A — Ardenia

Welcome to Ardenia. Ardenia is a mythical land, filled with adventure and danger, dwarves and dragons, mages and rouges. And puzzles. Lots of puzzles. Actually, the love for puzzles is the most important life ingredient of the inhabitants, and the only part they have in common.

This month, the people of Ardenia wonder what is the distance between two line segments in three dimensional space. (The distance between segments is defined as the minimum among distances between two points of different segments.) Actually, this problem had originally some motivation, but as nobody from Ardenia cares about motivations, neither should you.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 10^5$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains six space-separated integers $x_1, y_1, z_1, x_2, y_2, z_2 \in [-20, 20]$. Points (x_1, y_1, z_1) and (x_2, y_2, z_2) are the (different) endpoints of the first segment. The second line contains six integers in the same format, describing the second segment.

Single Instance Output

You should output a single line consisting of two co-prime integers ℓ and $m > 0$, such that ℓ/m is the squared distance between the given two segments.

Example

Input	Output

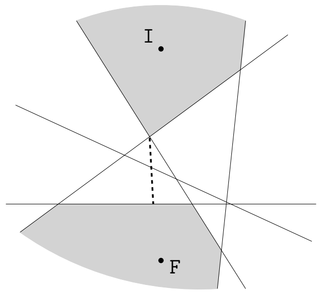
2
0 0 0 1 1 1
1 1 1 2 2 2
1 0 0 0 1 0
1 1 0 2 2 0
	0 1
	1 2

B — Beasts

The Fiery Beast and the Ice Beast are relicts of the early days of Ardenia. They live in their lairs and they are kept in the lairs' neighborhoods by powerful magic lines drawn by the Elders. These lines must not be crossed by the beasts. However, in the recent years some of these lines vanished. And if the beasts came too close to each other, the consequences for the whole human race would be catastrophic.

You, the king of the land, are certainly worried about it and gathered all the information about the existing magic lines. It appears that the land can be treated as an infinite plane (this fact was established by Herman the Wise, who traveled for many days in one direction and was still able to see new things!) The lair of the Ice Beast is at point $(0, 10^{10})$ and the lair of the Fiery Beast is at point $(0, -10^{10})$. There are several magic straight infinite lines and neither Fiery nor Ice Beast can cross any of it. You want to know what is the minimum distance which always separates the beasts.

An example is presented below. `I` and `F` denote the beasts' lairs and the gray regions denote the areas on which they may freely walk. Dotted segment corresponds to the minimum distance between these areas.



Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 20$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains one positive integer $n \leq 2 \cdot 10^5$, being the number of magical lines. Each of the next n lines contains three space-separated integers a, b, c, such that $-10^9 \leq a, b, c \leq 10^9$ and $b \neq 0$. These numbers denote the magic line $ax + by + c = 0$.

Single Instance Output

You should output a single line containing one number, being the square of the minimum distance between beasts. Your result is going to be accepted if and only if it is accurate to within a relative or absolute value of at most 10^{-5}.

Example

Input	Output

2
5
1 -1 0
1 1 0
0 1 -6
0 1 -10
0 1 10
4
1 1 10
2 6 7
-1 2 10
0 1 12
	400.000000
	93.250000

C — Casting Spells

Casting spells is the least understood technique of dealing with real life. Actually, people find it quite hard to distinguish between a real spells like “abrahellehhelleh” (used in the battles and taught at the mage universities) and screams like “rachelhellabracadabra” (used by uneducated witches for shouting at cats).

Finally, the research conducted at the Unheard University showed how one can measure the power of a word (be it a real spell or a scream). It appeared that it is connected with the mages’ ability to pronounce words backwards. (Actually, some singers were burned at the stake for exactly the same ability, as it was perceived as demonic possession.) Namely, the power of a word is the length of the maximum subword of the form ww^Rww^R (where w is an arbitrary sequence of characters and w^R is w written backwards). If no such subword exists, then the power of the word is 0. For example, the power of `abrahellehhelleh` is 12 as it contains `hellehhelleh` and the power of `rachelhellabracadabra` is 0. Note that the power of a word is always a multiple of 4.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 40$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The input is one line containing a word of length at most $3 \cdot 10^5$, consisting of (large or small) letters of the English alphabet.

Single Instance Output

You should output one integer k being the power of the word.

Example

Input	Output
2 abrahellehhelleh rachelhellabracadabra	12 0

D — Defense Lines

After the last war devastated your country, you — as the king of the land of Ardenia — decided it was high time to improve the defense of your capital city. A part of your fortification is a line of mage towers, starting near the city and continuing to the northern woods. Your advisors determined that the quality of the defense depended only on one factor: the length of a longest contiguous tower sequence of increasing heights. (They gave you a lengthy explanation, but the only thing you understood was that it had something to do with firing energy bolts at enemy forces).

After some hard negotiations, it appeared that building new towers is out of question. Mages of Ardenia have agreed to demolish some of their towers, though. You may demolish arbitrary number of towers, but the mages enforced one condition: these towers have to be consecutive.

For example, if the heights of towers were, respectively, 5, 3, 4, 9, 2, 8, 6, 7, 1, then by demolishing towers of heights 9, 2, and 8, the longest increasing sequence of consecutive towers is 3, 4, 6, 7.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 25$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The input instance consists of two lines. The first one contains one positive integer $n \leq 2 \cdot 10^5$ denoting the number of towers. The second line contains n positive integers not larger than 10^9 separated by single spaces being the heights of the towers.

Single Instance Output

You should output one line containing the length of a longest increasing sequence of consecutive towers, achievable by demolishing some consecutive towers or no tower at all.

Example

Input	Output

2
9
5 3 4 9 2 8 6 7 1
7
1 2 3 10 4 5 6
	4
	6

E — Enter The Dragon

The capital of Ardenia is surrounded by several lakes, and each of them is initially full of water. Currently, heavy rainfalls are expected over the land. Such a rain falls to one of the lakes: if the lake is dry and empty, then it will be filled with water; if the lake is already full, then it will overflow, which will result in a natural disaster. Fortunately, the citizens have a dragon at their disposal (and they will not hesitate to use it). The dragon may drink the whole water from a lake in one sitting. Also, the mages of Ardenia already predicted the weather conditions for the next couple of years. The only question is: from which lake and when should the dragon drink to prevent a catastrophe?

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 40$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains two space-separated positive integers $n \leq 10^6$ and $m \leq 10^6$, where n is the number of lakes. (There are at most 10 input instances for which $n \geq 10^5$ or $m \geq 10^5$.) The second line contains the weather forecast for the next m days: m space-separated integers $t_1, t_2, \ldots, t_m$ ($t_i \in [0, n]$). If $t_i \in [1, n]$, it means a heavy rainfall over lake t_i at day i. If $t_i = 0$, there is no rain at day i, and the dragon has the time to drink the water from one lake of your choice. Note that the dragon does not drink on a rainy day.

Single Instance Output

In the first line your program should output word `YES` if it is possible to prevent a catastrophic overflow and `NO` otherwise. In the former case, you should output the second line containing ℓ integers from the range $[0, n]$, where ℓ is the number of zeros in the weather forecast description, i.e., the number of non-rainy days. Each of these integers denotes the number of the lake from which the dragon should drink; zero means the dragon should not drink from any lake (this might be necessary, as even the dragon cannot drink from an empty lake).

Example

Input	Output

```
4
2 4
0 0 1 1
2 4
0 1 0 2
2 3
0 1 2
2 4
0 0 0 1
```
	```
	NO
	YES
	1 2
	NO
	YES
	0 1 0
	```

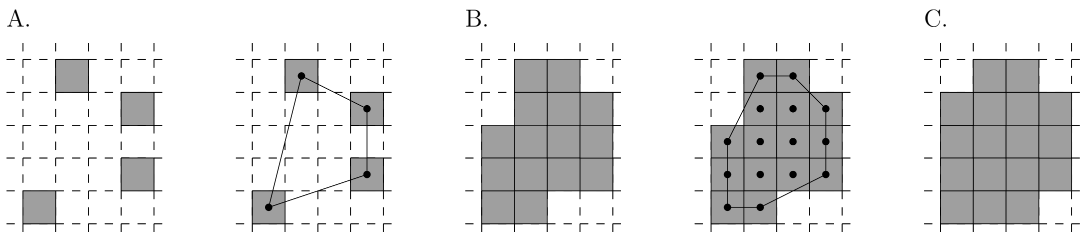
F — Fields and Farmers

Being a farmer in Ardenia is a tough job. We do not mean only the dry and hostile environment where they have to pasture sheep. The government (or actually the king himself) wants his people to invade foreign lands rather than to harvest theirs, and thus tries to make the lives of poor farmers as hard as possible. All difficulties start with a seemingly simple task: purchasing a piece of land, called farming parcel.

The whole farming territory is a huge rectangular grid consisting of square fields; a farming parcel consists of some of these fields. At the beginning a farmer buys an *initial set of fields*; his parcel consists initially just of these fields. However, the actual farming parcel is determined with the help of string and poles, by repeating the following steps.

1. Stick a pole into the middle of each field from the farming parcel.
2. Surround the poles with a string, creating the smallest region enclosing all the poles.
3. The new farming parcel is the set of all fields having a non-empty intersection with this region. A field sharing just the edge or a corner with the region does not count.

Of course, the parcel may only increase by implementing the operation above, so each farmer makes sure these steps are repeated till the farming parcel does not change; we call such a parcel *final*. An example is depicted below. The initial farming parcel consists of four fields (figure A), after one iteration it grows (figure B), and after another one it becomes final (figure C).



It appears, however, that the final farming parcel would sometimes be the same even if the farmer did not buy all the initial fields but just a subset of them. A subset of this property is called *valid*. The farmer wants to know in how many ways he may choose a valid subset of initial fields.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 50$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains a positive integer $n \leq 10^6$, being the number of initial fields of the parcel. Each of the following n lines contain two integers x_i, $y_i \in [-10^9, 10^9]$, being the coordinates of these fields. All initial fields are different.

Single Instance Output

Let k be the number of valid subsets of the initial fields. You should output a line containing the number $k \mod (10^9 + 7)$.

Example

Input	Output

2
4
0 0
0 1
0 2
0 3
5
0 0
-1 0
1 0
0 -1
0 1
	4
	2

G — Game

Tic-tac-toe is the third most popular activity to kill a lazy afternoon in Ardenia (right after solving puzzles and insulting your neighbors). Arthum and Breece are not fans of this game, but their mother told them to play, so they sit at a 5×5 board. Both have a large pile of marbles: marbles of Arthum have an A written on them and that of Breece have a B. However, as they are both two years old, they have no concept of rounds. Instead, they just toss their marbles as quick as possible, so after a while each board field has either marble A or marble B.

At that point, they would like to determine the winner, but counting is not their strong point, either. (They are two years old, remember?) Recall that the goal of tic-tac-toe is to have three own marbles in a row, i.e., lying at three consecutive fields horizontally, vertically or diagonally. If both Arthum and Breece have their three marbles in a row, or neither of them has it, we call it a draw.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 10^5$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The input instance describes marbles placed on the board in a single game. The instance consists of 5 rows and each of them consists of 5 letters: `A` or `B`.

Single Instance Output

You should output one line describing the outcome of the game, i.e., one of the three possible strings: `A wins`, `B wins`, or `draw`.

Example

Input	Output
2 AABBA BAAAB AAABA ABAAB BAAAB AAAAA AAAAA BAAAA ABAAA AABAA	A wins draw

H — Hanging Hats

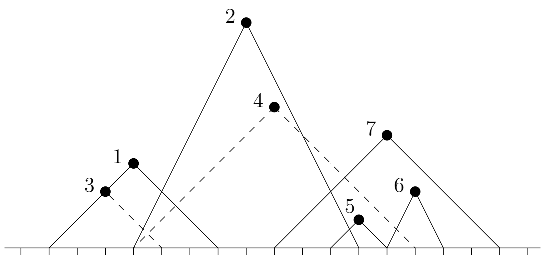
It is a well-known fact that all mages wear pointed hats. However, contrary to popular belief, mages do not sleep in them, and those of the Unheard University hang them on one common, very large wall before going to bed. Mages' hats come only in two shapes: a *wide* one and a *narrow* one.

The mages go to sleep one after another (it is easily assured as due to a tight budget, they have only one bathroom and one towel). Before going to bed, the i-th mage takes its hat and chooses an arbitrary position (x_i, y_i) on the wall, where y_i is the (positive) distance from the ground. Then he hammers a nail at position (x_i, y_i) and hangs his hat on this nail.

Not surprisingly, the hats are magical, which means that they magically unfold to a given height. Hence, when hung, a hat looks exactly like a isosceles triangle of height y_i, i.e., its left and right edges are of equal length. Its top vertex is exactly at the nail and its bottom edge touches the floor. If the hat is narrow, then the length of its bottom edge is y_i; if it is wide, it is equal to $2y_i$.

All nails are provided by the university and they have nice heads which glow in the dark. If a nail becomes covered by a hat hung later (even by its boundary), its glow is no longer visible. Moreover, if a mage tries to hammer a nail at an already hanging hat (also even at its boundary), this nail and his hat are thrown away, and he himself becomes expelled from the university. The Chancellor of the University (apparently not having more important things to worry about) wants to know how the number of visible glowing nail heads changes in time.

An example consisting of 7 hats is presented below. Dots correspond to glowing nail heads; numbers represent the order in which they are hung. Hats 2 and 6 are narrow, the remaining ones are wide. Mages 3 and 4 were expelled while trying to hang their hats (their hats were not hung). After mage 7 hung his hat, hats of mages 1, 2 and 7 were visible.



Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 30$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The first line of the input instance contains $n \in [1, 10^5]$, the number of mages. The i-th of the following n lines contains two space-separated integers $x_i \in [-10^9, 10^9]$ and $y_i \in [1, 10^9]$, followed by a letter `W` or `N`. These numbers are the coordinates where the i-th mage tries to hammer a nail; the letter denotes whether his hat is wide or narrow.

Single Instance Output

For each input instance you should output n lines. The i-th line should contain the word `FAIL` if by hammering the nail the i-th mage got himself expelled from the university. Otherwise, it should contain a positive integer, the number of nail heads visible after the i-th mage hangs his hat.

Example

Input	Output
`2`	`1`
`3`	`1`
`0 1 W`	`FAIL`
`0 2 N`	`1`
`0 1 W`	`2`
`7`	`FAIL`
`4 3 W`	`FAIL`
`8 8 N`	`3`
`3 2 W`	`4`
`9 5 W`	`3`
`12 1 W`	
`14 2 N`	
`13 4 W`	

I — Insults

Insulting your friends and neighbors is the Ardenia national sport. However, as every sport, it has a certain set of rules which are quite hard to learn even by natives, not to mention the tourists. First of all, the insults are single words consisting entirely of four vowels: `a`, `e`, `i`, and `o`. But not all words that consist of these letters are insults. The only two letter insults are `ae` and `io`. If words w_1 and w_2 are insults, then words $w_1 w_2$ and `a`w_1`e` and `i`w_1`o` are insults as well. Insults are created only in such way.

Usually, if somebody is insulting you, then you better have a sharp response insult prepared. Obviously, most of the answers are inappropriate. For example if you hear `aaeeio` (meaning *you fight like a dairy farmer*) you should not reply with `aeio` (*you are stupid*). To everybody's surprise, the linguists have found out that in all the cases the most appropriate reply is the insult of the same length and alphabetically next. Thus, for the insult above, the best reply would be `aaeioe` (*how appropriate; you fight like a cow*). This rule implies also the existence of so called ultimate insults, i.e., the ones for which there is no good reply. The eight letter ultimate insult is `ioioioio` (*your mother was a hamster and your father smelt of elderberries*).

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 2000$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The input instance is one line containing a string of length at most 10^6. The only letters occurring in the string will be `a`, `e`, `i` and `o`.

Single Instance Output

You should output a single line containing the word `INVALID` if the string is not an insult. Otherwise, you should output the best reply to the insult read or word `ULTIMATE` if no such reply exists.

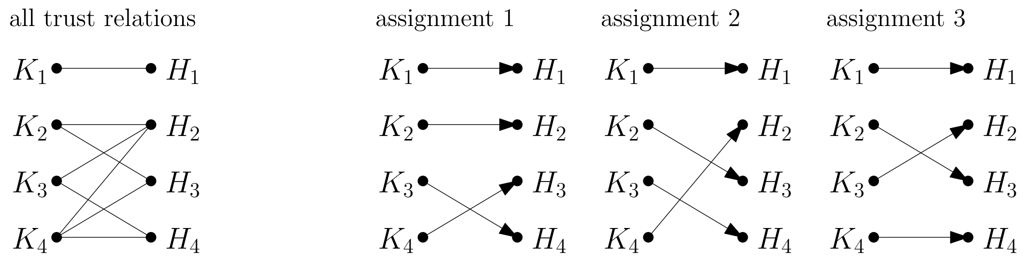
Example

Input	Output
3 eaeeio aaeeio ioioioioio	INVALID aaeioe ULTIMATE

J — Justice for All

Ardenia is going to war! The famous and prestigious Ardenia's military unit consisting of 200 knights and 200 horses started to prepare for battles. During the preparation, k knights and k horses are chosen (the remaining ones simply stay at their barracks) and a special kind of mutual *trust* relation is established between certain knights and certain horses. In such case, we simply say that knight A *trusts* horse B (and vice versa). One horse can trust arbitrarily many knights and one knight can trust arbitrarily many horses.

For a given team of k knights and k horses, the trust relations between them determines the number of battles they are willing to fight. For each battle, each knight chooses a single horse he or she trusts: this creates an *assignment*. An example of 4 knights (K_1, K_2, K_3 and K_4) and 4 horses (H_1, H_2, H_3 and H_4) with trust relations is depicted below. There are 3 possible different assignments.



The assignments for two different battles have to be different (the knights would get bored otherwise) and all possible assignments have to be tried out (the knights are curious enough to test them all). They are pretty good at their fighting skills, which means that no knight or horse will be harmed during the making of the war.

Your mages predicted that the war would consist of n battles. Your task is to choose k knights and k horses and establish trust relations between them, so that they are prepared for exactly n battles.

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 100$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

The input instance is one line containing the number of battles, $n \in [1, 10^6]$.

Single Instance Output

The first line of the output should contain a single positive integer $k \leq 200$. Each of the next k lines should consist of k binary digits (`0` or `1`, without spaces between them), where `1` in the j-th column of the i-th line means that knight i trusts horse j (and vice versa). The number of possible battles these knights and horses are prepared for should be exactly n.

Example

Input	Output
3	1
1	1
3	4
4	1000
	0110
	0101
	0111
	4
	1100
	1100
	0011
	0011