XXVIII Maraton Nacional de Programacion
Colombia - ACIS / REDIS 2014
ACM ICPC

Problems

(This set contains 10 problems; problem pages numbered from 1 to 19)

General Information

Unless otherwise stated, the following conditions hold for all problems.

Program name

1. Your source file (your solution!) must be called `<codename>.c`, `<codename>.cpp` or `<codename>.java`, as indicated below the problem title.

Input

1. The input must be read from standard input.
2. The input contains several test cases. Each test case is described using a number of lines that depends on the problem.
3. When a line of data contains several values, they are separated by single spaces. No other spaces appear in the input. There are no empty lines.
4. Every line, including the last one, has the usual end-of-line mark.
5. The end of input is indicated by the end of the input stream. There is no extra data after the test cases in the input.

Output

1. The output must be written to standard output.
2. The result of each test case must appear in the output using a number of lines that depends on the problem.
3. When a line of results contains several values, they must be separated by single spaces. No other spaces should appear in the output. There should be no empty lines.
4. Every line, including the last one, must have the usual end-of-line mark.
5. After the output of all test cases, no extra data must be written to the output.
6. To output real numbers, round them to the closest rational with the required number of digits after the decimal point. Ties are resolved rounding to the nearest lower value.

Problem A
Alternation Formulae

Source file name: `alternation.c`, `alternation.cpp` *or* `alternation.java`

For a positive integer n, let $S(n)$ be the string defined by the concatenation of the decimal notations (without leading zeroes!) of $1, 2, \ldots, n$. For instance, $S(11)$=1234567891011.

An (arithmetic) formula F is an *n-alternation* if it is built inserting in the string $S(n)$ arithmetic operators $+, -$ and parentheses $(,)$. Besides of that, it is required that the used arithmetic operators occur alternately in F.

An n-alternation, being an arithmetic formula, has an integer value. The following are two examples of 11-alternations with the indicated values:

$$1-(2+3)-4+5-6+7-8+9-1+0-11=-13$$
$$-1+2-3+4-5+6-7+89-1+011=95$$

Let's consider the following puzzle: given two integers n and m (n>0), decide if there exists an n-alternation F that evaluates to m. From the examples above it is clear that it is possible to build 11-alternations that evaluate to -13 and 95. However, it is easy to see that it is impossible to find a 3-alternation that evaluates to 10.

In order to be precise in the description of the required task, an (arithmetic) *formula* is defined as follows:

- The empty string is not a formula.
- A numeric string, i.e., a string made of digits `0` . . . `9`, with at most 5 of them, is a formula.
- If α and β are formulae, then $\alpha+\beta$ and $\alpha-\beta$ are formulae.
- If α is a formula, then $+\alpha$, $-\alpha$ and (α) are formulae.

Input

The input consists of several test cases, each one defined by a line containing two blank-separated integers n and m ($1 \le n \le 100$, $-10^7 \le m \le 10^7$).

The input must be read from standard input.

Output

For each test case, print a line with the character '`Y`' if there exists an n-alternation F that evaluates to m, or with the character '`N`', otherwise.

The output must be written to standard output.

Sample input	**Output for the sample input**

`11 -13`
`11 95`
`3 10`
	`Y`
	`Y`
	`N`

Problem B
Magic Squares

Source file name: `msquares.c`, `msquares.cpp` *or* `msquares.java`

According to *Wikipedia*, "a *magic square* of order n is an arrangement of n^2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant". This constant is the *module* of the magic square. There are well-known magic squares such as the order 3 chinese Lo Shu magic square:

4	9	2
3	5	7
8	1	6

It is allowed to use any collection of n^2 integer numbers to build a magic square of order n. The Passion façade of the Sagrada Família church in Barcelona, designed by Josep Subirachs, displays the magic square of order 4 and module 33 shown in the following figure. Note that, in this example, the given numbers are not the first n^2 integers and that there are repetitions.

1	14	14	4
11	7	6	9
8	10	10	5
13	2	3	15

Armadora de Cuadrados Magicos (*ACM*) is a recently founded enterprise that is interested on applications of magic squares to cryptography. For that reason, they want to develop software to help magic square builders in detecting if a given sequence of integer numbers may be arranged in a magic square. Your task is to help ACM in this task.

Input

The input consists of several test cases, each one defined by a line containing a sequence of m blank-separated integers x_1, x_2, ..., x_m ($1 \leq m \leq 16$, $-10^3 \leq x_i \leq 10^3$ for each $1 \leq i \leq m$).

The input must be read from standard input.

Output

For each test case, output a line with exactly one letter: '`Y`' to indicate that a magic square may be built with the numbers provided for the case, or '`N`' otherwise.

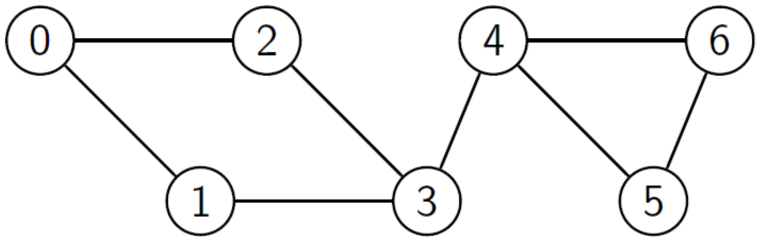
The output must be written to standard output.

Sample input	**Output for the sample input**
`1 2 3 4 5 6 7 8 9`	`Y`
`1 14 4 14 11 7 6 9 8 13 10 2 10 3 5 15`	`Y`
`4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 8`	`N`
`1 2 3 4`	`N`
`1 1 -1 -1`	`N`
`1 1 1 1`	`Y`
`-1 -1 -1 -1`	`Y`
`1 1 1`	`N`

Problem C
Weak Links

Source file name: `wlinks.c, wlinks.cpp` *or* `wlinks.java`

The Thought Police (TP) is trying to dismantle a network of thought-criminals (TC). As it may be easily inferred from the name, TCs commit thought-crimes and they do so by sending messages through a communication network that TCs believe to be secure, but which has already been infiltrated by the TP. In order to preserve the security of the communication network, each TC only communicates with a reduced set of contacts, so a message may need to go through several intermediate TCs to reach its destination. The TCs in the communication network are connected in such a way that a message from any source may reach any destination. Even though the TP knows all the TCs of the network, they want to interrupt its activity in the most subtle way. The plan is to identify the weak links of the network. A *weak link* is any single link that if removed, would make it impossible for at least one TC to communicate with some other TC in the network. For instance, the next figure shows a network with 7 TCs.



The connection between TCs 3 and 4 is a weak link since, if removed, it would make it impossible for TCs $0, 1, 2, 3$ to communicate with TCs $4, 5, 6$. All the other connections of the network are not weak links. The government has hired you to support its crusade to defend the country from thought-criminals and to preserve the security of its citizens, by helping the TP to carry on its plan identifying the weak links of the TCs communication network.

Input

The input contains several test cases. The first line of each test case contains two blank-separated integers n and m, where n is the number of TCs ($2 \leq n \leq 1000$), and m is the number of direct communications links between them ($1 \leq m \leq 10000$). Then m lines follow, each one containing two blank-separated integers N_i and N_j ($0 \leq N_i < N_j < n$), indicating that there is a direct communication link between TCs N_i and N_j in the communication network. Note that all communication links are bidirectional. A line with two zeros "`0 0`" indicates the end of the input.

The input must be read from standard input.

Output

For each test case, a line with the list of the weak links has to be printed.

Each line starts with a number p indicating the number of weak links in the communication network, then p links of the form N_{i_k} N_{j_k} must follow ($0 \le N_{i_k} < N_{j_k} < n$). Weak links must be printed in ascending order of their N_{i_k}. If two weak links have the same N_{i_k}, the link with the minimum N_{j_k} should be printed before. All numbers printed in each line should be separated by a single blank. There are no blanks after the last number of any line.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
4 3
0 3
0 1
0 2
7 8
0 1
0 2
1 3
2 3
3 4
4 5
4 6
5 6
0 0
```
	```
	3 0 1 0 2 0 3
	1 3 4
	```

Problem D
Don't Care

Source file name: `dontcare.c`, `dontcare.cpp` *or* `dontcare.java`

In mathematical logic and computer science, an *abstract rewriting system* (ARS) is a convenient framework for describing important properties of artifacts such as logical inference, software systems, social interaction, etc. In its simplest form, an ARS is a set *objects* together with a *binary relation* on the objects expressing how these can transition.

Some ARS are of very special interest because even if an object can make a transition in more than one way, these transitions will eventually yield the same result. Such an ARS guarantees that if some irreducible object is reached by successive transition steps, then this is also the case independently of the choice of the first transition taken. In other words, an ARS with this property converts nondeterminism from "don't know" into "don't care", thus avoiding the need for backtracking. Let us agree to call an ARS with this property a *Don't Care ARS.*

Consider the usual rules of elementary arithmetic: they form an abstract rewriting system. For example, the expression (11+9)×(2+4) can be evaluated starting either at the left or at the right parentheses; however, in both cases the same *irreducible* result is obtained eventually. This suggests that the arithmetic reduction system is don't care.

$$\begin{tikzcd}
\text{eval left} & (11+9)\times(2+4) \arrow[dl] \arrow[dr] & \text{eval right} \\
20\times(2+4) \arrow[dr] & & (11+9)\times 6 \arrow[dl] \\
\text{eval right} & 20\times 6 \arrow[d] & \text{eval left} \\
 & 120 &
\end{tikzcd}$$

Borrowed from Wikipedia.org

There is no general purpose algorithm that can *always* determine if a given ARS is don't care, i.e., checking the don't care property is in general undecidable for arbitrary ARS. However, because of its importance, this property has been thoroughly studied and decision procedures have been obtained for restricted versions of the problem. In particular, the don't care property can always be checked mechanically for systems having a finite number of objects.

Let $\mathcal{A}=(A,\to)$ denote an ARS with a set of objects A and a transition relation $\to$ on A (that is, $\to\subseteq A^2$). Then, $\mathcal{A}$ is don't care if and only if the following two conditions are satisfied:

1. the relation $\to$ is well-founded, i.e., no object can be reduced with $\to$ indefinitely, and
2. for any $a, b, c\in A$, if there are direct $\to$-transitions from a to b and from a to c, then there is $d\in A$ such that d is reachable via (zero or more) $\to$-transitions both from b and c.

You have been assigned the task of implementing an efficient algorithm for deciding the don't care property for finite ARS.

Input

The input consists of several test cases. The first line of each test case contains two numbers n and m ($1 \leq n \leq 1000$, $0 \leq m \leq 30000$) defining, respectively, the number of objects and the number of transitions in the ARS. Each of the next m lines contains a pair of blank-separated integers a, b indicating that there is a transition in the ARS from object a to object b ($0 \leq a < n$, $0 \leq b < n$). The last test case is followed by a line with two zeros "`0 0`".

The input must be read from standard input.

Output

For each ARS output exactly one line. If the input ARS is don't care, then output '`1`'; otherwise, output '`0`'.

The output must be written to standard output.

Sample input	**Output for the sample input**
`3 2`	`1`
`0 1`	`1`
`1 2`	`0`
`2 2`	`0`
`0 1`	
`0 1`	
`2 2`	
`0 1`	
`1 0`	
`3 2`	
`0 1`	
`0 2`	
`0 0`	

Problem E
Emacs Plugin

Source file name: `emacs.c, emacs.cpp` *or* `emacs.java`

Emacs is a text editor characterized by its extensibility via plugins. Sometimes Emacs is described as "the extensible, customizable, self-documenting, real-time display editor". Nowadays, Emacs is one of the main contenders in the traditional editor wars of Unix culture.

As a seasoned veteran in the competitive programming scene, Emacs is most of the time your editor of choice. You wish you could always use it but this is not possible because of a feature that is currently not working as expected: a very specific text search capability. In particular, you have gathered proof that searching a text against a regular expression pattern takes a considerable amount of time for some special kinds of patterns. As usual, to the extents of your ego, you are certain that you can implement a much better specific purpose algorithm than the general purpose one currently shipping with Emacs.

A *text* is a string made from lowercase letters ('`a`' to '`z`'). A *pattern* is a string made from lowercase letters and the wild-card symbol ('`*`'). The pattern p *matches* a text t iff p can be found in t as a substring, with each symbol '`*`' in p witnessing an arbitrary substring in t.

For example, let t='`heyhelloyou`', p_1='`hel*`', p_2='`*o*e`', and p_3='`e*o`'. Note that p_1 matches t with '`*`' witnessing, for example, the substring '`loyo`' or even the empty string ''. Also, p_3 matches t with, for example, witness '`lloy`'. However, p_2 does not match t.

You have decided to implement a new algorithm for matching a pattern to a text as part of a new Emacs plugin. Can you come up with a very efficient algorithm?

Input

The input consists of several test cases. Each test case begins with an integer number n indicating the number of patterns on a single line ($1 \leq n \leq 50$). The next line contains the text t ($1 \leq |t| \leq 10^5$). The next n lines contain the patterns $p_1, p_2, \ldots, p_n$ ($1 \leq |p_i| \leq 10^5$ for each $1 \leq i \leq n$), each on a single line.

The input must be read from standard input.

Output

For each test case output n lines, where the i-th line indicates whether p_i matches t or not. If p_i matches t, then output '`yes`'; otherwise, output '`no`'.

The output must be written to standard output.

Sample input	Output for the sample input
`4`	`yes`
`heyhelloyou`	`no`
`hel*`	`yes`
`*o*e`	`yes`
`e*o`	`no`
`hello`	
`1`	
`hello`	
`x`	

Problem F
Friendship Networks

Source file name: `fnets.c, fnets.cpp` *or* `fnets.java`

For a group of N people registered at a social network, a *friendship network* can be constructed with the N people as the nodes and a non-directed edge between every pair of friends. For such a network, it is usual to say that a node, i.e., a person, has degree d iff there are d nodes connected to it. Note that the degree of a node is exactly his/her number of friends. Since nobody is friends with himself/herself, the degree of each node is less than N.

A key property of a friendship network is the number of friends for each one of its members, i.e., its members' degrees. However, given an intended number of network friends for each one of the members, such a network may or may not exist. For example, for a group of 4 people there is a friendship network in which the people have 3, 2, 2, and 3 friends. However, there is no friendship network corresponding to the enumeration 1, 3, 2, and 3 for a group of 4 people.

You are working at a company that researches friendship networks with the business idea of eventually developing applications on them. Your specific job is part of a test data validation. More precisely, you must write a program that given an enumeration of positive integers, decides if there exists a friendship network with such number of friends for each of its members.

Input

The input consists of several cases. Each test case is given in a single line by a blank-separated list of integers $N, d_1, d_2, \ldots, d_N$, with N $(2 \leq N \leq 1000)$ the number of people in the social network and the d_i $(1 \leq i \leq N)$ an enumeration of a possible friendship network. You can assume that $0 < d_i < N$ for $1 \leq i \leq N$.

The input must be read from standard input.

Output

Output a single line for each test case. If $d_1, d_2, \ldots, d_N$ describes a friendship network with such number of friends for each of its members, then output 1; otherwise output 0.

The output must be written to standard output.

Sample input	**Output for the sample input**
`4 3 2 2 3`	`1`
`4 1 3 2 3`	`0`
`8 2 5 4 5 4 3 5 2`	`1`

Problem G
Looking-Glass House

Source file name: `lookingglass.c`, `lookingglass.cpp` *or* `lookingglass.java`

There was a book lying near Alice on the table, ..., she turned over the leaves, to find some part that she could read, ' -- for it's all in some language I don't know,' she said to herself. It was like this.

YKCOWREBBAJ
sevot yhtils eht dna ,gillirb sawT'
;ebaw eht ni elbmig dna eryg diD
,sevogorob eht erew ysmim llA
.ebargtuo shtar emom eht dnA

She puzzled over this for some time, but at last a bright thought struck her. 'Why, it's a Looking-glass book, of course! And if I hold it up to a glass, the words will all go the right way again.'
This was the poem that Alice read.

JABBERWOCKY
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

Lewis Carroll, Through the Looking Glass

Some few days later, Alice noticed that when she said 51, some people understood 15 and, some others, 51. And when she said 43, they understood 34 or 43. Of course, such a risk of misunderstanding is a serious trouble to play arithmetic games. Then she realized that there was no problem saying 343, because in this case the others would always understand what she meant.

But, what can she do if she wanted to say 51, anyway? She discovered that she could say the number in base 2, since 51_2=110011. She could say a number n in a base b if n was written in that base as a *palindrome*, i.e., as a number that reads the same from left to right than from right to left. For example, neither 43_{10}=43 nor 43_2=101011 are palindromes, but 43_6=111 is a palindrome.

Your task is to write a program to help Alice calculating the smallest base $b \geq 2$ for which a given number n is a palindrome, if there is such a base b. The number n is given in base 10.

Input

The input contains several test cases, each one described by one single line with an integer number n, written in base 10 ($1 \leq n \leq 10^6$).

The input must be read from standard input.

Output

For each test case, output a line with the smallest base $b \geq 2$ in which the number n is written as a palindrome, if there is such a base. If there is not such a base, answer a line with a 0 value.

The output must be written to standard output.

Sample input	Output for the sample input
51 43 7 19	2 6 2 18

Problem H
Smooth Factor

Source file name: `smooth.c, smooth.cpp` *or* `smooth.java`

Sage is a bright shining star when it comes down to sorting stuff; she can handle multiple sorting scenarios in a fashion that can only be described as "awesome, spectacular, and, above all else, awesometacular". By performing sorts of all sorts during the years, Sage has developed interesting skills that she plans on using for sorting information efficiently in the technology world, where the "big bucks" are.

Currently, Sage is working on a mysterious heuristic that attempts to sort data in two passes. The first one is the *smoothing phase* and the second one is the *sorting phase*. The general idea is that the smoothing phase leaves the data almost ordered so that the sorting phase can be done very efficiently.

Sage is in the process of testing and validating the effectiveness of the sorting phase. She has developed a quantitative measure on an array of values called the *smooth factor*. For an array $a = a_1, a_2, \ldots, a_n$ of values, the smooth factor of a is the length of a longest subarray $a_p, \ldots, a_q$ of a ($1 \le p \le q \le n$) such that there is at most one i ($p < i \le q$) such that $a_{i-1} > a_i$, where $<$ is some fixed order on the values in a. The higher the smooth factor, the more effective the smooth phase is.

For example, under the natural order of integer numbers, for the array $1, 2, 3$ the smooth factor is 3 and for the array $1, 2, 1, 2, 1, 2, 3, 1$ the smooth factor is 5. In the first case the entire array serves as a witness and in the second case the witness is $1, 2, 1, 2, 3$.

Sage has gathered some output sample arrays of the smoothing process and would like to know how effective this process is before she signs any deals with the information moguls. In other words, for a given input array, she wants to compute its smooth factor. Can you help her?

Input

The input consists of several test cases. Each test case consists of two lines. The first line of a test case contains an integer n ($1 \le n \le 10^5$) indicating the length of the sample array. The second line contains n blank-separated integer numbers $a_1, \ldots, a_n$ (with $0 \le |a_i| < 10^8$).

The input must be read from standard input.

Output

For each test case, output a line with the smooth factor of the array $a_1, \ldots, a_n$.

The output must be written to standard output.

Sample input	Output for the sample input

```
3
1 2 3
1
0
8
1 2 1 2 1 2 3 1
4
1 -10 -100 -100
```
	```
	3
	1
	5
	3
	```

Problem I
Space Invaders

Source file name: `invaders.c, invaders.cpp` *or* `invaders.java`

It is year 2551. The *Nostalgia for Infinity*, captain John Brannigan's ship, is cruising towards Resurgam, a planet considered a backwater on the edge of colonized human space. Some of the scientists aboard the ship will lead a team excavating the remains of the Amarantin, a long-dead, 900,000-year-old civilization that once existed on Resurgam. It is suspected that the entire Amarantin race achieved a much higher level of technological sophistication than was previously known.

As the Nostalgia for Infinity approaches Cerberus, a planet near Resurgam, the ship's defenses detect the presence of a highly hostile alien ship. Captain Brannigan is ready to fight and has stationed the ship behind a shield that seems to have been previously built by the Amarantin. Nostalgia for Infinity instruments depict space battle scenarios in $2D$. In particular, the shield is depicted as a rectangular region with r rows and c columns. Each cell in a shield can be either *active* (represented by '`#`') or *disabled* (represented by '`.`'). It is said that a shield is *breached* iff there is a connected sequence of disabled cells between the first and the last rows of the shield. Two cells are *connected* iff they are adjacent vertically or horizontally. For example, the figure below depicts two shields of r=4 rows and c=5 columns (numbers added for clarity: on top, they indicate the column number; on the right, they indicate the row number). The shield on the left is not breached, while the one on the right is breached.

```
12345          12345
##..# 1        ##..# 1
..### 2        ...## 2
#.#.. 3        #.#.. 3
..##. 4        ..##. 4
```

If it is assumed that the first row of a shield is facing north, then the alien ship is located north of the first row and it fires in the south direction; also, the Nostalgia for Infinity is located south of the last row and it fires in the north direction. In this way, the shield is a barrier between the two enemy ships. The effect of a shot fired from a ship is to disable the first active cell it encounters in the shield, if any. Shots are indicated by the column on which they are fired and always follow a vertical path. For example, if the aliens shoot at either column $1, 2, 3, 4$ the left shield depicted above would be breached. The right shield corresponds to the result of shooting column 3.

You, as the ship's new gunnery officer, have been assigned with the task of determining whether or not, during battle, the shield is breached. If so, captain Brannigan would also like to know when and who breached the shield first.

Input

The input consists of several test cases. The first line of a test case contains three blank-separated integers r, c, and s ($1 \leq r \leq 1000$, $1 \leq c \leq 1000$, $0 \leq s \leq 10000$) indicating, respectively, the number of rows and columns in the shield, and the number of shots during the battle. Each of the next r lines contain c characters '`#`' or '`.`'. The next s lines contain each an integer a ($1 \leq |a| \leq c$) indicating a shot fired at column $|a|$ of the shield: if $a>0$ then it is an alien shot; otherwise it is Nostalgia for Infinity shot.

The input must be read from standard input.

Output

For each test case, output exactly one line:

- If the shield was breached before any shot was fired, then output `0`.
- If the shield was never breached during the battle, then output `X`.
- If the shield was first breached by the Nostalgia for Infinity during the battle at shot k (counting from 1), then output the number $-k$.
- If the shield was first breached by the aliens during the battle at shot k (counting from 1), then output the number k.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
2 2 2
.#
..
1
1
2 3 2
##.
.##
2
-2
2 3 2
##.
.##
-2
2
2 1 1
#
#
-1
```
	```
	0
	-2
	2
	X
	```

Problem J
The "Win-stay and Lose-shift" Strategy

Source file name: `wsls.c, wsls.cpp` *or* `wsls.java`

In recent weeks, geek tabloids have hit the newsstands around the world with a truly remarkable breakthrough in science: a group of researchers from Chinese universities have written a paper about the role of psychology in winning (or losing) at rock-paper-scissors (RPS). After studying how players change or keep their strategies during multiple-round sessions, the scientists figured out a basic rule that people tend to play by that could potentially be exploited. This rule is called the *win-stay and lose-shift* strategy.

An RPS *session* is a finite sequence of rounds played between two opponents. In each *round*, players simultaneously form one of three shapes with an outstretched hand: *rock* (`R`), *paper* (`P`), and *scissors* (`S`). Rock beats scissors, scissors beat paper, and paper beats rock; if both players throw the same shape, the round is tied. The *outcome of a round* for a player is 1 point if he/she wins, -1 point if he/she loses, and 0 points if it is a tie. The *outcome of a session* for a player is the sum over the outcome points of his/her rounds. For example, assume that a and b are playing a session of three rounds. In the first round a plays scissors and b plays paper; in the second round a plays paper and b plays paper; and, in the last round a plays rock and b plays rock. Then, the outcome of the first round for a is 1 point (for b is -1 point), and the outcomes of the second and third rounds for a is 0 points (for b is also 0 points). Consequently, the outcome of this session for a is 1 point and for b is -1 point.

During an RPS session, the *win-stay and lose-shift* strategy for a player p is as follows:

- If it is the first round or if it was a tie in the previous round, for the current round p makes a guess.
- If p lost in the previous round, for the current round p switches to the thing that beats p's opponent previous choice.
- If p won in the previous round, for the current round p switches to the thing that beats p's previous choice.

For example, assume that a and b are playing a session of three rounds, and a is playing under the win-stay and lose-shift strategy and that b plays as above. Initially a guesses `R` and loses (`P` beats `R`). In the second round a switches to `S` because it beats b's previous winning choice (i.e., `P`) and wins (`S` beats `P`). In the third round a switches to `R` because it beats a's previous choice (i.e., `S`) and ties (b also plays `R`). In this session the outcome for a is 0 points. However, this is not the only possible outcome for a under the win-stay and lose-shift strategy.

Given a session of n rounds for players a and b, and the probabilities of a guessing `R`, `P`, and `S` during the session, you are asked to write a program that decides if a's *expected* session outcome when playing under the win-stay and lose-shift strategy against b is better than a's actual session outcome.

Input

The first line of the input contains a non-negative integer number N ($N \geq 0$) indicating the number of test cases. Then N test cases follow, each consisting of three lines of input. The first and second lines of a test case contain, respectively, strings a and b only containing characters `R`, `P`, and `S` ($1 \leq |a| \leq 10^4$, $1 \leq |b| \leq 10^4$, with $|a| = |b|$) defining an RPS session of $|a|$ rounds played between players a and b. The third line of a test case contains three blank-separated integer numbers p_R, p_P, and p_S ($0 \leq p_R \leq 100$, $0 \leq p_P \leq 100$, $0 \leq p_S \leq 100$, with $p_R + p_P + p_S = 100$) indicating, respectively, the probability (amplified by 100) of a guessing rock, scissors, and paper.

The input must be read from standard input.

Output

For each test case output a single line containing three blank-separated quantities of the form

```
x y z
```

where

- x is an integer indicating a's actual session outcome against b,
- y is a floating point number indicating a's expected session outcome when playing against b with probabilities p_R, p_P, and p_S under the win-stay and lose-shift strategy (rounded up to exactly 4 decimal places, with no leading zeroes but at least one digit before the decimal point), and
- z is the character `Y` if y is strictly greater than x, and `N`, otherwise.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
4
SPR
PPR
5 80 15
RRR
PPR
5 80 15
S
S
33 34 33
S
S
34 33 33
```
	```
	 1 0.3060 N
	-2 0.3060 Y
	 0 -0.0100 N
	 0 0.0100 Y
	```