Problem A. Eating Brains

Input file:	`brains.in`
Output file:	`brains.out`
Time limit:	2 seconds (3 seconds for Java)
Memory limit:	256 Megabytes

Two zombies are playing a game. The goal of the game is, well, to eat brains. Your opponent's ones, of course. It is played by moving a brain-shaped token on a board. Board has several circles with arrows between them. Each arrow has a small English letter on it. There are no two outgoing arrows from one circle having the same letter. When the game begins, players select special winning word, then place the token at starting circle and start making turns. Each turn zombie rolls a die with letters 'a'...'z' on it fairly. After that she moves the token along the arrow outgoing from the current circle that has selected letter and passes turn to her opponent. If there is no arrow with such letter, she rolls a die once again and so on. Letters of traversed arrows are written down to form a string. When the winning word appears as a substring of this string for the first time, the game stops and zombie who made the last turn wins and eats brain of her opponent.

There is also one thing you should be aware of. Ancient legend says there is a creature sleeping deep in the ocean. Whoever writing magical phrase awakens this creature. Once awakened, it will eat everybody's brains. The real danger of this phrase is that to awake evil creature it is sufficient to write some other text containing magical phrase as subsequence of letters. Obviously, two poor zombies playing the game can eventually write something that contains magical phrase, thus inducing Apocalypse on everyone including themselves.

After reading all this, you most likely wonder, how many turns it will take until somebody's brain will be eaten. More specifically, you are interested in statistical expectation of number of turns until that event. Write a program to satisfy your curiosity.

Input

The first line of the input file contains two integers N and M — the number of circles on the game board and the number of arrows respectively ($1 \le N \le 20$). Each of the next M lines has description of one arrow in the form "u v c" (without quotes). The edge goes from u-th to v-th circle and has c letter on it. Circles are numbered from 1 to N, each circle has at least one outgoing arrow. Starting circle always has number 1. Then two lines follow. First of them contain a word selected by zombies, second — a secret magical phrase. Both selected word and magical phrase are non-empty and consist only from lowercase English letters. Word has at most 10 letters, phrase — 50 letters in length.

Output

Output file must contain one number with at least two digits after decimal point — the answer for this problem. If there is non-zero probability that the game will never end, output `Infinity` instead. It is guaranteed that if the answer is finite, it will not exceed 10^6.

Example

brains.in	brains.out
1 2 1 1 a 1 1 b aa aab	4.50
5 5 1 2 c 2 3 t 3 4 h 4 5 u 5 3 l brains phngluimglwnafhcthulhurlyehwgahnaglfhtagn	Infinity

Problem B. Deep In The Ocean

Input file:	`deep.in`
Output file:	`deep.out`
Time limit:	1 second
Memory limit:	256 megabytes

After eating brain zombie starts to play a new game. In this game she has N vertices. Some pairs of vertices are connected by edges. Every vertex has the same even number of adjacent edges. Zombie needs to keep some edges so that every vertex has two adjacents edges. If zombie does it she will eat brain of creature sleeping deep in the ocean. Help her!

Input

In the first line of the input file there are two integers N and M representing the number of vertices and the number of edges ($1 \le N \le 1\,000, 0 \le M \le 50\,000$). Following M lines contain u and v ($1 \le u, v \le N$) — numbers of distinct vertices which are connected. Every pair is connected by at most one edge.

Output

If it is possible to eat the brain of the creature, print E — the number of edges kept, following E lines must contain u and v — numbers of vertices. If there is no way to keep edges, print "`Impossible`"

Example

deep.in	deep.out
3 3 1 2 1 3 2 3	3 1 2 2 3 3 1
7 14 1 2 1 3 1 4 1 7 2 3 2 5 2 6 3 4 3 6 4 5 4 7 5 6 5 7 6 7	7 1 4 2 3 3 1 4 7 5 6 6 2 7 5

Problem C. LCP Problem

Input file:	`lcp.in`
Output file:	`lcp.out`
Time limit:	2 seconds (3 seconds for Java)
Memory limit:	256 Megabytes

Consider string $s = s_1 s_2 \dots s_l$ consisting of small letters 'a'...'z'. *The longest common prefix* for two indices i and j ($1 \le i, j \le l$) is the string of the maximal length k such that it is prefix for both substrings $s_i \dots s_l$ and $s_j \dots s_l$. We will denote the length of it as $\text{lcp}(i, j)$.

Well-known problem is to find lcp for two given indices. In this problem you will have to solve the inverse problem: given string s and a set of queries k_q, for each query find a pair (i_q, j_q) such that $\text{lcp}(i_q, j_q) = k_q$.

Input

First line of the input file contains non-empty string s by itself. Its length does not exceed 200000. In the next line there is a non-negative integer number Q ($Q \le 100001$) — the number of queries. Each of the next Q lines contains one query — integer k_q ($0 \le k_q \le \text{Length}(s)$).

Output

Output file must contain exactly Q lines each containing a pair of integers i_q and j_q — the answer for the corresponding query. If there is no such pair (i_q, j_q) for which $\text{lcp}(i_q, j_q) = k_q$, then output "`-1 -1`" instead (without quotes). If there are several possible answers output one for which longest common prefix is lexicographically smallest. If there are still several solutions, output one with smallest i_q, amongst them — one with smallest j_q.

Example

lcp.in	lcp.out
abacaba 4 1 2 3 4	1 3 2 6 1 5 4 4

Problem D. Yet Another Language

Input file:	`yal.in`
Output file:	`yal.out`
Time limit:	10 seconds
Memory limit:	256 Megabytes

Vasya Pupking has a new idea. He has decided to create a new programming language called YAL (or *Yet Another Language*). This language will be interpreted, dynamically typed, like many others. The killer-feature of this language will be its extensive out-of-the-box support for regular expressions. Not only you will be allowed to use regexps in data processing, but in variable and function names, class names and so on. This should help programmers and make programming easier and funnier.

Now Vasya needs to make a demo-version of interpreter for YAL to demonstrate it to possible investors. As he is very busy devising commercial model for YAL, he asks you to write interpreter.

Because this is only demo-version, interpreter will have certain limitations. There will be only two types for variables and constants: integer and string. There will be no support for functions and classes. Every statement will be on individual line. There will be only two types of statements: assignment and printing. BNF of YAL is as follows:

assignment	::⇒	varname '`=`' expression
printing	::⇒	print expression
expression	::⇒	summand ['`+`' expression]
summand	::⇒	multiplicand ['`*`' summand]
multiplicand	::⇒	constant | varregexp | '`(`' expression '`)`'
constant	::⇒	integer | string

Here `print` is a keyword, `integer` is an integer constant consisting of digits '`0`'...'`9`', `string` is a string constant enclosed by double quote marks '`"`', `varregexp` is a regular expression representing some variable, `varname` — variable name. String constant will never contain any double quote marks inside it. It is assumed that operation '`*`' has higher priority than '`+`'. Operations of the same priority are evaluated from left to right.

For the sake of simplicity, regular expressions will also have only limited support.

Regular expressions can be expressed in terms of formal language theory. Regular expressions consist of constants and operators that denote sets of strings and operations over these sets, respectively. Given a finite alphabet Σ the following constants are defined:

- (empty set) $\varnothing$ denoting the set $\varnothing$
- (empty string) ε denoting a string with no characters {“”}.
- (literal character) a in Σ denoting a set containing only a single-character string of the language - {“a”}.

The following operations are defined:

- (concatenation) RS denoting the set $\{\alpha\beta \mid \alpha \in R \text{ and } \beta \in S\}$. For example {“ab”, “c”}{“d”, “ef”} = {“abd”, “abef”, “cd”, “cef”}.
- (alternation) $R|S$ denoting the set union of R and S. For example {“ab”, “c”}|{“d”, “ef”} = {“ab”, “c”, “d”, “ef”}
- (Kleene star) R* denoting the smallest superset of R that contains ε and is closed under string concatenation. This is the set of all strings that can be made by concatenating zero or more strings in R. For example, {“ab”, “c”}* = {ε, “ab”, “c”, “abab”, “abc”, “cab”, “cc”, “ababab”, “abcab”, ...}.

> The above constants and operators form a Kleene algebra.
>
> To avoid brackets it is assumed that the Kleene star has the highest priority, then concatenation and then set union. If there is no ambiguity then brackets may be omitted. For example, $(ab)c$ can be written as abc, and $a|(b(c*))$ can be written as $a|bc*$.
>
> *From Wikipedia, the free encyclopedia*

BNF for regular expressions in YAL is following:

regexp	::⇒	regcat ['`|`' regexp]
regcat	::⇒	regstar [regcat]
regstar	::⇒	regclass ['`*`']
regclass	::⇒	char | '`[`' ['`^`'] regc '`]`' | '`(`' regexp '`)`'
regc	::⇒	char [regc]
char	::⇒	'`A`'–'`Z`', '`a`'–'`z`', '`0`'–'`9`', '`.`', '`_`', ' '

The last character for `char` is space (ASCII 32). Square brackets define character classes, that is, any character listed in square brackets matches corresponding character class. Character '`^`' means negation, so, any character NOT listed in square brackets will match the class. For example, "`[ad9]`" represents any character of '`a`', '`d`', '`9`' and "`[^0123456789]`" represents any non-digital character. Dot character '`.`' has special meaning: it matches any character while being specified as character of alphabet or inside square brackets as element of character class. So, the following notations are equivalent: "`[.0123456789]`"= = "`[.]`" = "`.`". All of them represent a string of length 1 the only character of which is arbitrary.

As the same symbols can appear both in regular expressions and in statement expressions, Vasya has decided that variable identifiers (such as variable names and regular expressions representing variable), integer and string constants must be divided from other parts of statement expression by at least one space. There is also at least one space on both sides of equation mark '`=`' in assignment statement and between "`print`" keyword and expression in printing statement. Additional spaces can appear at arbitrary positions in statement expressions but they will never appear inside integer constant, although string constants can have spaces which are parts of them. Neither variable names nor variable regular expression will contain any spaces. Any variable name is non-empty string that can contain upper and lower case English letters, digits and underscores ('`_`'). Variable name will not begin with a digit, neither variable regexp will do.

Interpreter of YAL works as follows. It reads program line by line and execute it. When it meets printing statement, it simply evaluates expression and prints its value. In the case of assignment statement, it evaluates expression first, then it looks for a variable being assigned. If there is already variable with that name, it assigns this variable with a new value (if the value of the expression has type different from the type of the variable, then type of the variable changes accordingly). If there is no such variable, it creates a new one and assigns it with the value.

Statement expression being evaluated by interpreter can contain regular expressions for variables. In this case interpreter have to find a variable which name matches corresponding regular expression. If there are several such variables, it chooses one which name comes first lexicographically. Note that all regular expressions are case-sensitive, so is the search for variable matching regexp. After variable is determined, interpreter uses its value in the corresponding expression. If there is no such variable at all, it prints the message "`No variable found matching regexp`", where "`regexp`" is the regular expression for which variable cannot be determined, then it proceeds to executing the next statement. If there are several regexps within the same expression not having matching variables, the one that comes first (from left to right) must be reported.

As the variable type is determined during the execution, sometimes it is possible that operands of '+' or '*' operators will have different types. This is not an exceptional situation, and table below contains description of '+' or '*' operators for different left and right argument types.

left	right	operator	description
integer	integer	+	The sum of `left` and `right` operands.
integer	integer	*	The product of `left` and `right`.
integer	string	+	The integer operand is being converted to its string representation (without leading zeros) and then concatenated with the string operand.
string	integer	+	
integer	string	*	If the integer operand is positive, the result is the string operand repeated the value of integer operand times. Otherwise, the result is an empty string.
string	integer	*	
string	string	+	Concatenation of `left` and `right`.
string	string	*	`right` is treated as regexp. The result is substring of `left` of maximal length matching `right`. If there are several such substrings, one that comes first lexicographically is chosen. If there are no such substrings, the result is an empty string.

In the last case, if right operand is not a correct regular expression, then interpreter prints error message "`Invalid regexp`" and then skips current statement just like when it cannot find variable.

Note that addition and multiplication for strings are not commutative. Moreover, operations are no longer associative, because value of expression can depend on the operations order.

So, as you are given requirements specification, write a program that will read program in YAL from the input and interpret it.

Input

Input has one or more lines (not more than 100). Each line represents some program statement. Each statement expression does not have more than 10 operators. Every variable name and variable regexp is non-empty string meeting all requirements specified above. Length of variable names, variable regexps and string constants does not exceed 20 characters. You may also assume that all intermediate results for string evaluations are also at most 20 characters long. Integer constants, as well as intermediate results for evaluating integers always fit in 32-bits signed integer. You may assume that string constants contain only the following characters: 'A'...'Z', 'a'...'z', '0'...'9', '[', ']', '(', ')', '+', '*', '.', '^', '_' (ASCII 95), '|' (ASCII 124) and ' ' (ASCII 32).

Output

Output must contain values of print expressions and error messages, on separate line each, in order they are generated.

Example

yal.in	yal.out

`abba = "ab" * 3`
`abacaba = [ab]* * "(ba)*"`
`print ([ab]*|[^c])* +( " " + .*c.* )`
`print bu[goa]*`
`print 2 * 2`
`print abba * "[^a"`
	`ababab baba`
	`No variable found matching bu[goa]*`
	`4`
	`Invalid regexp`

Problem E. Sequence

Input file:	`seq.in`
Output file:	`seq.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given an infinite sequence $x_0, x_1, x_2, \ldots$ where $x_i = a + p * i$ (p is the prime number). You must find out whether there is an infinite subsequence containing exact squares of integer numbers only.

Input

The first line of the input file contains t — the number of tests ($1 \le t \le 1\,000$). The following t lines contain two integer numbers each — a and p ($0 < a < p < 2^{32}$), p is the prime number.

Output

For each test case output "`Yes`" or "`No`".

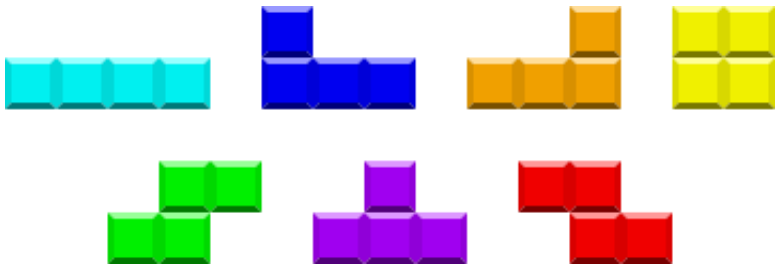
Example

seq.in	seq.out
2 8 17 13 239	Yes No

Problem F. Tetris

Input file:	`tetris.in`
Output file:	`tetris.out`
Time limit:	1 seconds
Memory limit:	256 Megabytes

Consider Tetris game played on a field of width 4 and height N. In this problem you are to count the number of different ways to fill the field with tetrominoes gaplessly. It is considered that you can just place some tetromino at free location in the field even if this free area is enclosed by other tetrominoes. Tetrominoes of the same type and orientation are indistinguishable. All possible types of tetrominoes are depicted in the picture below.



So, write a program to count the number of fillings. As this number could be very large, it is sufficient to count it modulo some prime number P.

Input

Input file will contain two integer numbers N and P on its first and the only line. You may assume that $1 \le N \le 10^9$ and $2 \le P \le 10^9 + 9$. P is guaranteed to be a prime number.

Output

Output file must contain one number — the answer for this problem.

Example

tetris.in	tetris.out
1 1000000007	1
2 1000000009	4

Problem G. Fraction

Input file:	`fraction.in`
Output file:	`fraction.out`
Time limit:	1 second
Memory limit:	256 megabytes

A positive rational number is normally represented as $\frac{p}{q}$ where p and q are positive integer numbers. It can also be represented as finite continued fraction

$$q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots \cfrac{}{q_{n-1} + \cfrac{1}{q_n}}}}}$$

where n is minimized, $q_0 \geq 0$ and $q_i > 0$ for each $i > 0$.

You are given number $\frac{p}{q}$ (p, $q > 0$) in the second representation. Find $\frac{q}{p}$ in the same representation.

Input

The first line of the input file contains n ($0 \leq n \leq 10^5$). The second line contains $n + 1$ integer numbers — q_i ($0 \leq q_0 \leq 10^9, 1 \leq q_i \leq 10^9, 1 \leq i \leq n$).

Output

Output fraction $\frac{q}{p}$ in the same format as in the input.

Example

fraction.in	fraction.out
1 1 2	2 0 1 2

Problem H. GCD

Input file:	gcd.in
Output file:	gcd.out
Time limit:	1 seconds
Memory limit:	256 megabytes

Greatest common divisor (GCD) of several non-zero numbers is the largest positive integer that divides all these numbers without a remainder. For each positive number N let's define P_N as: $P_N = \{x \mid x - \text{permutation of digits of } N\}$. For example, $P_{102} = \{012, 021, 102, 120, 201, 210\}$. Your task is to write a program that calculates GCD of P_N.

Input

The first line of the input file contains 10 integer numbers: $a_0, \ldots, a_9$ ($0 \le a_i \le 10^{12}$) describing N. a_i is the number of digits i in N. It is guaranteed that at least one of a_i ($1 \le i \le 9$) is non-zero.

Output

Output one integer number – GCD of P_N. If the answer has more than 10^6 digits, output `TOO LONG` instead.

Example

gcd.in	gcd.out
1 1 1 0 0 0 0 0 0 0	3
0 0 1000001 0 0 0 0 0 0 0	TOO LONG

Problem I. Frozen Orb

Input file:	`frozenorb.in`
Output file:	`frozenorb.out`
Time limit:	2 seconds (3 seconds for Java)
Memory limit:	256 Megabytes

Vasya is a long-time fan of the "Clissard Entartainment" company. He really enjoys playing its Action-RPG games. Especially the series called "Angel". As the development of the new game "Angel VIII" has been unveiled recently, Vasya wants to recover his Angel skills to get prepared for the release. This time Vasya is playing "Angel VII" as a cold sorceress. And he has entered "The Secret Cow Level" once again.

The cow level is an infinite plane with Vasya and several Eden Bovines (cows) on it. Vasya is represented by a point with coordinates $(X_{\text{vasya}}; Y_{\text{vasya}})$. Vasya utilizes the Frozen Orb skill. The Frozen Orb can be shot in an arbitrary direction.

Let φ be the polar angle of the shot direction. The Frozen Orb flies along the straight line with constant velocity V from the point where Vasya stands:

$$X_{\text{orb}}(t) = X_{\text{vasya}} + Vt\cos\varphi$$
$$Y_{\text{orb}}(t) = Y_{\text{vasya}} + Vt\sin\varphi$$

The moment of shot is $t = 0$. At the moment $t = T$ the orb disappears. While flying the orb releases Ice Bolt packs after equal periods of time. Each Ice Bolt pack consists of M bolts. During the flight the orb releases K bolt packs. Let's label each bolt with two numbers: $k \in [1..K]$ - the bolt pack number and $m \in [0..M-1]$ - the bolt number in the k-th pack. Notice that $K \cdot M$ Ice Bolts are shot overall. The moment $t = T_k$ of k-th bolt pack release is determined as follows:

$$T_k = T\frac{k}{K}$$

The Frozen Orb rotates with constant angular velocity ω:

$$\theta_{k,0} = \varphi + \theta + \omega T_k$$
$$\theta_{k,m} = \theta_{k,0} + 2\pi\frac{m}{M}$$

where $\theta_{k,m}$ is the polar angle of the shot direction of m-th bolt from k-th bolt pack. Each Ice Bolt flies eternally with constant velocity U along straight line:

$$X_{k,m}(t + T_k) = X_{\text{orb}}(T_k) + Ut\cos\theta_{k,m}$$
$$Y_{k,m}(t + T_k) = Y_{\text{orb}}(T_k) + Ut\sin\theta_{k,m}$$

Vasya is surrounded by N Eden Bovines. i-th bovine is a circle with center $(X_i; Y_i)$ and radius R_i. It has H_i Hit Points. We will say that the particular Ice Bolt hits the particular cow if and only if at some moment the bolt is located inside or on the boundary of the cow. When the cow is hit by the bolt it loses 1 Hit Point. When the number of Hit Points of the bovine becomes nonpositive, it dies. The Ice Bolt **doesn't** disappear after a hit.

In this problem Vasya and cows cannot move. Vasya wants to cast one Frozen Orb and kill as much bovines as possible. Help him choose the direction to maximize the number of dead cows. Due to some Vasya-specific mouse moving skills you are to choose the direction with minimal polar angle φ among ones with the maximal number of kills. The polar angle must satisfy the condition $0 \le \varphi < 2\pi$. In this problem the point with polar angle g has coordinates $(R\cos g; R\sin g)$.

Input

The first line contains three integers:
K - the number of bolt packs ($K \geq 1$)
M - the number of bolts per pack ($M \geq 2$)
N - the number of Eden Bovines ($N \geq 1$)
The second line contains three real numbers:
L - the overall distance the orb passes. ($L = VT > 0$)
δ - the overall rotation angle of the orb. ($\delta = \omega T$)
θ - the orientation of the orb at moment $t = 0$.
The third line contains two real numbers X_{vasya} and Y_{vasya} - the coordinates of Vasya.
The next N lines contain descriptions of cows.
Real X_i and Y_i, real R_i and integer H_i are written on the $i + 3$-th line.
X_i, Y_i - the coordinates of the i-th cow.
R_i - the radius of the i-th cow. ($R_i > 0$)
H_i - the number of Hit Points of the i-th cow. ($1 \leq H_i \leq 100000$)

It is guaranteed that no two cows have common points including the boundaries, Vasya doesn't lie inside or on the boundary of any cow.
Number of bolts in a pack M is guaranteed to be even.
Overall number of bolts $K \cdot M$ does not exceed 1000.
The magic product $N \cdot K \cdot M$ does not exceed 100000.

Output

Output file must contain the number of dead bovines (integer) and the polar angle φ of direction of orb shot (real). Write the angle φ as precisely as possible.

Example

frozenorb.in	frozenorb.out
2 4 5 5.0 1.570796326794897 0.0 1.0 2.0 6.0 2.0 1.0 1 2.0 -2.1 1.0 3 -1.0 0.0 2 2 1 6.1 1 1 1 -5 1 1	4 4.7123889803846897
2 4 5 5.0 -1.570796326794897 -0.35 1.0 2.0 6.0 2.0 1.0 10 2.0 -2.1 1.0 1 -1.0 0.0 2 5 1 6.1 1 8 1 -5 1 9	2 3.2502922744309739

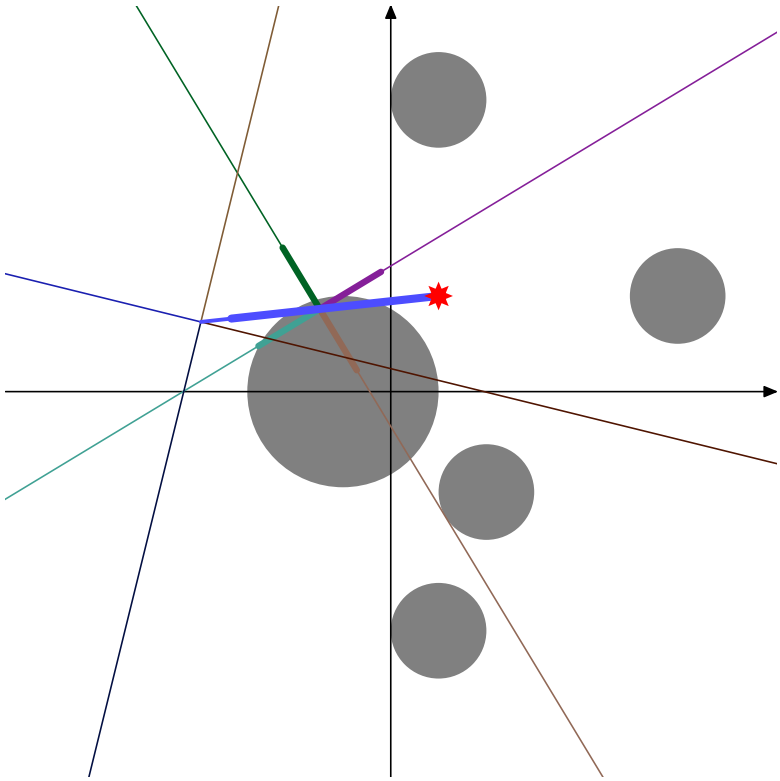


Figure 1: Explanatory picture for test 2.

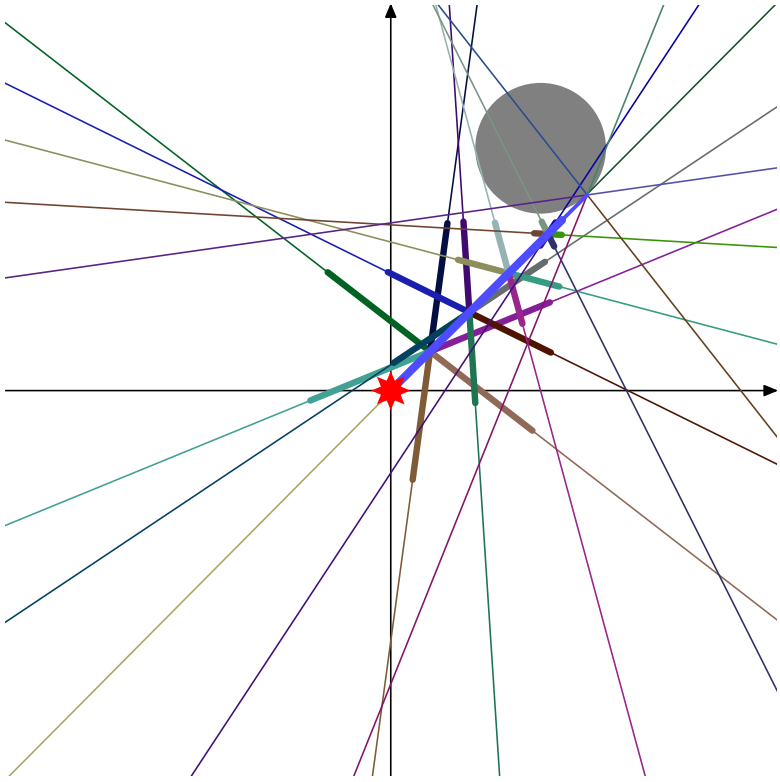


Figure 2: Yet another frozen orb example: $K = 5$, $M = 6$, $L = 6.0$, $\delta = 1.0$, $\theta = 1.5$

Problem J. Cycles

Input file:	`cycles.in`
Output file:	`cycles.out`
Time limit:	3 seconds (4 seconds for Java)
Memory limit:	256 Megabytes

Vasya Pupkin is simulating some conservative field. He knows that the integral of such field about any closed path must be equal to zero. Checking this property could be quite complicated though, so he has decided to check it only for paths composed of mesh edges. Vasya has already calculated the projections of the field on every edge and now asks for your assistance.

More formally, you are given a weighted connected directed graph with antisymmetric weight matrix, i.e. $\forall (u, v) \in E : (v, u) \in E \,\&\, w(u, v) = -w(v, u)$. You are to determine whether every cycle of this graph has zero weight. Weight of the cycle is the sum of weights of all its edges. You may safely assume that the graph has neither loops nor multiedges, but you should not assume that the graph meets any other criteria than specified above.

Input

In the first line of the input file there are two integers N and M representing the number of vertices of the graph and the number of edges specified in the input respectively ($1 \le N \le 100000$, $0 \le M \le 200000$). The following M lines contain the description of the edges in the form "u v w" (without quotes). Here u is the starting vertex of the edge, v is the ending vertex ($u \ne v$) and w is its weight. Vertices are numbered from 1 to N and edges' weights do not exceed 10^9 by absolute value. It is guaranteed that for every pair of edges (u, v) and (v, u) exactly one edge will be specified in the input.

Output

The only line of the output file must contain the answer for the problem: either `YES` or `NO`.

Example

cycles.in	cycles.out
3 3 1 2 10 2 3 -7 3 1 -3	YES
3 3 1 2 10 2 3 -7 1 3 -3	NO

Problem K. Killer

Input file:	killer.in
Output file:	killer.out
Time limit:	12 seconds
Memory limit:	256 Megabytes

$2N$ persons are playing Killer. Rules of this game are following.

Everybody gets a card with a name of another person. There are no two cards referring to the same person. Nobody knows cards of other people. Then if person A has a card with a name of person B and person A stays alone with person B, then she kills person B and takes all cards B had. The goal of this game is to survive and kill everybody but the player having card with your name. Cards are always distributed in such a way that every person can achieve this goal.

This time after getting cards, players have decided to split into pairs and go to a cliff one pair after another. During this pairs of players stay alone, so some people may be killed.

In this problem you are to calculate the probability of exactly K people to be killed. You may assume that players split into pairs randomly in the uniform manner.

Input

The only line of the input file contains two integer numbers N and K ($0 \le K \le N \le 10000$, $2 \le N$).

Output

The output file must contain the required probability in the form of irreducible fraction. Look in the sample for details.

Example

killer.in	killer.out
2 0	1/3
2 1	0/1
2 2	2/3