Problem C. Catch Me If You Can

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

This is an interactive problem.

Carl Hanratty is a dour FBI agent usually pursuing criminals in the financial sector, but he also likes to play his Pokemon GO game in a park during his time off.

He still plays an old school version of the game with a working pokemon radar. This radar shows one, two or three paws telling him how far the pokemon is from the current player's location. Three paws means that the pokemon is not farther than $3 \cdot r$ meters from the player, two paws mean that it is not farther than $2 \cdot r$ meters, and one paw means not farther than r meters. It is known that the value of r is a real number between 1 and 100 meters, inclusive, but the exact value of r is unknown to Carl.

Since Carl is a very good FBI agent and always strives for excellence, he wants to improve his pokemon catching practice by utilizing a secret FBI analytical task force. You are a part of that analytical team. You will receive field information from Carl, and then direct his movements in order to find and catch the pokemon.

You may assume all the movements are happening on a two-dimensional plane. Carl is located at the point with coordinates $(0, 0)$, and he just saw the pokemon on the radar for the first time (with three paws). This means Carl is initially located $3 \cdot r$ meters away from the pokemon. To communicate with Carl, you are allowed to perform no more than 5 iterations of the following process:

1. You transmit to Carl an angle in degrees: the direction in which he should move. The angle might be any real value from 0 to 360 inclusive.
2. Carl goes straight in this direction until the radar indication changes. This happens if he catches the pokemon (then the game stops), reaches the radar zone that is closer to the pokemon (the number of paws on the radar decreases by 1) or reaches the radar zone that is farther from the pokemon (or pokemon disappears from the radar at all). In the last case, Carl immediately makes a step back to the previous three-paw zone, as he doesn't want to move in a direction which is definitely wrong.
3. Carl tells you the exact distance he traveled and the current number of paws p $(1 \le p \le 3)$ of the radar zone. It means the current distance to the pokemon is $p \cdot r$. Note that the communication counts even if Carl returned a distance of 0 which means that walking any positive distance in the given direction will immediately result in radar indication change.

During your 5-th communication with Carl, his behavior is a bit different. As he knows he won't be able to get any instructions from you anymore, he ignores all the radar zones and moves straight in the direction you gave him until either he finds the pokemon or it totally disappears from the radar.

It is guaranteed that the pokemon location is fixed and does not change during each session of the game. It is also guaranteed that the unknown real value r lies between 1 and 100, inclusive.

Interaction Protocol

You have at most 5 communication attempts. To start the next attempt, your program must provide a direction to Carl by printing one real value a $(0 \le a \le 360)$ on a separate line. After that, your program gets the result on the next line of the standard input.

If at any moment of time Carl is within $r/10$ from the actual pokemon location, he instantly catches it and returns you one final word "`Gotcha!`" (without quotes).

Otherwise, if you run out of communication sessions with Carl and he was unable to find the pokemon during the last move, he sends you one final line "`The pokemon ran away!`" (without quotes).

In all other cases, Carl transmits you the traveled distance d with at least ten digits after the decimal point and the integer number of paws p ($1 \le p \le 3$) of the new radar zone. These two numbers are separated by one space character.

Your communication with Carl must stop when the pokemon is caught or ran away.

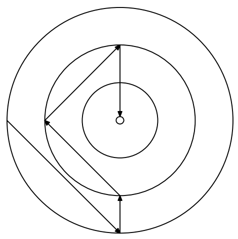
Examples

standard input	standard output
	`90`
`0.0000000000 3`	
	`0`
`3.0000000000 2`	
	`270`
`0.0000000000 2`	
	`90`
`0.0000000000 2`	
	`0`
`Gotcha!`	
	`315`
`14.1421356237 3`	
	`90`
`3.3333333333 2`	
	`135`
`9.4280904158 2`	
	`45`
`9.4280904158 2`	
	`270`
`Gotcha!`	

Note

In the first sample input, pokemon is sitting at point $(9, 0)$.

In the second sample input, pokemon is sitting at point $(10, 0)$.



The pipe from your program to the interactor program and the pipe back have limited size. Your program must read from the standard input to avoid deadlock. Deadlock condition is reported as "`Time Limit Exceeded`".

To flush the standard output stream, use the following statements:

In C, use `fflush(stdout)`;

In C++, use `cout.flush()`;

In Java, use `System.out.flush()`;

In Python, use `sys.stdout.flush()`.

If your program receives an `EOF` (end-of-file) condition on the standard input, it MUST exit immediately with exit code 0. Failure to comply with this requirement may result in "`Time Limit Exceeded`" error.

Problem D. Demolition Time

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Darkwing Duck is in trouble again! As soon as he returned to his town, it was announced that Negaduck has been attacking the place. As a proper villain, he is going to rob some set of **consecutively** placed buildings and escape with the loot.

The panorama of the town can be represented as a string s where each character represents the shape of the corresponding building. Different buildings may have the same shape, and hence are represented in the string by equal characters. Negaduck has picked some string p as his robbery plan. Fortunately, this string p has become known to the police. Now they want to find all the buildings which are in danger, but there is one complication on the way.

By the order of the new Mayor, some of the buildings in the town should be demolished. They are demolished one after another **in the order they appear** in the string s: from left to right. This demolition plan is well known in the city.

The robbery can take place anytime: before all planned demolitions, after them, or between the demolition of any two buildings which follow one another in the demolition plan. However, no demolitions can happen during the robbery.

Negaduck robs buildings strictly according to his robbery plan from left to right. He does not skip any buildings except for demolished ones. He can execute only the whole plan. Nevertheless, there can exist a lot of subsets of buildings which are in danger of being robbed.

Formally, consider string t we get from s by removing the characters corresponding to already demolished buildings. Note that string t is initially equal to s but changes after each demolition. Negaduck can rob a subset of buildings if there exists a moment of time when this subset forms a substring of t, and this substring is equal to string p.

Your task is to find the exact amount of subsets which are in danger of being robbed.

Input

The first line of input contains a non-empty string s consisting of lowercase and uppercase English letters, digits, characters '`,`', '`!`', '`_`', '`.`' and '`-`' representing the buildings in the town from left to right. The length of this string doesn't exceed 1 000 000.

The second line of input contains a string p consisting of lowercase and uppercase English letters, digits, characters '`,`', '`!`', '`_`', '`.`' and '`-`' representing the Negaduck's plan from left to right. The length of this string doesn't exceed 1 000 000.

Note that uppercase and lowercase English letters are considered different.

The third line of input contains one integer m ($0 \le m \le |s|$) — the length of the Mayor's demolition plan.

The next line contains an **increasing** sequence of integers $x_1, x_2, \ldots, x_m$ ($1 \le x_1 < \ldots < x_m \le |s|$) — the indices of buildings in order they should be demolished.

Output

Output one integer: the number of different subsets of buildings that can be robbed according to Negaduck's plan.

Example

standard input	standard output
aabbcc abc 3 2 4 5	2

Note

The answer for the given sample is 2 because the following possible sequences of buildings can be robbed: 1 3 5 (after two building demolitions) and 1 3 6 (after all three building demolitions).

Problem F. Format

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

In C/C++, format "`%[..]`" is used by `scanf` to read a string consisting of characters defined by this format string.

The left bracket is followed by a sequence of characters and a right bracket. The sequence f of characters between the brackets defines a set of accepted characters. Only characters with ASCII codes from 32 (space) to 126 ('`~`') inclusive are acceptable there. In this problem, the characters '`^`', '`]`' and '`-`' are used in format only as control characters.

If the first character of sequence f is not a circumflex ('`^`'), then the sequence f defines the set of available characters s, and the function reads all characters up to the first character that is not in the set s or the end of the input. If the first character of sequence f is '`^`', then it must be followed by a **non-empty** sequence of characters defining the set of forbidden characters s, and the function reads all characters up to the first character in the set s or the end of the input.

A group of characters with sequential ASCII codes may be abbreviated as an *interval*: first character in the group, '`-`' (minus sign, ASCII code 45), last character in the group. For example, interval "`B-Q`" defines a set of uppercase English letters from '`B`' to '`Q`' inclusive.

A string is called *alphanumeric* if it consists only of upper- and lowercase English letters, digits and spaces. Given an alphanumeric string t, build the "`%[..]`" format which:

1. Accepts string t: when this string is given as input, scanf reads it fully.
2. Accepts the least possible number of other alphanumeric strings of length $|t|$.
3. Has the minimum possible length (as a string) among all format strings that conform to all the points above.
4. Is lexicographically smallest among all format strings that conform to all the points above.

Input

The first line of input contains one non-empty string t consisting only of upper- and lowercase English letters, digits and spaces. It is guaranteed that this string is no longer than 100 000 characters, does not have leading or trailing spaces, and does not have two or more consecutive spaces.

Output

Print the answer as a format string "`%[..]`" (without quotes).

Examples

standard input	standard output
`bc0123456789ABCDEFxxyyzz`	`%[!-Fbcx-z]`
`012345678 abcdefABCDEF`	`%[^9G-Zg-z]`
`01234567 abcdefABCDEF`	`%[ -7:-F[-f]`

Note

Note that the format string may contain non-alphanumeric characters. For example, in sample 1, '`!`' (character with ASCII code 33) is used instead of '`0`' because the number of alphanumeric characters between '`!`' and '`F`' and between '`0`' and '`F`' is the same, but format string with '`!`' is lexicographically smaller.

String a is lexicographically smaller than string b if b is not a prefix of a and one of two conditions hold:

1. a is a prefix of b.
2. Let k be the first positions where a and b differ. The ASCII code of the character at position k in the string a is smaller than ASCII code of the character at position k in the string b.

Hint:

32		48	0	64	@	80	P	96	‘	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		

Problem H. Hockey Cup

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Recently the World Hockey Cup was held once again, 12 years after the previous one! During the group stage, all teams were divided into groups, four teams in each of them. The group stage is played using a round robin system: each team plays with each other team exactly once. Thus, each group will feature six games in total. After all games are played, two best teams from each group advance to the knockout stage. We are only interested in the group that contains the Russian team.

Currently five out of six games were already played and their results are known. We want to check whether the Russian team is guaranteed to advance to the next stage, or at least has a chance to advance.

The rules for the group stage of the competition are the following. Each team is playing exactly one game against each of its opponents. If the game's prime time finishes with a tie (teams' scores are equal), then teams play overtime **until one of them scores**, and that team is declared the winner of the game. Each team gets 2 points for each victory, 1 point for each loss in overtime and no points for losses in the prime time.

At the end of the round robin, teams are ordered by the total number of points earned. If two or more teams have the same number of points, the tie-breaking procedure is applied.

If only two teams are tied, the tie is always broken by the result of their head-to-head game. If there are three or four teams tied, the following values are calculated (the bigger the better) based on the results **of all games played by the team**, not only games between the tied teams.

1. The total number of wins.
2. The total number of wins in prime time.
3. The goal differential: the number of goals scored by the team minus the total number of goals conceded by the team.
4. The total number of goals scored.

The rules are applied **one by one** until all four-way and three-way ties are broken. At this point, if any two-way ties remain, each of them will automatically be broken by the head-to-head game between the two tied teams (see two-way tie criteria outlined above) and not by continuing on with any additional steps.

If there is still a tie among some teams after all the above steps, those teams will be ranked in **random order**.

Note that in the last game, each team can score **any** number of goals.

Input

The first five lines of input contain the results of the first five games, each on its own line. Each of these lines contains four or five tokens separated by single spaces: $team_A$ $team_B$ $goals_A$ $goals_B$ ($team_A \neq team_B$; $goals_A \neq goals_B$; $0 \leq goals_A, goals_B \leq 10$; $team_A, team_B \in$ {"`Russia`", "`Sweden`", "`Finland`", "`NA`"}). If the teams were playing overtime in this game, the corresponding line ends with the fifth token "`OT`" separated by a single space character.

It is guaranteed that each two teams were playing against each other at most once, and the number of goals in each overtime game differs exactly by 1.

The last line contains the names of two teams that are about to play the last game in that group separated by a single space character. It is guaranteed that they haven't played their game yet.

Output

If the team with name "Russia" is advancing to knockout stage irrelevant of the results of the group's last game, output a single line with text "`Already in playoff!`" (without quotes).

Otherwise, if the team with name "Russia" is not advancing to knockout stage no matter what the result of the group's last game is or what the result of random ranking is, output a single line with text "`No chance`" (without quotes).

Otherwise, output a line with text "`Believe in playoff!`" (without quotes).

Examples

standard input	standard output

`Russia Sweden 1 2`
`Finland NA 1 4`
`NA Russia 3 4`
`Finland Sweden 0 2`
`NA Sweden 4 3 OT`
`Finland Russia`
	`Believe in playoff!`

`Russia Sweden 1 0 OT`
`Finland NA 0 5`
`NA Sweden 5 0`
`Finland Russia 0 1 OT`
`Russia NA 0 5`
`Sweden Finland`
	`Already in playoff!`

`Russia Sweden 0 1`
`Finland NA 0 1`
`NA Russia 0 5`
`Sweden Finland 0 1`
`Russia Finland 0 1`
`NA Sweden`
	`No chance`

`Russia Sweden 0 4`
`Finland NA 0 1`
`NA Sweden 0 1`
`Finland Russia 4 0`
`Sweden Finland 1 0`
`Russia NA`
	`Believe in playoff!`

`Russia Sweden 0 1`
`Finland NA 10 0`
`Sweden Finland 0 10`
`Russia NA 1 0`
`Finland Russia 10 0`
`NA Sweden`
	`Believe in playoff!`

Problem L. Lazy Coordinator

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Boris is a contest coordinator on WWWforces, an online platform that conducts regular "What? Where? When?" competitions. There are two types of events in his life:

1. Someone submits a contest proposal (a set of questions) to Boris. He puts this new proposal in the waiting pool.
2. The time comes to organize a new contest. As Boris doesn't remember the initial order of proposals in his pool, he just picks a random proposal, removes it from the pool and uses it for the upcoming contest. Each proposal in the pool is picked with the same probability, regardless on how long it already waits.

There will be exactly n events of the first type and exactly n events of the second type. Moreover, for the k-th event of the second type, there will be at least k events of the first type preceding it. In other words, there will be at least one proposal in the pool any time the coordinator needs it. Finally, it is guaranteed that no two events happen simultaneously.

For each proposal, you should determine the expected time it will stay in the waiting pool.

Input

The first line of the input contains a single integer n ($1 \le n \le 100\,000$) — the number of events of each type.

Then $2n$ lines follow that describe the events. Each line is either "`+` t_i" or "`-` t_i" meaning that an event of the first or the second type (respectively) occurs at the moment t_i.

It is guaranteed that all t_i are positive integers not exceeding 10^9, and the events will be given in the order of strictly increasing t_i. Moreover, for the k-th event of the second type, there will be at least k events of the first type preceding it. Finally, it is guaranteed that there will be exactly n events of the first type and exactly n events of the second type.

Output

For each proposal (event of the first type), print the expected time it will stay in the waiting pool. Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}.

Formally, assume that your answer is a, and the answer of the jury is b. The checker program will consider your answer correct if $\frac{|a-b|}{\max(1,b)} \le 10^{-6}$.

Examples

standard input	standard output
2	9.0000000000
+ 1	7.0000000000
+ 3	
- 8	
- 12	
4	1.0000000000
+ 1	1.0000000000
- 2	1.0000000000
+ 3	1.0000000000
- 4	
+ 5	
- 6	
+ 7	
- 8	
3	31.5000000000
+ 4	25.5000000000
+ 10	44.0000000000
- 11	
+ 16	
- 20	
- 100	

Note

In the first sample, Boris receives all two proposals first and then uses them in two contests. Each proposal has the probability $\frac{1}{2}$ to be used in each of the contests. Thus, the expected waiting times are $(8-1)\cdot 0.5+(12-1)\cdot 0.5 = 3.5+5.5 = 9$ for the first proposal and $(8-3)\cdot 0.5+(12-3)\cdot 0.5 = 2.5+4.5 = 7$ for the second one.

In the second sample, each proposal is almost immediately used to conduct a contest.