spaces on the left (columns are right-aligned). The only exception is the second column (team names), which is padded from right (left-aligned).

The contents of individual columns are as follows:

1. Rank of the team (number of the line, starting with one) and a dot (“.”).
2. Team name.
3. Number of games played by the team.
4. Number of games won (the team scored more goals than the opponent).
5. Number of tied games (both teams scored the same number of goals).
6. Number of games lost (the other team scored more goals).
7. Score consisting of three parts: number of goals scored by the team in all its games, colon (“:”), and number of goals scored by opponents in all games of the team.
8. Number of points achieved. Each team gets one point for a tie and three points for a victory. There are no points for lost games.

The standings must be sorted by the following criteria, in the given order:

1. Points, the team with more points is listed first.
2. Score difference (number of goals scored minus the number of goals scored by opponents), the team with a higher difference is given first.
3. Number of goals scored by the team, the one that scored more is listed first.
4. Number of games won. The team with more victories will be first.
5. *Reversed* order in the input. The team that was given later in the input list will be ranked better.

Refer to the sample output below whenever the format is ambiguous or unclear.

Sample Input

```
4
Slavia
Arsenal
Steaua
FCSevilla
4
Slavia - Arsenal 2:0
FCSevilla - Steaua 1:1
Steaua - Slavia 1:2
Arsenal - Slavia 0:0
0
```

Output for Sample Input

```
RESULTS:
+-----+---+---+---+---+
|          |Sla|Ars|Ste|FCS|
+-----+---+---+---+---+
|Slavia   | X |2:0|   |   |
+-----+---+---+---+---+
|Arsenal  |0:0| X |   |   |
+-----+---+---+---+---+
|Steaua   |1:2|   | X |   |
+-----+---+---+---+---+
|FCSevilla|   |   |1:1| X |
+-----+---+---+---+---+

STANDINGS:
-----
1. Slavia      3 2 1 0 4:1 7
2. FCSevilla  1 0 1 0 1:1 1
3. Steaua      2 0 1 1 2:3 1
4. Arsenal     2 0 1 1 0:2 1
```



CTU Open Contest 2007

Hypertheseus

`theseus.c`, `theseus.C`, `theseus.p`, `theseus.java`

For one of their clients, ACM prepares a TV advertisement that features ancient heroes, for example Prometheus, Achilles, Odysseus and many others. To demonstrate how difficult was life for these heroes, ACM runs computer simulations of their most famous tasks. In this problem, we will try to solve the problem of Theseus.

Theseus was an Athenian hero who killed Minotaur, the half-man, half-bull monster residing in an inescapable Labyrinth constructed by Daedalus. The hardest part of Theseus' task was not killing the monster — the legend says Minotaur was just sleeping when Theseus came to him. Thus, the hero was able to batter the monster with his bare hands. But then the *real* challenge came: to find the way out. As you may know, Ariadne, beautiful daughter of king Minos, played an important role in this part. But that's a different story.

As the technology advanced a lot since the ancient times, there are several important differences in today's simulations:

1. Labyrinth-making skills advanced as well. Two-dimensional labyrinths are not a big challenge anymore. Thus, our labyrinth is d -dimensional. It looks like a regular "grid" of $n_1 \times n_2 \times \dots \times n_d$ (hyper)cubes, each of them being either empty (corridor) or filled with a rock (wall). Theseus moves in steps, each step means travelling between two neighboring empty hypercubes in any dimension. Two hypercubes are considered neighboring, if and only if the difference in their space coordinates is exactly one in one dimension, and zero in all other dimensions.
2. Our Minotaur is stronger, thus, it is no more possible to batter him with bare hands. Theseus must use a sword which lies somewhere inside the labyrinth. Note that before he takes the sword, he cannot go through the hypercube where Minotaur resides!

Input Specification

The input consists of several labyrinth descriptions. Each description begins with a line containing a single integer number d ($2 \leq d \leq 20$) — the number of dimensions. On the second line of each description, there are d integer numbers $n_1, n_2, \dots, n_d$ separated by spaces. These numbers give the size of the labyrinth in every dimension, measured in unit hypercubes ($\forall i : 2 \leq n_i$). You may assume that the total number of hypercubes in any labyrinth will not exceed $2^{20} = 1048576$.

Then a labyrinth map follows, the description is defined recursively: Two-dimensional labyrinth (of the size $n_1 \times n_2$) is described by n_2 lines containing n_1 characters each. Every character describes a single hypercube ("square" in the case of 2 dimensions) and is either "#" (rock), "." (free space), or one of uppercase letters "T" (Theseus' position), "M" (Minotaur), and 'S' (sword). Each of these three objects will appear exactly once in the whole labyrinth. The hypercubes containing the letters are considered "empty" otherwise, i.e., Theseus can walk (and must walk) through them.

For better clarity of the input, each two-dimensional map is followed by one empty line.

For $d > 2$, any d -dimensional labyrinth is a sequence of n_d "layers", each layer being a description of a $(d - 1)$ -dimensional labyrinth.

The last labyrinth description is followed by a line containing zero. This line terminates the input and no output should be produced for it.

Output Specification

For each labyrinth, output the sentence "Theseus needs S steps.", where S is the smallest possible amount of steps needed to reach the hypercube with the sword, the hypercube with Minotaur, and then back to the original Theseus' position (where the exit is supposed to be). It is not possible to leave the labyrinth area, i.e., all hypercubes outside the labyrinth are considered rocks. Exit may appear in an inner hypercube, it may not reside on the "border".

If it is not possible to solve the situation at all, output sentences "No solution. Poor Theseus!"

Sample Input

```
2
3 3
T..
M#.
S..
```

```
2
3 2
T#S
..M
```

```
3
4 4 3
#S..
###.
#M..
###.
```

```
.###
.###
.###
....
```

```
....
###.
###T
####
```

```
0
```

Output for Sample Input

```
Theseus needs 8 steps.
No solution. Poor Theseus!
Theseus needs 40 steps.
```