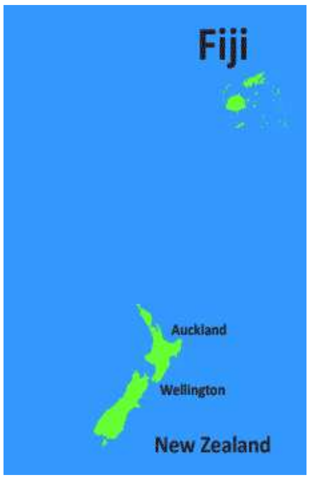


ACM ICPC South Pacific Region Divisional Round

August 22, 2015

Eastern Division Contest Problems

A: Football
B: Talking Money
C: Be Rational
D: Resistance Is (Not) Futile!
E: Knight's Shortest Path
F: Almost an Anagram
G: Who Do You Think You Are?
H: Banking
I: Shelob's Lair
J: Painting Floors
K: Selling Numbers

This page has intentionally been left almost blank

acm International Collegiate Programming Contest 2015 IBM event sponsor
SOUTH PACIFIC REGION

A: Football

Time Limit: 4 second(s)

I turned the TV on the other day and a football game (Rugby League) was being broadcast. The score was 26 to 17 and I wondered how the game had progressed and which ways the teams had scored and the order of those scoring plays to get to their current scores. After some quick back of the envelope calculations I worked out that there were 4 507 705 365 837 803 005 ways that the game could have progressed to get to the current score.

Each code of football has different ways of scoring. In soccer, the teams can only score one point at a time. In some codes different ways of scoring are worth the same number of points e.g. in Rugby Union a drop goal and a penalty goal are both worth 3 points but are different modes of scoring. In many games a team can score points for a particular play and can then attempt to score bonus points of differing values. For example, in American football (gridiron), a team can score a touchdown worth 6 points and can then attempt either a 1 point (kick) conversion or 2 point (pass or run) conversion. In this way a team could score a touchdown followed by an unsuccessful conversion for 6 points, or a touchdown followed by a successful conversion for 7 or 8 points depending on the option taken.

In football (soccer) there is only one way of scoring i.e. with a 1 point goal. A 1-1 draw can be achieved in two ways. The score could have progressed from 0-0 to 0-1 to 1-1 or from 0-0 to 1-0 to 1-1. This different ordering of team scoring also needs to be taken into consideration.

From the different ways points can be scored for a particular game, write a program to work out the number of ways, including different orders of plays, the teams could have reached their current scores. As the number of ways is likely to be large report the results modulo 1 000 000 009.

Input

The input contains a single test case.

The first line contains two integers a and b specifying the scores of the two teams, $0 \leq a,\ b \leq 200$. All scores will be attainable using the permitted means of scoring.

The second line contains a series of groups of integers, which may comprise base score values and bonus score values, signifying the different scoring options for the game being played. Each of the groups of integers are separated by a single space. Base score values $(0 < s_i \leq 20)$ will be listed in non-descending order. There will be at least one base score value and no more than 20 base score values in a test case.

When bonus points can be scored for a particular play the bonus score values are separated from the base score value by a colon e.g. for American football 6:1:2. Bonus score values for a base score value, $(0 < b_j < s_i)$, will also be listed in ascending order.

Output

Output for each test case consists of a single line as specified in the example. The word "way" should be pluralised when there are multiple ways that the scoreline could be achieved.

Sample Input and Output

Sample Input 1	Output for Sample Input
`4 6`	`4 vs 6 can be achieved 3838 ways`
`1 2 4:2`	

Sample Input 2	Output for Sample Input
`26 17`	`26 vs 17 can be achieved 268455080 ways`
`1 2 3 6:1:2`	

Sample Input 3	Output for Sample Input
`1 0`	`1 vs 0 can be achieved 1 way`
`1`	

B: Talking Money

Time Limit: 1 second(s)

The company you work for, Automatic Conversation Machina, a text to speech service provider, has just won a contract for a telephone banking system. Unfortunately their text to speech software does not yet work with numerical values like the balance in a bank account. They need you to take a currency value and convert that to words so the software will be fully functional for the telephone banking system.

The conversion must work for values between negative `$999 999 999 999.99` and positive `$999 999 999 999.99`. None of the bank's customers are trillionaires just yet.

The value must be fully converted to words for all parts of the currency amount including the cents amount i.e. `zero dollars` and `zero cents` are to be included if the dollar or cents value are zero, respectively. The following rules must be observed:

- If there is no billions, millions, thousands or units group in the value these groups must not be converted.
- All non-multiples of ten between `21` and `99` inclusive must separate the tens word from the units word with a single hyphen e.g. '`twenty-one`' and '`ninety-nine`'.
- The word '`and`' must appear after the word '`hundred`' in all cases except when the value being converted is a round hundred *e.g.* compare '`one hundred thousand`' with '`one hundred and twenty-three thousand`'.
- The word '`and`' must appear between the least of the billions, millions or thousands groups and the units group if the units group is less than one hundred except when the units group equals `0` *e.g.* compare '`two thousand and forty-six`' with '`two thousand five hundred and fifty-seven`'.
- The word '`dollars`' must appear after the dollar amount except if the dollar amount is `1` in which case the word '`dollar`' must appear.
- The word '`and`' must appear between the dollars amount and the cents amount.
- The word '`cents`' must appear after the cents amount except if the cents amount is `1` in which case the word '`cent`' must appear.
- If the value is negative, the words '`in debit`' must be added to the end of the amount.
- If the value is positive, the words '`in credit`' must be added to the end of the amount.
- If the value is zero, the words '`in debit`' or '`in credit`' must not be included.

The correct spelling for all values likely to be needed are: `zero`, `one`, `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, `nine`, `ten`, `eleven`, `twelve`, `thirteen`, `fourteen`, `fifteen`, `sixteen`, `seventeen`, `eighteen`, `nineteen`, `twenty`, `thirty`, `forty`, `fifty`, `sixty`, `seventy`, `eighty`, `ninety`, `hundred`, `thousand`, `million`, `billion`.

Input

The input contains a single test case.

The input will consist of one currency value v ($-\$999\,999\,999\,999.99 \leq v \leq \$999\,999\,999\,999.99$).

Output

The output for the test case must be on a single line with a single space between each word. **Note:** The sample output is displayed over multiple lines so that it fits on the page.

Sample Input and Output

Sample Input 1	Output for Sample Input

`-$123456789012.34`	`one hundred and twenty-three billion four hundred and`
	`fifty-six million seven hundred and eighty-nine thousand`
	`and twelve dollars and thirty-four cents in debit`

Sample Input 2	Output for Sample Input

`$14019.50`	`fourteen thousand and nineteen dollars`
	`and fifty cents in credit`

C: Be Rational

Time Limit: 1 second(s)

Captain Jean-Luc Picard and the crew of the U.S.S. Enterprise NCC-1701-D have been dispatched yet again to negotiate a peace treaty, this time between two warring cultures, the Decimators and the Fractionalists. They have fought for many years over the correct way to represent rational numbers. The Decimators represent each rational number as a possibly repeating decimal number, such `0.444...` (which they write as `0.(4)` with parentheses to denote the repeated part), whereas the Fractionalists represent each rational number as a fraction, such as `4/9`. After a devastating war in which hundreds of millions died, the Fractionalists have won. The crew of the U.S.S. Enterprise has the task of converting all numbers in the treaty to fractional form.

Input

The input contains a single test case.

The input consists of a positive rational number, represented as a possibly repeating decimal number. The whole number part comes first, and is always present. The whole number part may then be followed by both a period and a decimal part. The decimal part may end with a repeating part, which is contained in parentheses. For example, `0.(4)` represents the repeating decimal number 0.444.... There is no whitespace within a line. Each test case is no more than 10 characters long.

Output

Output a single line containing a fraction representing the input rational number. The fraction must be in reduced form i.e. the numerator and denominator contain no common factor.

Sample Input and Output

Sample Input 1	Output for Sample Input
2015	2015/1

Sample Input 2	Output for Sample Input
0.(4)	4/9

Sample Input 3	Output for Sample Input
3.(142857)	22/7

Sample Input 4	Output for Sample Input
9.(9)	10/1

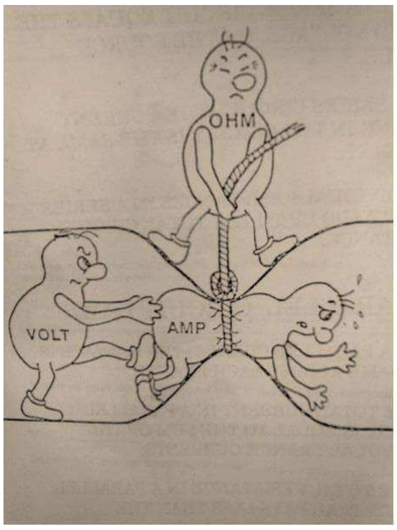
This page has intentionally been left almost blank

acm International Collegiate Programming Contest 2015 IBM event sponsor
SOUTH PACIFIC REGION

D: Resistance Is (Not) Futile!

Time Limit: 2 second(s)

You have been hired by Acme Circuit Manufacturers to help reduce the number of resistors used in their mass-produced electrical circuits, which will reduce manufacturing costs.



The humble resistor is a small, but crucial component in every electrical circuit. It plays a major role in regulating the flow of electrons (*current*) throughout a circuit by converting electrical energy into kinetic energy (*heat*), and dissipating that heat, in a controlled and quantified manner. We refer to this energy conversion as *resistance*. The unit of measurement to quantify resistance is *ohms*. The higher the ohm value, the higher the resistance.

When deciding on the number and type of resistors to be used, we need to first consider how much current we want within a path of a circuit, and also the potential energy needed to transfer electrons from one point along that path to another (*voltage*). We relate current, voltage and resistance, using a very simple formula known as *Ohm's Law*:

$V = IR$ (where V is voltage, I is current and R is resistance).

For example, let's say that on a given path in our circuit, we have 44558 volts applied and need 10 amperes of current through that path. By applying Ohm's Law (and rearranging the equation to make R the subject), we determine that we need to place a resistor along that path whose resistance is 4455.8 ohms. Problem solved, right?

Wrong.

Unfortunately, ACM only assembles circuits, it does not manufacture the components. This includes resistors. In fact, most electronics companies rely on pre-manufactured resistors. Because of this, there has been a need for standardisation of resistor values. ACM makes use of the E-12 standard range of resistors, so called because there are 12 standard base resistor values that all resistors in that range make use of, namely:

10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82.

This is referred to as the first *decade* of E-12 resistor values (measured in ohms). The second decade is:

100, 120, 150, 180, 220, 270, 330, 390, 470, 560, 680, 820.

The third and subsequent decades can easily be derived by multiplying each base value by the appropriate power of 10.

The resistances of E-12 resistors are usually only approximately equal to their nominal value but ACM have found a supplier that guarantees exact resistances. ACM wishes to use combinations of these exact resistors to achieve close approximations to actual desired resistances while using the fewest number of resistors. Resistors are always to be connected in series so that the resistance value of a set of resistors is the sum of their resistances. To measure the closeness of an approximation, ACM define the error as the distance of the approximate value (the sum of the resistances) from the target value expressed as a percentage of the approximate value. They wish to ensure that that error is at most 1%.

For example, if we wish to approximate 4455.8 ohms, we could choose the following set of resistors:

3900, 470, 82

as they sum up to 4452 ohms. The error is only $|(4455.8 - 4452)| * 100/4452 = 0.085\%$ which is well within the desired accuracy of 1%. However, a better choice would be:

3900, 560.

While the total resistance of 4460 ohms is not as accurate as that achieved with the previous choice, the error is still well under 1% and, importantly, this combination uses one less resistor (remember, manufacturing costs add up on a large scale).

Your task is to write a program that, given a voltage and current, chooses the best set of resistors to provide the required amount of resistance to within the 1% error as defined above.

Input

The input contains a single test case.

The input has two integer values V ($1 \leq V \leq 10^9$) and I ($1 \leq I \leq 10^7$). V is the voltage and I is the current.

Output

Output a series of E-12 resistor integer values, from largest value to smallest value, separated by a space, which consists of the lowest number of resistors that approximates the target resistance with an error of at most 1%.

You can use the same resistor value more than once. If you find two or more resistor sets with the same number of resistors that are within the error range, output the set whose sum is closest to the target value. If there are two sets that are the same distance from the target value, output the set whose sum is smaller. If there are still ties, output the set of resistors which is lexicographically least (when the resistors are ordered from largest to smallest).

If there are no possible sets of resistors that allow this resistance, output `Impossible` instead.

Sample Input and Output

Sample Input 1	Output for Sample Input
`50000 5`	`10000`

Sample Input 2	Output for Sample Input
`44558 10`	`3900 560`

Sample Input 3	Output for Sample Input
`1 1`	`Impossible`

acm International Collegiate Programming Contest 2015 IBM event sponsor
SOUTH PACIFIC REGION

E: Knight's Shortest Path

Time Limit: 2 second(s)

Master Li likes to play various types of chess and was thinking about how efficient the knight piece moves between positions on a chess board. It is known that the knight travels (in one move) two cells in one direction then one cell in the other axis as shown in the international 8×8 chessboard shown in Figure 1. There are also various flavours of chess, including the Chinese version (xiangqi) shown in Figure 2, where the knight is placed on the intersections of lines (equivalent to a 10×9 cell-based board).

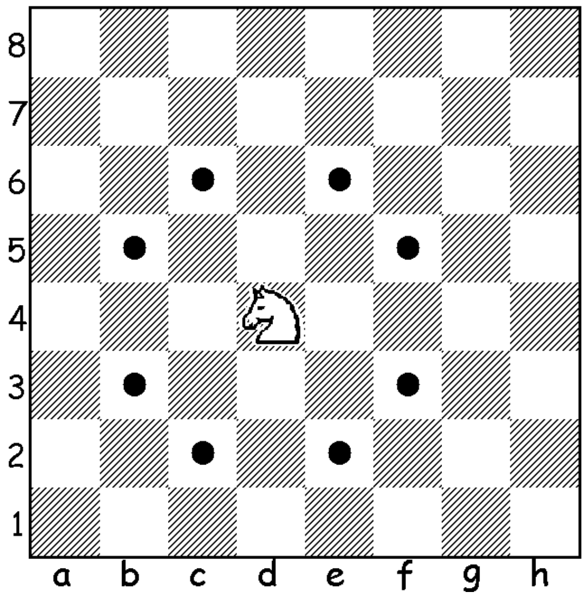


Figure 1: International Board

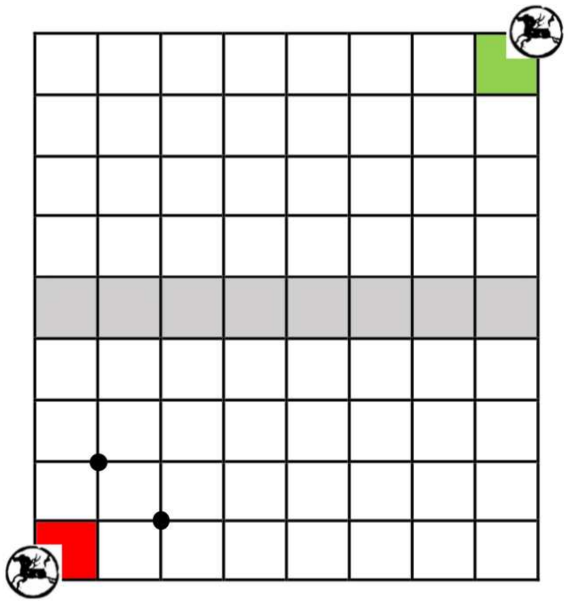


Figure 2: Xiangqi Board

With various sizes of boards, Master Li wants to know the minimum number of knight moves to travel from opposite corners of a board (e.g. go from cell 1a to cell 8h). In addition, he wants to know the total number of different minimum paths. Can you help him calculate this information?

Input

The input contains a single test case.

The input will contain two integers, r and c ($1 \leq r, c \leq 400$), denoting the number of cells in vertical direction (rows) and horizontal direction (columns) for the chessboard, respectively.

Output

If there is no path for the knight to take, output `None`. Otherwise, output a single line containing two integers showing the smallest number of knight moves and a count of the number of distinct paths of that shortest distance between two opposite corners. Since the number of paths can be quite large, output this number modulo 1 000 000 009.

Sample Input and Output

Sample Input 1	Output for Sample Input
`4 4`	`2 2`

Sample Input 2	Output for Sample Input
`4 5`	`3 1`

Sample Input 3	Output for Sample Input
`2 4`	`None`

F: Almost an Anagram

Time Limit: 1 second(s)

Andy loves anagrams. For the uninitiated, an anagram is a word formed by rearranging the letters of another word, for example `rasp` can be rearranged to form `spar`. Andy is interested to know if two words are almost anagrams. A word is almost an anagram of another word if:

- one word is shorter than the other by one letter but otherwise contains the same letters in any order; or
- the two words are the same length and their character multisets differ by one character only e.g. "`aaa`" and "`aab`"

Your job is to help Andy to determine if two words are identical, anagrams, almost anagrams or nothing like each other.

Input

The input contains a single test case.

The input will be a single line of text containing a pair of words separated by a single space. The words will be in lower case and will contain alphabetic characters only. Words will contain between 1 and 1000 letters inclusive.

Output

Your program should produce one line of output as follows:

- If the words are identical, output: `word`$_a$ `is identical to word`$_b$
- If the words are anagrams, output: `word`$_a$ `is an anagram of word`$_b$
- If the words are almost anagrams, output: `word`$_a$ `is almost an anagram of word`$_b$
- Otherwise, output: `word`$_a$ `is nothing like word`$_b$

In all cases the first word in the output sentence must be the shorter word or if the words are the same length the first word must be the lexicographically least.

Sample Input and Output

Sample Input 1	Output for Sample Input
`rasp spar`	`rasp is an anagram of spar`

Sample Input 2	Output for Sample Input
`table able`	`able is almost an anagram of table`

Sample Input 3	Output for Sample Input
`sable table`	`sable is almost an anagram of table`

This page has intentionally been left almost blank

acm International Collegiate Programming Contest 2015 IBM event sponsor

SOUTH PACIFIC REGION

G: Who Do You Think You Are?

Time Limit: 2 second(s)

Aunt Clara-May has been taking an interest in the genealogy of the family. She is able to construct a family tree but is getting confused with the relationships between different members of the family.

She wants to identify the following relationships: father, mother, uncle, aunt, son, daughter, nephew, niece, cousin, husband and wife. She also wants to be able to recognise whether the members in the family are related by blood or by marriage (i.e. in-laws), as well as different generations in the family such as grandparents, grandchildren, great grandparents, great grandchildren, great great grandparents, great great grandchildren etc. and different degrees of cousins including levels of removedness (e.g. second cousins-in-law twice removed).

Aunty C-M, as you call her, has some definitions of these relationships but needs your help to write a program to construct the family tree and name the relationships.

You have told Aunty C-M that you will help under the following conditions:

- no second marriages which require step relationships e.g. step-brother and step-father will be recorded
- all children in the family tree will be the offspring of a male father and a female mother who are married
- no marriages between siblings or between cousins of any type have occurred
- all people in the family tree are connected

Further to those conditions, the following definitions apply:

- `father` and `mother` are the parents of a child
- `brother` and `sister` are male and female siblings with the same parents
- `son` and `daughter` are the children of a parent
- `uncle` and `aunt` are the brother and sister of a child's parent
- `nephew` and `niece` are the male and female children of a sibling
- a `grandfather` and `grandmother` are the male and female parents, respectively of a child's parent
- a `great grandfather` and `great grandmother` are the father and mother, respectively, of a child's grandparent
- a `great uncle` or `great aunt` is a sibling to a child's grandparent
- `cousins` (not removed) are at the same level in the family tree
  - first cousins have the same grandparents
  - second cousins have the same great grandparents
  - and so on

- removed cousins are at different levels in the family tree
  - a `first cousin once removed` is the child of one of the first cousins
  - a `first cousin twice removed` is the grandchild of one of the first cousins
  - and so on
- cousin relationships are symmetric e.g. if A is the first cousin twice removed of B, B is also the first cousin twice removed of A
- where a relationship occurs due to marriage of two people the relationship is said to be `in-law`
  - the parent of a person's husband or wife is a `father-in-law` or `mother-in-law`
  - the sibling of a person's husband or wife is a `brother-in-law` or `sister-in-law`
  - the cousin of a person's husband or wife is a `cousin-in-law`

Figure 1 displays the family tree described in the Sample Input. Your program should be able to say that `Claire and Carol are 1st cousins`, `Claire and Diva are 1st cousins 1-time removed`, and `Claire and Chris are cousins-in-law`. Your program should also be able to generate any other relationship combinations when queried.

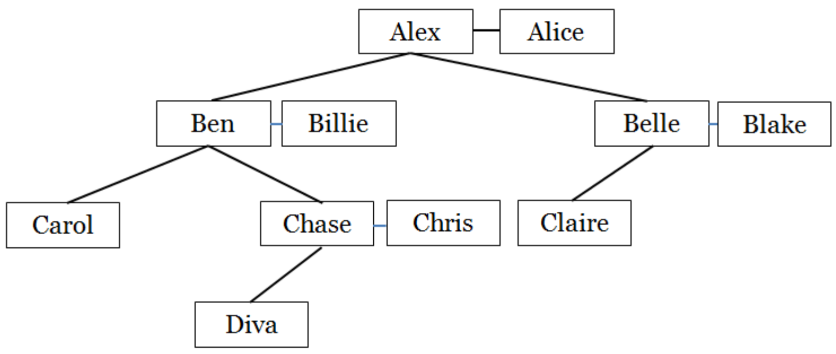


Figure 1: Sample Input

Input

The input contains a single test case.

The input consists of a list of relationships for construction of the family tree. The list of relationships will be followed by a list of queries for which you will name the relationship. Relationships will be provided to infer the gender of all family members.

All relationships will be in lower case and all names will be unique. At most one person in each marriage will have parents present in the input.

The first line of input contains a single integer r ($1 \leq r \leq 200$) being the number of relationships for building the family tree. r lines of relationship definitions follow. Each relationship consists of three alphabetic strings, $name_1$, $name_2$ and *relation*, each separated by a single space. *relation* will be one of `husband`, `wife`, `son` or `daughter`. The relationship line can be read as:

$name_1$ `is the` *relation* `of` $name_2$

The relationships are followed by a line containing a single integer q ($1 \leq q \leq 200$) being the number of queries on the family tree. q query lines follow. Each query line consists of two strings, $name_1$ and $name_2$ separated by a single space. The names in the queries will be contained in the family tree.

Output

For each relationship query, output the relationship between *name*$_1$ and *name*$_2$ on a single line.

In the following definitions mandatory items are delimited with `(` and `)`, optional items are delimited with `[` and `]`, options are separated by `|`. Elements which may require repetition (1 to many) are followed by `*`.

- For a spousal relationship i.e. `husband` or `wife`, output a sentence of the following form:

  name$_1$ `is the (husband|wife) of` *name*$_2$

- For a sibling relationship i.e. `brother` or `sister`, output a sentence of the following form:

  name$_1$ `is the (brother|sister)[-in-law] of` *name*$_2$

- If the relationship is some kind of cousin, output a sentence which includes the degree of cousinship i.e. `1st`, `2nd`, `3rd` etc. followed by the word `cousins`, then the suffix `-in-law` if and only if the relationship is by marriage and finally the number of times removed (`1-time removed`, `2-times removed`, `3-times removed`, etc.).

  name$_1$ `and` *name*$_2$ `are (1st|2nd|3rd|...) cousins[-in-law][ (1-time|2-times|3-times|...) removed]`

- If the relationship is aunt, uncle, nephew or niece, the output may require one or more instances of the word `great`.

  name$_1$ `is the [great ]*(aunt|uncle|nephew|niece)[-in-law] of` *name*$_2$

- Otherwise the relationship will be one of `son`, `daughter`, `father` or `mother`. Relationships which are two generations apart will require the use of the word `grand` before the relationship. Relationships which are more than two generations apart will require the use of one or more instances of the word `great` before the word `grand`.

  name$_1$ `is the [[great ]*grand](son|daughter|father|mother)[-in-law] of` *name*$_2$

Sample Input and Output

Sample Input	Output for Sample Input
10	Claire and Carol are 1st cousins
Alex Alice husband	Claire and Diva are 1st cousins 1-time removed
Ben Alex son	Claire and Chris are 1st cousins-in-law
Chase Ben son	Billie is the sister-in-law of Belle
Diva Chase daughter	Billie is the mother-in-law of Chris
Carol Ben daughter	Billie is the mother of Chase
Belle Alex daughter	Belle is the aunt of Carol
Chris Chase wife	Blake is the uncle-in-law of Carol
Blake Belle husband	Carol is the niece of Belle
Billie Ben wife	Carol is the niece-in-law of Blake
Claire Belle daughter	
10	
Claire Carol	
Claire Diva	
Claire Chris	
Billie Belle	
Billie Chris	
Billie Chase	
Belle Carol	
Blake Carol	
Carol Belle	
Carol Blake	

This page has intentionally been left almost blank

acm International Collegiate Programming Contest 2015 IBM event sponsor

SOUTH PACIFIC REGION

H: Banking

Time Limit: 1 second(s)

Internet banking sites have a variety of methods to authenticate their users. The methods usually involve passwords or Personal Identification Numbers (PINs) together with a mechanism to verify that a person is attempting to authenticate rather than a computer program.

The Actuarial Commerce Merchant bank has a scheme where, when you login, you are provided with a "pattern word", containing only upper and lower case letters. You must use this pattern word to extract and sum digits from your PIN as follows.

Letters in the pattern word are to be interpreted as numbers, with a (or A) = 1, b (or B) = 2, ... z (or Z) = 26. A lower case letter specifies a count of digits to extract from the PIN while an upper case letter specifies a counts of digits to be skipped. The letters in the pattern word are processed from left to right resulting in a sequence of extracted digits, which are added together to yield a number. You then enter that number into a field on the web page form to authenticate yourself. For example, if your PIN was `1093373`, and the pattern provided to you was `aBcA` you would extract one digit (namely 1) skip two digits (09), extract 3 digits (337) and then skip 1 digit (3), before totalling the extracted digits (1337) and entering 14 into the field on the web page form.

The bank allows you to have a PIN containing up to 256 digits and they intend to provide a pattern word in which the letters, when interpreted as numbers, sum to the length of the PIN. However, sometimes they get this wrong!

Write a program that reads a PIN and a pattern word and outputs the sum of the digits extracted from the PIN if the pattern is valid or outputs `non sequitur` if the length of the PIN and the length indicated by the pattern are different.

Input

The input contains a single test case.

The first line of input will contain an n-digit PIN, $6 \leq n \leq 256$. The second line will contain an m-digit pattern word containing only upper and lower case letters, $1 \leq m \leq 256$.

Output

The test case will produce one line of output being either the sum of the extracted digits from the PIN if the pattern word is valid or the text `non sequitur` if the pattern is invalid.

Sample Input and Output

Sample Input 1	Output for Sample Input
092384907653 bGc	23

Sample Input 2	Output for Sample Input
092384907653 bGb	non sequitur

This page has intentionally been left almost blank

acm International Collegiate Programming Contest 2015 IBM event sponsor

SOUTH PACIFIC REGION

I: Shelob's Lair

Time Limit: 10 second(s)

Sam Gamgee and Frodo Baggins are trapped in Shelob's lair. Shelob is a gigantic spider who lives in the caves at the edge of Mordor. Sam and Frodo are Hobbits, which means that they are little people with hairy feet.

The cave is a large rectangular cavern and Shelob has cast many great webs in the cave, and now Frodo and Sam (who are at the South wall of the cave) must reach the North wall to escape. If Sam and Frodo touch any of the web, they will become stuck and Shelob will come and eat them.

Sam has a magic sword, Sting, that can cut through Shelob's web. However, Frodo is poisoned and Sam is exhausted from their adventures, so Sam only has the strength to make one vertical slice through the web once. A well-chosen slice at a point will cut through all of the webs that pass through the point, allowing Sam and Frodo to pass. Sam and Frodo, being little people, can fit through an infinitely small slit.

Given the locations of all the webs in the cave, determine if it is possible for Sam and Frodo to escape.

We suppose that each web is a vertical sheet that runs from one point (given by Cartesian coordinates) to another point. The webs are fixed at the roof and the floor of the cave, and run in a straight line between the two points. Multiple webs can cross one another (Shelob is a skilled web spinner) and if Sam were to slice exactly where they cross he could slice all of the webs at once. No web touches the North or South wall of the cave. Sam is also able to cut at precisely the point one or more webs connect to the East or West walls of the cave. You may treat Frodo and Sam as a point, so they can fit through the vertical cut and can fit between two webs that do not intersect.

Input

The input contains a single test case.

The first line consists of three integers, w (the width of the cave), d (the depth of the cave), and n (the number of webs that are cast), where $1 \leq w, d \leq 1000$ and $1 \leq n \leq 500$.

Next, n lines follow where each line contains 4 integers, x_1, y_1, x_2, y_2, where $0 \leq x_1, x_2 \leq w$ and $0 < y_1, y_2 < d$. (x_1, y_1) is the Cartesian coordinates of one end of the web, and (x_2, y_2) is the Cartesian coordinate at the other end of the web. The coordinates are arranged so that the South-West corner of the cave is the point $(0, 0)$, and the North-East corner is the point (w, d).

Output

If it is possible for Frodo and Sam to reach the North wall, making at most one slice through the web, print the line:

```
We can make it Mr Frodo!
```

If it is impossible for Frodo and Sam to reach the North wall without making more than one slice through the web, print the line:

```
We're doomed Mr Frodo!
```

Sample Input and Output

Sample Input 1	Output for Sample Input

`5 5 3`
`0 3 5 3`
`0 1 5 4`
`0 2 5 4`
	`We're doomed Mr Frodo!`

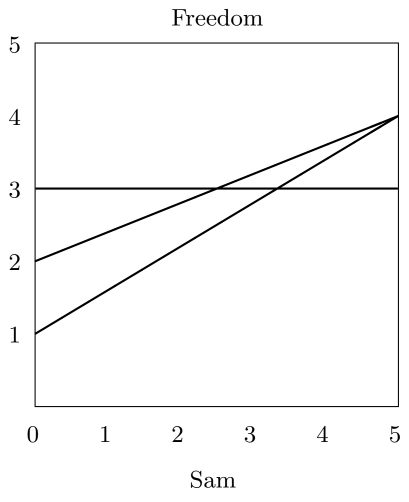
Sample Input 2	Output for Sample Input

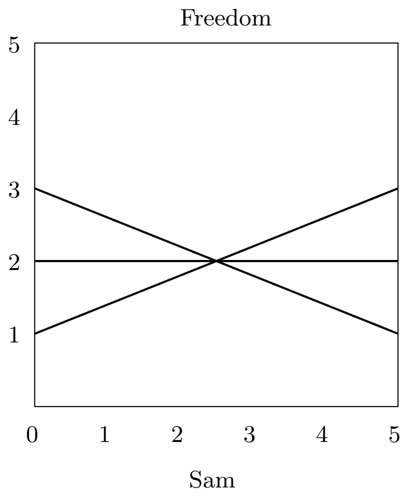
`5 5 3`
`0 1 5 3`
`0 3 5 1`
`0 2 5 2`
	`We can make it Mr Frodo!`

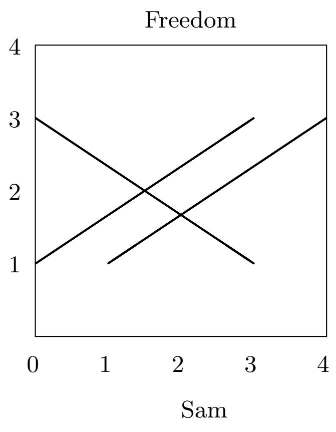
Sample Input 3	Output for Sample Input

`4 4 3`
`0 1 3 3`
`0 3 1 3`
`4 3 1 1`
	`We can make it Mr Frodo!`

The following diagrams are images for the Sample Inputs:







J: Painting Floors

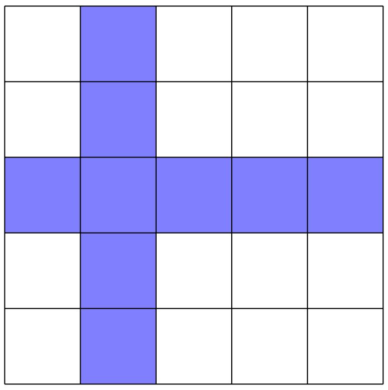
Time Limit: 1 second(s)

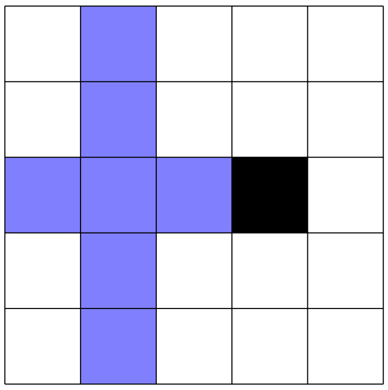
I really need to paint my floor! My floor is rectangular and has some furniture on it. Instead of hiring someone to paint it for me, I decided to do it myself. So I went out the local paint store, Antonio's Colourful Masterpieces, and asked for some paint. The man behind the counter asked me if I would like to try some experimental paint. Curious as to what could be experimental about paint, I said, "Yes!" I then purchased 1 000 paint cans and went home (it was on a very good sale).

When I got home, I opened the instruction manual for the paint and was extremely surprised:

> "When you pour the whole can of paint onto the ground, it will fill in the 1×1 block of floor it is in and then will expand out and fill every square that is in the same row or same column as the original 1×1 block so long as there is not an obstacle in the way."

For example, in the left room, the paint is poured in the third row and second column and it fills in coloured squares. In the right room, the paint is poured in the same square, but it only goes until it hits the black obstacle (piece of furniture).





You may not pour paint onto any of the furniture (obviously!). The goal is to paint every square in the room that is not furniture. Painted squares do not become obstacles for future pours of paint and it is okay to paint squares multiple times.

For small rooms, I can easily figure out how to paint the rooms with this experimental paint, but for large rooms, I'm worried that I will run out of paint before I finish the floor! Can you tell me where to pour the paint? You do not need to give me an optimal solution, but you must give me a solution that uses no more than the 1 000 paint cans that I purchased. It is guaranteed that 1 000 paint cans are enough to paint the floor.

Input

The input contains one test case.

The first line will contain two integers m and n ($1 \le m, n \le 500$) being the dimensions of my room in metres. The next m lines will contain n characters each. These lines will show the layout of my room. The map of the rooms will only contain the characters '`.`' and '`x`'. An '`x`' denotes an obstacle in the room and a '`.`' denotes no obstacle. There will be at most 500 `x`'s in the input.

Output

The output will consist of several lines. In each line, output two integers: the row and the column of where to pour the ith can of paint. The rows and columns are 1-based from the top-left corner. The number of lines of output must be no more than 1 000. The lines need not be in any particular order. Any valid output will be considered correct.

Sample Input and Output

Sample Input 1	Output for Sample Input

`5 5`
`.....`
`.....`
`.....`
`.....`
`.....`
	`1 1`
	`2 2`
	`3 3`
	`4 4`
	`5 5`

Sample Input 2	Output for Sample Input

`7 7`
`.......`
`.xxxxx.`
`.xx.xx.`
`.x...x.`
`.xx.xx.`
`.xxxxx.`
`.......`
	`1 1`
	`4 4`
	`7 7`

acm International Collegiate Programming Contest 2015 IBM event sponsor
SOUTH PACIFIC REGION

K: Selling Numbers

Time Limit: 10 second(s)

Revolutionising telephony is expensive business. That's why young entrepreneur Ace E. Emme is hoping to sell some of his trademarked Global Unique phone numbers first, and then direct the resulting cash at the technical hurdles to see what happens.

Global phone numbers will need to have plenty of digits, which makes it more important than ever to buy a number that is easy to recite from memory. To this end, each phone number is given a Memorisability Score. For a particular phone number, the score is determined as follows:

1. Initialise the score to zero.
2. For each substring of length L, add L to the score if the substring is a palindrome and $L \geq 2$. A palindrome is a sequence that reads the same backwards as forwards.
3. For each pair of non-overlapping substrings A and B, where B appears after A, and each is of length L, with $L \geq 2$, add L to the score for each and every one of the following conditions that holds:
   (a) $A = B$
   (b) $A = B$ and B is adjacent to A
   (c) A is equal to B in reverse.

Mr Emme is interested in pricing the phone numbers, therefore counting how many there are with a particular score is crucial for designating them as Gold Class, Diamond Class and Diamond Class Plus Plus.

Note that each rule on a given substring or pair of substrings is treated independently of the application of this or other rules to other substring(s). For instance, a palindrome of length 5 always contains a palindrome of length 3, as well as a match of rule 3(c). Therefore, the effective score for such a five-character substring will be at least 5 + 3 + 2. This is intentional, as longer patterns appear more lucrative to customers than multiple smaller patterns of the same total length, so a higher score is warranted.

Input

The input contains no more than 11 000 test cases.

Each test case will consist of two integers D ($0 < D < 12$) and S ($0 \leq S < 1000$) on a line, separated by a single space. This is a query asking how many phone numbers with D digits are there with Memorisability Score equal to S. Note that phone numbers with leading zeros are considered valid.

The input concludes with a pair of zeros on a line by itself.

Output

For each test case, print a sentence: "`Among D digit phone numbers, there are N with score S.`" Follow the format of the sample output.

Sample Input and Output

Sample Input 1	**Output for Sample Input**
`2 2`	`Among 2 digit phone numbers, there are 10 with score 2.`
`3 7`	`Among 3 digit phone numbers, there are 10 with score 7.`
`3 0`	`Among 3 digit phone numbers, there are 720 with score 0.`
`0 0`	