2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
Epithets Among Enemies

The Swamp County Computer User's Group has decided to raise funds by distributing a free mobile app that is supported by advertising. A committee formed from unemployed adolescent programmers has decided to produce a networked game that uses letter tiles to form words on a rectangular grid game board. To distinguish the SCCUG game from the many popular word game alternatives, the dictionary of allowed words will be expanded to include obscene and vulgar terms. Additionally, tile frequencies and letter point values will be altered to reward the use of foul language. The game will be titled "Epithets Among Enemies." The committee is currently immersed in Tosh.0 and South Park archives, researching contemporary English profanity. The results will be applied to tile frequencies and letter points to maximize the offensiveness of game play. While the research is ongoing, your team must code the word scoring engine.

Input to your program has two parts: the letter point values followed by the words to score. The first part contains three lines specifying the point values for the letters A through Z and the "wild card" blank tile in that order. Each line will contain nine integer values in the range 0–99 inclusive separated by whitespace. No input line will exceed 80 columns.

The remaining input consists of two-line pairs: the first line describes the squares (normal or bonus) that the word is played on, and the second line contains the letter tiles themselves. The blank wild card tile is represented in the input by an underscore. Words will not be more than 30 characters long. The last two-line pair will be followed by the end-of-file.

The square description uses dots (".") to indicate normal squares. Bonus squares are marked with a "2" to indicate that the value of the letter played on that square is to be doubled, a "3" to indicate that the values of the letter played on that square is to be tripled, a "d" indicates that the total word score is to be doubled, and a "t" indicates that the total word score is to be tripled. Apply the "2" or "3" bonus only to the corresponding tile. Sum the points in the word, then apply each "d" and "t" bonus to the word point total. For example, one occurrence of "d" multiplies the word value by 2. Two occurrences of "d" multiplies the word value by 4. The presence of a "d" and "t" in the squares covered by a word multiplies the word value by 6, and so on. Final word scores will be non-negative integers less than 10^6.

For each word, print the final word score on a line by itself beginning in the first column without trailing whitespace.

Your program is not to be concerned with game rules or order of play. Each word will be legal and should be scored independently.

Sample Input

```
5 3 2 1 1 20 6 3 7
2 4 3 2 1 2 4 1 1
2 1 15 4 2 1 2 1 0
....
SCAT
2...
DARN
.....
FIEND
.d..2.3t
COERCI_E
```

Problem 1
Epithets Among Enemies (continued)

Output for the Sample Input

```
10
9
30
108
```

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Full Board

"Full Board" is an interactive one-person puzzle/game by LightForce played on a rectangular grid.

The game starts with a board of M rows by N columns with some grid squares marked as "obstacles" (drawn as black dots). The player chooses a starting position to place a ball (marked "S" in Figure 1) and chooses a direction to advance (left, right, up, or down). Once a direction is chosen, the ball will advance in that direction until it hits an obstacle, the boundary of the game board, or a square that the ball has already been through. The player then chooses another direction, and the ball will advance in the same manner. The game ends when no legal move can be made. The player wins if and only if the ball has traveled through all the empty grid squares on the board.

Given initial settings for a series of game boards, your team is to write a program to calculate the minimum number of steps needed to win each game from an optimal starting position. The input for each game board begins with a line containing two integers M and N, separated by whitespace, giving the dimensions of the game board. M and N will be in the range 1–50 inclusive. The next M lines describe the initial setting of the board. Each line contains N characters, each of which is either an asterisk or a dot, indicating that the corresponding grid square is an obstacle or empty, respectively. It is guaranteed that the initial board is not fully covered by obstacles. The last game board in the input will be followed by the end-of-file.

For each game board, your program is to print a line containing a single integer indicating the minimum number of steps to win the game. No leading or trailing whitespace is to appear on an output line. If it is not possible to win a given game, print the value -1.

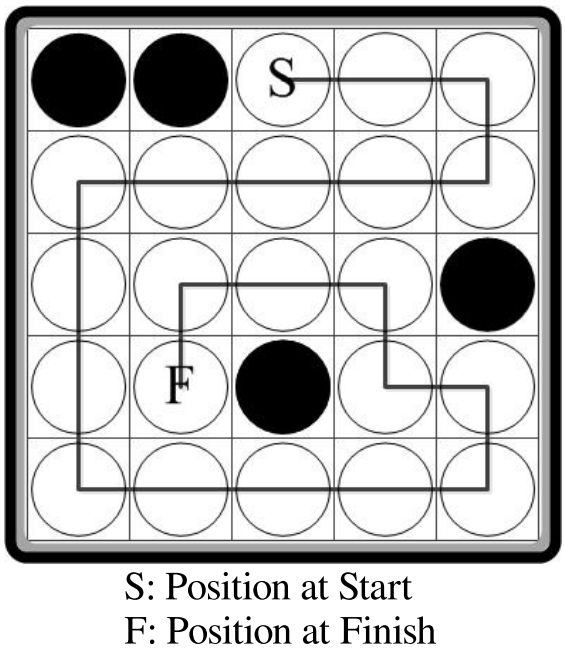

Figure 1. Game board used in the sample input.

Problem 2
Full Board (continued)

Sample Input

```
5 5
**...
.....
....*
..*..
.....
3 4
****
*...
*..*
```

Output for the Sample Input

```
10
3
```

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
CART

To celebrate the fiftieth anniversary of computing at Swamp County College the Computing Department wants to reproduce some of the early programs that the department developed.

Your team has been selected to re-implement CART, a simple text processing language. See the attached manual for the language definition. Your job is to write the interpreter for CART.

Sample Input

```
#<rs;//>
##<def;badloop;
  <
    ##<ps;#1>
    #<numgt;#1;1;<#<badloop;##<sub;#1;1>;>>>
> >
##<def;goodloop;
  <
    ##<ps;#1>
    #<numgt;#1;1;<#<goodloop;##<sub;#1;1>;>>>
  >
>
//
##<tn>
#<goodloop;3>
#<badloop;3>
##<tf>
##<ps;goodby>
```

Problem 3
CART (continued)

Output for the Sample Input

```
+##<ps;#<goodloop;3>>+
+#<goodloop;3>>+
+##<ps;3>#<numgt;3;1;<#<goodloop;##<sub;3;1>;>>>>+
3
+#<numgt;3;1;<#<goodloop;##<sub;3;1>;>>>>+
+#<goodloop;##<sub;3;1>;>>+
+##<sub;3;1>;>>+
+##<ps;2>#<numgt;2;1;<#<goodloop;##<sub;2;1>;>>>>+
2
+#<numgt;2;1;<#<goodloop;##<sub;2;1>;>>>>+
+#<goodloop;##<sub;2;1>;>>+
+##<sub;2;1>;>>+
+##<ps;1>#<numgt;1;1;<#<goodloop;##<sub;1;1>;>>>>+
1
+#<numgt;1;1;<#<goodloop;##<sub;1;1>;>>>>+
+##<ps;#<badloop;3>>+
+#<badloop;3>>+
+##<ps;3>#<numgt;3;1;<#<badloop;##<sub;3;1>;>>> >+
3
+#<numgt;3;1;<#<badloop;##<sub;3;1>;>>> >+
+#<badloop;##<sub;3;1>;> >+
+##<sub;3;1>;> >+
+##<ps;2>#<numgt;2;1;<#<badloop;##<sub;2;1>;>>>  >+
2
+#<numgt;2;1;<#<badloop;##<sub;2;1>;>>>  >+
+#<badloop;##<sub;2;1>;>  >+
+##<sub;2;1>;>  >+
+##<ps;1>#<numgt;1;1;<#<badloop;##<sub;1;1>;>>>   >+
1
+#<numgt;1;1;<#<badloop;##<sub;1;1>;>>>   >+

+##<ps;##<tf>>+
+##<tf>>+
goodby
```

CART
Compile and Reckon Text

Swamp County College Programming Report No. 8
May 1969

Introduction

CART is a simple string processing language. In general, it copies input to output until it processes certain significant characters, namely '#', '<', '>', ';', and '@'. The sequences "#<" and "##<" start a function call and are followed by any number of parameters separated by ';' characters. The first parameter (parameter 0) is the name of the function. The function call is ended by the matching '>'.

Within a parameter there may be another function call that will be executed while collecting the parameter. There are times when an inner function call (or what looks like the start of one) should not be executed. There are two mechanisms for this: the '@' character can be used to prevent the recognition of the immediately following character or a string may be enclosed in matching brackets '<' '>'.

There are two types of functions: primitive functions which are built in to CART and defined functions which are created using the 'def' primitive.

There are two types of function calls: "#<" is an active call and "##<" is a passive call. The results of a passive call are simply appended to the parameter being collected while the results of an active call are processed as if they were the next part of the input.

If a function is called with more parameters than expected, the excess parameters are ignored. If called with less, the missing parameters are the empty string, "", a string with 0 length.

If an undefined function is called, say foo as in "#<foo>", the result is "!no such function:foo!".

Processing Algorithm

CART is intended to be embedded in other applications. This version reads a line of input, discards any end-of-line characters, prepends "##<ps;" to the line, appends a '>' to the line, copies the result into the CART program-string, and starts the CART engine on it.

The CART processing engine consumes characters from the program-string from left to right. It has three states: scan, parameter collection, and balanced bracket collection.

In the scan state it simply discards characters until "#<" or "##<" are encountered and then it enters parameter collection.

During parameter collection if "#<" or "##<" are encountered the inner function call is processed recursively. A ';' separates each parameter. The escape character, '@', may be used to prevent the significance of any special character: the '@' will be deleted and the next character included in the parameter without examination. For instance, "#<ps;foo@;>" will print 'foo;'. When a '<' is encountered balanced bracket collection will take place. When a '> that matches the initial "#<" or "##<" is seen the function call is executed. During a defined function call, all sequences of the form '#0' through '#9' will be replaced by the respective parameter. If the call was passive ("##<") the result, if any, will be appended to the parameter being collected at the outer level. If the call was active ("#<") the result will be prepended to the program-string and will be processed.

During a balanced bracket scan only '<', '>', and '@' are examined. The escape character will protect the following character but it will not be deleted. All the characters between the matching '<' and '>' will be appended to the parameter being scanned.

Primitive Functions

Utility

tn: turn trace on
#<tn>
result: the empty string
effect: Before collecting the parameters of a function call, print the program sting, including the "#<" or "##<", preceded and followed by '+' characters.

tf: turn trace off
#<tf>
result: the empty string
effect: turn off tracing.

define: create or replace a defined function
#<def;string1;string2>
result: the empty string
effect: Create or replace the function with name string1 with value string2. If string1 is the name of a primitive function, that primitive will be hidden, that is the defined function will be called, not the primitive function.

undef: remove a defined function
#<undef;string1>
result: the empty string
effect: If there is a defined function of name string1, it is deleted. There is no way to remove a primitive function. If the removed defined function was hiding a primitive function that primitive will be active again.

isdef: check if a function is defined
#<isdef;string1;string2;string3>
result: if string1 is the name of a defined function, string2, else string3.

ps: print string
#<ps;string1>
result: the empty string
effect: if string1 is not the empty string, it is printed on a separate line.

rs: read string
#<rs;string1;string2>
result: if string1 is the empty string, the next line in the input with any system dependent end-of-line characters removed and string2 appended. If string1 is not the empty string, the concatenation of the next lines in the input with leading and trailing white space removed from each, any end-of-line characters removed, and string2 appended to each until either the end-of-file or the start of the line matches string2. That line is -not- included.

Arithmetic

A number in CART is a string consisting of only decimal digits '0123456789' and, if the number is negative, a leading '-'. Primitive functions returning numbers will have no unnecessary leading '0's. If there is an error, one or more parameters not being a number or a division or remainder by 0, '!Math Error!' will be returned. The number range is implementation dependent but will be at least -30000..30000.

add: add
#<add;num1;num2>
result: num1 + num2 or '!Math Error!'

sub: subtract
#<sub;num1;num2>
result: num1 - num2 or '!Math Error!'

mul: multiply
#<mul;num1;nu2>
result: num1 * num2 or '!Math Error!'

```
  div: divide
  #<div;num1;num2>
  result: num1 / num2 or '!Math Error!'
  Integer division truncates, it rounds toward 0 (-15/4 => -3).

  rem: remainder
  #<rem;num1;num2>
  result: num1 % num2 or '!Math Error!'.

  abs: absolute value
  #<abs;num1>
  result: absolute value of num1 or '!Math Error!'

  neg: negate
  #<neg;num1>
  result: -num1 or '!Math Error!'

  numeq: numeric equality
  #<numeq;num1;num2;string1;string2>
  result: if (num1 == num2) string1 else string2, or '!Math Error!'

  numlt: numeric less than
  #<numlt:num1;num2;string1;string2>
  result: if (num1 < num2) string1 else string2, or '!Math Error!'

  numgt: numeric greater than
  #<numgt;num1;num2;string1;string2>
  result: if (num1 > num2) string1 else string2, or '!Math Error!'

String Utilities
  The index of the leftmost character in a string is 0.

  len: length of a string
  #<len;string1>
  result: the number of characters in string1

  substr: get a substring
  #<substr;num1;num2;string1>
  result: the substring of string1 starting at num1 of at most
    length num2. If num1 or num2 are not positive numbers, return the
    empty string.

  isc: initial scan
  #<isc;string1;string2;string3;string4>
  result: if the start of string1 matches all of string2, string3,
    else string4.

  find: find a substring
  #<find;string1;string2>
  result: the index of the first character in string1 of the first
  substring that matches string2. It is -1 if there is no match or string1
  or string2 is the empty string.

  flip: flip a string
  #<flip;string1>
  result: the reverse of string1

  eq: compare strings
  #<eq;string1;string2;string3;string4>
  result: if string1 is equal to string2, string3 else string4

  lt: less than
  #<lt;string1;string2;string3;string4>
  result: if string1 is lexicographically less than string2, string3
    else string4

  gt: greater than
  result: if string1 is lexicographically greater than string2, string3
    else string4
```

Character Scanning Utilities
  The first argument of these primitives is taken as a list of characters.

  callcl: call class
  #<callcl;class;string1>
  result: the initial part of string1 that consists of characters only
    in class.

  callncl: call negative class
  #<callncl;class;string1>
  result: the initial part of string1 that consists of characters -not-
    in class.

  skipcl: skip class
  #<skipcl;class;string1>
  result: the final part of string1 after skipping initial characters in
    class.

  skipncl: skip negative class
  #<skipncl;class;string1>
  result: the final part of string1 after skipping initial characters -not-
    in class

  allcl: all in class
  #<allcl;class;string1;string2;string3>
  result: if all of the characters in string1 are in class, string2, else
    string3

  firstcl: first in class
  #<firstcl;class;string1;string2;string3>
  result: if the first character of string1 is in class, string2, else
    string3

                              Examples

#<ps;ab;cd>   => ab
#<ps;ab@;cd>   => ab;cd
#<ps;<ab@;cd>ef>   => ab@;cdef
#<ps;<ab@>cd>ef>   => ab@>cdef
<ab@>cd>ef   => ab@>cdef
#<def;x;<##<add;1;2>>   =>
#<ps;##<x>>   => ##<add;1;2>
#<ps;#<x>>   => 3
#<ps;#<callcl;ezarf;eazy living>>   => eaz

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Collatz Sequence

In 1937 the mathematician L. Collatz posed the following problem:

Given an integer $a_0 > 0$ and the rule
if a_i is even
$a_{i+1} = a_i \ / \ 2$
else
$a_{i+1} = 3 \times a_i + 1$
does the sequence always reach $a_n = 1$ for some n? Note that if the sequence is continued after reaching 1, the sequence repeats: 1, 4, 2, 1, 4, ...

So far no one has proved that the conjecture is either true or false or found a counterexample that gets stuck in a repeating sequence other than 4, 2, 1, ... Your team is to write a program that will explore the conjecture for assorted values of a_0.

Input is a series of unsigned integers greater than zero, one per line, terminated by end-of-file. Each input line has no leading zeroes or leading or trailing spaces. Each integer consists of at most 80 digits.

For each value of a_0, your program is to print a line containing n, the iteration number where the sequence is first equal to 1. No leading zeroes, leading whitespace, or trailing whitespace are to appear on a printed line.

Sample Input

```
1117065
1
6
27
1341234558
9780657630
2361198062777205778683
309308933983492348683495812819483581648162841841846814814681248164814681481486614814866
```

Output for the Sample Input

```
527
0
8
111
987
1132
2240
1755
```

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
BiBytes

Historically, computer professionals have used the SI prefixes kilo, mega, giga, etc. to mean multiples of 2^{10} instead of the usual 10^3: a kilobyte (KB) is 1024 bytes, a megabyte (MB) is (1024×1024) bytes, and so forth. This was convenient since computer professionals generally work in powers of two, and 1024 is "close enough" to 1000.

However, two things have changed in the last decade. First, people other than computer professionals have been buying memory and storage in large quantities; and second, memory and storage capacities themselves have grown rapidly. Today disk (and even solid-state) drives are sold in GB or TB, memory is routinely sold by the GB, and it is not uncommon for large enterprise storage arrays to be measured in PB.

Even in computing, the meaning of KB, MB, etc. is not always based on multiples of 2^{10}. Disk and solid-state drive manufacturers assume that a KB is 1000 bytes, a MB is (1000×1000) bytes, and so forth. Therefore, today's drive capacities are measured in GB (10^9 bytes) or TB (10^{12} bytes). Telecommunications engineers also use the SI prefixes in their decimal sense. Computer hardware engineers and software developers still generally refer to powers of two when sizing memory. As the values represented by the prefixes get larger, so does the divergence between the binary and decimal meanings of the prefixes.

To address this issue, at the end of 1998, the International Electrotechnical Commission approved as an IEC International Standard names and symbols for prefixes for binary multiples for use in the fields of data processing and data transmission. The prefixes are as follows:

Factor	Name	Symbol	Origin	Derivation
2^{10}	kibi	Ki	kilobinary: $(2^{10})^1$	kilo: $(10^3)^1$
2^{20}	mebi	Mi	megabinary: $(2^{10})^2$	mega: $(10^3)^2$
2^{30}	gibi	Gi	gigabinary: $(2^{10})^3$	giga: $(10^3)^3$
2^{40}	tebi	Ti	terabinary: $(2^{10})^4$	tera: $(10^3)^4$
2^{50}	pebi	Pi	petabinary: $(2^{10})^5$	peta: $(10^3)^5$
2^{60}	exbi	Ei	exabinary: $(2^{10})^6$	exa: $(10^3)^6$

Table 1. Binary prefixes with their decimal derivations.

By using the binary prefixes in Table 1 and reserving K, M, G, ... for decimal multiples, it is possible to clearly state what is intended when discussing storage or network needs. This leads to the following question—if an application needs to store x GiB of data, or to transmit y MiB of data per second, how much disk space or network bandwidth is actually needed when stated in decimal units? Conversely, given available bandwidth of b KB/second or disk drive space totaling d TB, how much usable capacity is available in binary units? Your team is to write a program that will answer these questions.

Input to your program will be lines containing conversion requests for values in bytes. Each input line will contain three fields separated from each other by single spaces: a positive numeric value (possibly including a decimal point), the unit for that value, and the desired unit that the value is to be expressed in. The units will be one of KB, MB, GB, TB, KiB, MiB, GiB, or TiB. All conversion requests will be less than 2^{50} bytes.

Your program is to process each conversion request and print the result on a separate line starting in the first column. The numeric result is to be rounded to, and printed with, three digits after the decimal point and printed without leading zeroes or spaces. This result is to be followed by a single space and the abbreviation for the desired unit without trailing whitespace.

Problem 5
BiBytes (continued)

Sample Input

```
4 TB MiB
300 KB KiB
1000 MiB GiB
0.5 GB MiB
2.5 GiB MB
1.544 MB MiB
```

Output for the Sample Input

```
3814697.266 MiB
292.969 KiB
0.977 GiB
476.837 MiB
2684.355 MB
1.472 MiB
```

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
Light Switch

A lighting company has announced a new product—a blinking string of lights that can be ordered with a variable number of bulbs (N) on it. This string of lights is unique because instead of blinking on and off in unison, there is a special pattern that governs how bulbs get turned on and off. Every second, one or more bulbs toggle from on to off or off to on depending on their last states and their positions from the beginning of the string—a bulb is toggled if its position is a multiple of time t. Assume that at time $t = 0$ all bulbs are off. At time $t = 1$ all bulbs would toggle on (1, 2, 3, 4, etc.) At time $t = 2$ only even numbered bulbs (2, 4, 6, 8, etc.) change state, at time $t = 3$ every third bulb (3, 6, 9, 12, etc.) changes state, and so on. This continues until time $N + 1$ at which point all bulbs reset to off (as they were at $t = 0$). The lights all toggle on again at time $N + 2$ and the process repeats.

The company's Quality Control department is having a hard time verifying that the bulbs are turning on and off at the appropriate time. Your team has been asked to write a verification program that, given the number of bulbs on the strand N, a particular time t, and a bulb number b, will determine if that bulb should be on or off.

The following limits hold for N, t, and b:

$$\begin{aligned} 3 \leq N &< 2^{54}, \\ 1 \leq t &< 2^{54}, \\ 1 \leq b &< 2^{54}, \\ b &\leq N. \end{aligned}$$

Input to your program will be a series of lines, each containing values for N, t, and b, in that order, separated from each other by one or more spaces. Input is completed at end of file. No input line will exceed 80 columns.

For each input line, your program is to print a line containing the bulb number followed by a colon, a single space, and the string "On" if the bulb is on, or "Off" if the bulb is off. No leading or trailing whitespace is to appear on an output line.

Sample Input

```
55 10 24
55 68 24
20 70 5
```

Output for the Sample Input

```
24: Off
24: On
5: Off
```

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
SPECTRUM

Swamp County Consulting has just won a contract from a mysterious Government Agency to build a data base to investigate connections between what they call "targets." Your team has been sub-contracted to create the query engine for this system. You are to report on targets, connections between targets, and the hop counts between connections as these terms are defined below.

A "target" is represented by a string of up to 32 printing characters with no embedded spaces.

A "connection" is a bi-directional relationship between two targets.

The "hop count" between a given target (referred to here as "target1") and another target is determined by the following rules:

- Targets directly connected to target1 are 0 hops away.
- Targets directly connected to the 0 hop targets, and not already counted as a 0 hop target, are 1 hop targets.
- Similarly, targets directly connected to the n hop targets, and not already counted as 0 through n hop targets are $n + 1$ hop targets.
- Targets are not treated as directly or indirectly connected to themselves.

The query engine is to have only three commands: *add*, *associated*, and *connections*. Targets and connections are never deleted because the Agency never forgets or makes mistakes.

Commands start in the first column of a line. Commands and their parameters are separated by whitespace. Input lines are at most 80 characters long. The input is terminated by end-of-file.

The commands are defined as follows:

`add` *target1*

One-parameter form of add command:
If target is not yet in the data base, add it with no connections. If target is already in the database do nothing (this is not an error).

`add` *target1 target2*

Two-parameter form of add command:
Create a bidirectional connection between the targets. There can be at most one direct connection between targets. If either target is not yet in the database, add it/them, and create the connection. If there is already a connection between the targets, do nothing (this is not an error). If target1 and target2 are the same, treat this as if the command was "add target1" (this is also not an error).

`connections` *target*

Command to print the number of hops to direct and indirect connections from the target:
Beginning with hop count 0, print the hop count, a colon, a single space, and the number of targets with that hop count with no leading spaces on a separate line. Continue by incrementing the hop count by one, ending with the hop count with the last non-zero number of targets.

- If the target has no connections print a line containing only the string "no connections".
- If the target is not in the database print a line containing only the string "target does not exist".

Problem 7
SPECTRUM (continued)

`associated` *target1 target2*

Command to print information about the existence of a connection between the two targets:
If there is a path between the targets print "yes: n" on a separate line where n is the hop count of target2 with respect to target1. Print one space after the colon with no leading zeroes and no trailing spaces. If there is no connection between the targets, print "no" on a separate line.

- If either target1 or target2 is not in the database, print a line containing only the string "target does not exist".

The input to be processed will have fewer than 800,000 lines.

Sample Input

```
add Batman Superman
add Batman Gotham-City
add Superman 10.110.139.100
add Metropolis Superman
add Gotham-City 213-555-1984
add Gotham-City Riddler
add 213-555-1984 The-Joker
add The-Joker laughing-gas
add medicare-fraud question-mark-cane
associated Batman laughing-gas
associated laughing-gas Batman
associated 213-555-1984 question-mark-cane
associated Batman The-Joker
connections Batman
connections laughing-gas
add question-mark-cane Riddler
associated Batman medicare-fraud
connections Batman
add The-Joker Batman
connections Batman
associated Batman The-Joker
add Tesla
add Goat-Island Goat-Island
connections Goat-Island
add Batman Goat-Island
add Tesla Goat-Island
connections Goat-Island
```

Problem 7
SPECTRUM (still continued)

Output for the Sample Input

```
yes: 3
yes: 3
no
yes: 2
0: 2
1: 4
2: 1
3: 1
0: 1
1: 1
2: 1
3: 2
4: 1
5: 2
yes: 3
0: 2
1: 4
2: 2
3: 2
0: 3
1: 5
2: 1
3: 1
yes: 0
no connections
0: 2
1: 3
2: 5
3: 1
4: 1
```

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
First to Solve

At the Southern California regional of the ACM International Collegiate Programming Contest, bonus prizes may be awarded to the first team to solve each of the contest problems. There is one restriction—no team can win more than one such bonus prize in any given regional contest.

It is of course possible for a team to actually be the first to solve more than one of the contest problems. Such a team receives the bonus prize for the problem with the fewest total solutions they were the first to solve—and if they were the first to solve more than one problem with the same number of total solutions, they receive the prize for the problem they solved last.

To implement this rule, at the end of the contest the problems are listed in ascending order of number of accepted solutions. If two problems have the same number of solutions, those problems are listed in descending order of their first solution time. Using the ordered list, the team that solved the first problem in the list is assigned its prize. The remaining problems in the list are examined in turn in one or more passes. During the first pass, the team that solved each problem first is assigned a prize unless that team has already received a prize earlier in the process. Should the first team to solve a given problem have already been assigned a prize, that problem remains on the list to be processed in the next pass. Problems are removed from the list as their prizes are awarded.

After reaching the end of the first pass, a second pass is made over the problems that remain in the same order (the problem processing order does not change between passes). The second team to solve each problem is now considered, and if that team has not been assigned a prize, the team is assigned the prize for that problem. If that team has already been assigned a prize, the problem remains on the list again. The passes continue until either a prize has been assigned for each problem with at least one solution or no teams remain to be considered for a given problem.

Consider the case of a contest with five problems and six teams as seen in the sample input. Table 1 shows the solutions for each problem. Problems 1, 2 and 5 each had one accepted solution by a different team, hence the appropriate teams are assigned the prizes for those problems. Problem 4 was solved by three teams, and Problem 3 was solved by four teams. The second team to solve Problem 4 was T04, and that team already received a prize. The second team to solve Problem 3 was T03, therefore team T03 is assigned the prize for Problem 3. The last solution for Problem 4 was from T03, which now already has a prize—hence there is no award for Problem 4.

Your team is to write a problem that will take a log of accepted solutions from a contest and produce a list of the problems (in problem number order) along with the team ID that is assigned the prize for that problem.

The first line of input to your program will contain two values, separated from each other by a single space: the number of problems n (in the range 2–16 inclusive), and the number of teams t (in the range 2–400 inclusive).

The remaining input will consist of lines representing accepted submissions. Each line will contain the problem number (integer between 1 and n inclusive), the team ID (a string of one to eight alphanumeric characters), and the time of the submission in integer seconds (between 1 and 18,000 inclusive). These fields will be separated from each other by single spaces. The input lines will not be in any particular order.

Problem 8
First to Solve (continued)

Your program is to print one line for each problem giving the problem number, a colon, a single space, and either:

- the team ID of the team that is to receive the prize for that problem,
- the string "Not Solved" if no accepted solution was received for a problem, or
- the string "No Award" if the problem was solved but no award is made based on the rules above.

No leading or trailing white space is to appear on an output line.

You may assume that every entry in the accepted solution log will have a unique timestamp.

Sample Input

```
5 6
3 T03 13431
5 T04 14172
2 T02 12474
4 T03 15157
4 T04 4888
3 T06 14809
1 T01 11509
3 T05 14677
4 T02 3936
3 T01 7860
```

Output for the Sample Input

```
1: T01
2: T02
3: T03
4: No Award
5: T04
```

Problem	**1**	**2**	**3**	**4**	**5**
Accepted Solutions in Order	T01	T02	~~T01~~	~~T02~~	T04
			T03	~~T04~~	
			T05	~~T03~~	
			T06		
Solution Count	*1*	*1*	*4*	*3*	*1*

Table 1. Chart showing sample input solutions and prizes awarded.

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Moving Triangles

Planet ACM has two moons that travel around it. The shadow each moon casts on the ground is shaped as a triangle. Through sophisticated calculations, astronomers on Planet ACM have forecasted that the shadows of the moons are going to overlap tonight! The overlapping of the moon shadows is a very important event on Planet ACM. People believe that the circumference of the overlapping area is related to the fortune of the coming years. Astronomers have detected the initial positions and velocities of the shadows. They are working hard to calculate the maximum circumference of the overlapping area during the motion of the two moons but need some help. Your team is to write a program to help the astronomers calculate the maximum circumference of the overlapping area.

Figure 1 shows an example of the moon shadows. Two shadows A and B—drawn in solid lines—are moving at constant velocities v_A and v_B. At some time T, they come to the place drawn by dashed lines. The overlapping area (the shaded area in Figure 1) has circumference C that is what the astronomers are looking for. The area where the moon shadows are being observed is flat enough to be treated as a Cartesian plane.

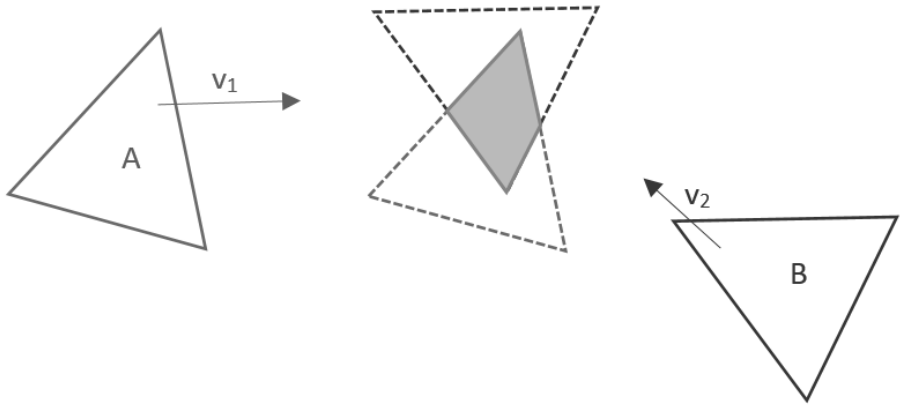

Figure 1. Two Moving Triangles

Input to your program consists of a pair of lines, describing A and B. Each triangle is described by eight real numbers on a single line separated by spaces:

$x1$ $y1$ $x2$ $y2$ $x3$ $y3$ vx vy

where $(x1, y1)$, $(x2, y2)$, and $(x3, y3)$ are the three vertices of the triangle at time $T = 0$, given in counter-clockwise order, and (vx, vy) is the constant velocity of the triangle.

Your program is to print two real numbers on a single line separated by a single space: T and C in that order. T is the time that the overlapping circumference first reaches a maximum, and C is the maximum circumference at time T. Results are to be rounded to six digits after the decimal point.

It can be assumed that the triangles will always overlap at some $t \geq 0$ and therefore $C > 0$. It is guaranteed that input triangles are not degenerated to lines, that the two triangles have different velocities, and that any edges of two triangles are not parallel.

Problem 9
Moving Triangles (continued)

Sample Input

```
0 5 3 4 2 9 6 0
7 9 9 4 11 9 -1 1
```

Output for the Sample Input

```
0.882353 8.671551
```

2013/2014 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 10
BigClass

In many programming languages, a simple way to allocate a large block of memory would be by using arrays. However, in some small programming languages where arrays are not implemented, programmers need to seek an alternative way to allocate large amounts of memory. To be specific, we seek to define a structure as large as possible within a limited length of code. Consider the following block of code:

```
struct A1 {
char a;
char b;
};
struct A2 {
A1 a;
A1 b;
A1 c;
};
struct A3 {
A1 a;
A2 b;
A1 c;
};
```

In the above 14 lines of code, we have a structure A3 that contains 10 bytes (A1 with 2 bytes, A2 with 6 bytes).

Formally, we define our task as follows:
1) Build a structure that contains the most bytes of memory within N lines of code.
2) Each structure declaration starts with a line "`struct` *ClassName* `{`" and ends with a line "`};`".
3) In each line only one variable can be declared (no comma is allowed).
4) The type of declared variable can only be char (one byte) or any structure that is already defined in the above code.

For example, the above block of code declares a structure A3 with 10 bytes. This is not optimal because the following 14 lines of code declare a structure with 25 bytes:

```
struct A1 {
char a;
char b;
char c;
char d;
char e;
};
struct A2 {
A1 a;
A1 b;
A1 c;
A1 d;
A1 e;
};
```

For a given length of code, what is the largest structure that can be declared? Your team is to write a program to answer this question.

Problem 10
BigClass (continued)

Input to your program will be a series of integers, one per line, each starting in the first column and immediately followed by end-of-line. Each integer represents the length of code, in lines, to be written as per the specifications above. These integers will be in the range 3–160 inclusive.

For each input value, your program is to print a line containing an integer value that is the largest amount of memory (in bytes) that can be allocated by a structure in the specified length of code. No leading or trailing whitespace is to appear on an output line.

Sample Input

```
5
10
14
```

Output for the Sample Input

```
3
9
25
```