Problem A. Access Control Lists

Input file:	`access.in`
Output file:	`access.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Nick is developing a new web server. The feature he is working on now is support for *access control lists*. Access control list allows to restrict access to some resources on the web site based on the IP address of the requesting party.

Each IP address is a 4-byte number that is written byte-by-byte in a decimal dot-separated notation "byte0.byte1.byte2.byte3" (quotes are added for clarity). Each byte is written as a decimal number from 0 to 255 (inclusive) without extra leading zeroes. IP addresses are organized into IP networks.

IP network is described by two 4-byte numbers — network address and network mask. Both network address and network mask are written in the same notation as IP addresses.

In order to understand the meaning of network address and network mask you have to consider their binary representation. Binary representation of IP address, network address, and network mask consists of 32 bits: 8 bits for byte0 (most significant to least significant), followed by 8 bits for byte1, followed by 8 bits for byte2, and followed by 8 bits for byte3.

IP network contains a range of 2^n IP addresses where $0 \le n \le 32$. Network mask always has $32 - n$ first bits set to one, and n last bits set to zero in its binary representation. Network address has arbitrary $32 - n$ first bits, and n last bits set to zero in its binary representation. IP network contains all IP addresses whose $32 - n$ first bits are equal to $32 - n$ first bits of network address with arbitrary n last bits.

For example, IP network with network address 194.85.160.176 and network mask 255.255.255.248 contains 8 IP addresses from 194.85.160.176 to 194.85.160.183 (inclusive).

IP networks are usually denoted as "byte0.byte1.byte2.byte3/s" where "byte0.byte1.byte2.byte3" is the network address and s is the number of bits set to one in the network mask. For example, the IP network from the previous paragraph is denoted as 194.85.160.176/29.

Access control list contains an ordered list of rules. Each rule has one of the following forms:

- "`deny from <IP network>`" — denies access to the resource to any IP from the specified IP network.
- "`deny from <IP address>`" — denies access to the resource to the specified IP address.
- "`allow from <IP network>`" — allows access to the resource to any IP from the specified IP network.
- "`allow from <IP address>`" — allows access to the resource to the specified IP address.

When some party requests some resource its IP address is first checked against its access control list. The rules are scanned in order they are listed, and the first matching rule is applied. If none of the rules matches the IP address of the party, access is granted.

Given access control list and the list of requesting IP addresses, find out for each request whether it will be granted access to the resource.

Input

The first line of the input file contains n — the number of rules in the access control list ($0 \le n \le 100\,000$). The following n lines contain rules, one per line. IP network is always specified as "byte0.byte1.byte2.byte3/s".

The next line contains m — the number of IP addresses to check ($1 \le m \le 100\,000$). The following m lines contain IP addresses to check, one per line.

Output

For each request output '`A`' if it will be granted access to the resource, or '`D`' if it will not be granted access. Output all answers in one line, do not separate output by spaces.

Example

access.in	access.out
4 allow from 10.0.0.1 deny from 10.0.0.0/8 allow from 192.168.0.0/16 deny from 192.168.0.1 5 10.0.0.1 10.0.0.2 194.85.160.133 192.168.0.1 192.168.0.2	ADAAA

Problem B. Billboard

Input file:	`billboard.in`
Output file:	`billboard.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

At the entrance to the university, there is a huge rectangular billboard of size $h \times w$ (h is its height and w is its width). The board is the place where all possible announcements are posted: nearest programming competitions, changes in the dining room menu, and other important information.

On September 1, the billboard was empty. One by one, the announcements started being put on the billboard.

Each announcement is a stripe of paper of unit height. More specifically, the i-th announcement is a rectangle of size $1 \times w_i$.

When someone puts a new announcement on the billboard, she would always choose the topmost possible position for the announcement. Among all possible topmost positions she would always choose the leftmost one.

If there is no valid location for a new announcement, it is not put on the billboard (that's why some programming contests have no participants from this university).

Given the sizes of the billboard and the announcements, your task is to find the numbers of rows in which the announcements are placed.

Input

The first line of the input file contains three integer numbers, h, w, and n ($1 \le h, w \le 10^9$; $1 \le n \le 200\,000$) — the dimensions of the billboard and the number of announcements.

Each of the next n lines contains an integer number w_i ($1 \le w_i \le 10^9$) — the width of i-th announcement.

Output

For each announcement (in the order they are given in the input file) output one number — the number of the row in which this announcement is placed. Rows are numbered from 1 to h, starting with the top row. If an announcement can't be put on the billboard, output "`-1`" for this announcement.

Example

billboard.in	billboard.out

`3 5 5`
`2`
`4`
`3`
`3`
`3`
	`1`
	`2`
	`1`
	`3`
	`-1`

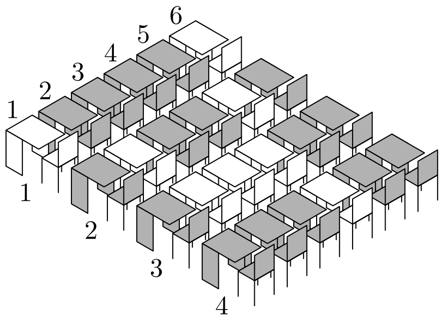
Problem C. Class

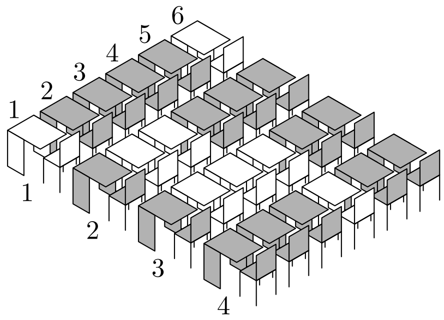
Input file:	`class.in`
Output file:	`class.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Dr. Strange is a really strange lecturer. Each lecture he calculates class fullness and if it is small, he decreases all semester grades by one. So the students want to maximize the class fullness.

The *class fullness* is the minimum of row fullness and column fullness. The *column fullness* is the maximum number of students in a single column and the *row fullness* is the maximum number of students in a single row.

For example there are 16 students shown on the left picture (occupied desks are darkened). The *row fullness* of this arrangement is 5 (the 4-th row) and the *column fullness* is 3 (the 1-st, the 3-rd, the 5-th or the 6-th columns). So, the *class fullness* is 3. But if the students rearrange as shown on the right picture then the *column fullness* will become 4 (the 5-th column), and so the *class fullness* will also become 4.





The students of Dr. Strange need to know the arrangement that maximizes *class fullness* so they ask you to write a program that calculates it for them.

Input

The first line of the input file contains three integer numbers: n, r and c — number of students, rows and columns in the class ($1 \le r, c \le 100$, $1 \le n \le r \times c$).

Output

The first line of the output file must contain a single integer number — the maximum possible *class fullness*.

The following r lines must contain the optimal student arrangement. Each line must contain a description of a single row. Row description is a line of c characters either "`.`" or "`#`", where "`.`" denotes an empty desk, and "`#`" denotes an occupied one. If there are multiple optimal arrangements, output any one.

Example

class.in	class.out

`16 4 6`	`4`
	`.####.`
	`#..###`
	`#...##`
	`###.##`

Problem D. Deposits

Input file:	`deposits.in`
Output file:	`deposits.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Financial crisis forced many central banks deposit large amounts of cash to accounts of investment and savings banks in order to provide liquidity and save credit markets.

Central bank of Flatland is planning to put n deposits to the market. Each deposit is characterized by its amount a_i.

The banks provide requests for deposits to the market. Currently there are m requests for deposits. Each request is characterized by its length b_i days.

The regulations of Flatland's market authorities require each deposit to be refinanced by equal integer amount each day. That means that a deposit with amount a and a request with length b match each other if and only if a is divisible by b.

Given information about deposits and requests, find the number of deposit-request pairs that match each other.

Input

The first line of the input file contains n — the number of deposits ($1 \le n \le 100\,000$). The second line contains n integer numbers: $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$).

The third line of the input file contains m — the number of requests ($1 \le m \le 100\,000$). The forth line contains m integer numbers: $b_1, b_2, \ldots, b_m$ ($1 \le b_i \le 10^6$).

Output

Output one number — the number of matching pairs.

Example

deposits.in	deposits.out
4 3 4 5 6 4 1 1 2 3	12

The following pairs match each other: $(3,1)$ twice (as (a_1, b_1) and as (a_1, b_2)), $(3,3)$, $(4,1)$ twice, $(4,2)$, $(5,1)$ twice, $(6,1)$ twice, $(6,2)$, and $(6,3)$.

Problem E. Enchanted Mirror

Input file:	`enchanted.in`
Output file:	`enchanted.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Alice likes two things in this world — her mirror and her toy bricks. Alice's toy bricks were designed to help the children to learn the alphabet, so there are some letters written on their top faces. Alice likes to play with the bricks near the mirror.

When Alice learned the alphabet, she noticed that something was wrong with her mirror! A brick in the mirror can show a different letter on it. Alice enjoyed this thing very much, and she invented a new game, trying to make some funny words from the bricks in the real world and in the mirror simultaneously.

The rules of this game are the following. Alice creates a line from some bricks that shows the word S_1. This line is shown in the mirror as some word S_2, which may be different from the reflection of S_1 because the mirror is enchanted. But the length of each of these words is equal to the same integer number N.

Then Alice can repeat the following step. She selects some two bricks i and j and swaps them. The reflected Alice in the mirror does exactly the same with the mirrored line, except that she of course swaps the bricks with positions $N - i + 1$ and $N - j + 1$ in it.

The goal is to create word T_1 in the real world simultaneously with the word T_2 in the mirror. Alice wonders whether it is possible and she asks you for help. Write a program which can determine whether the goal can be achieved.

Input

The input file contains four words S_1, S_2, T_1 and T_2, in this order, each on the separate line. All words have the same length N ($1 \le N \le 100$) and consist only of uppercase English letters.

Output

If the goal can be achieved, output "`Yes`". Otherwise output "`No`".

Example

enchanted.in	enchanted.out
TEAM TIED MATE EDIT	Yes
TEAM MATE TAME MEAT	No
AAAA AAAA AAAA AAAA	Yes

Problem F. Fenwick Tree

Input file:	`fenwick.in`
Output file:	`fenwick.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Fenwick tree is a data structure effectively supporting *prefix sum* queries.

For a number t denote as $h(t)$ maximal k such that t is divisible by 2^k. For example, $h(24) = 3$, $h(5) = 0$. Let $l(t) = 2^{h(t)}$, for example, $l(24) = 8$, $l(5) = 1$.

Consider array $a[1], a[2], \ldots, a[n]$ of integer numbers. Fenwick tree for this array is the array $b[1], b[2], \ldots, b[n]$ such that

$$b[i] = \sum_{j=i-l(i)+1}^{i} a[j].$$

So

$$\begin{aligned}
b[1] &= a[1], \\
b[2] &= a[1] + a[2], \\
b[3] &= a[3], \\
b[4] &= a[1] + a[2] + a[3] + a[4], \\
b[5] &= a[5], \\
b[6] &= a[5] + a[6], \\
&\ldots
\end{aligned}$$

For example, the Fenwick tree for the array

$$a = (3, -1, 4, 1, -5, 9)$$

is the array

$$b = (3, 2, 4, 7, -5, 4).$$

Let us call an array *self-fenwick* if it coincides with its Fenwick tree. For example, the array above is not self-fenwick, but the array $a = (0, -1, 1, 1, 0, 9)$ is self-fenwick.

You are given an array a. You are allowed to change values of some elements without changing their order to get a new array a' which must be self-fenwick. Find the way to do it by changing as few elements as possible.

Input

The first line of the input file contains n — the number of elements in the array ($1 \le n \le 100\,000$). The second line contains n integer numbers — the elements of the array. The elements do not exceed 10^9 by their absolute values.

Output

Output n numbers — the elements of the array a'. If there are several solutions, output any one.

Example

fenwick.in	fenwick.out
6 3 -1 4 1 -5 9	0 -1 1 1 0 9

Problem G. Ground Works

Input file:	`ground.in`
Output file:	`ground.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

The *Hilbert Mole* is a small and very rare mole. The first and only specimen was found by David Hilbert at his backyard. This mole lives in a huge burrow under the ground, and the border of this burrow forms a Hilbert curve of n-th order (H_n).

Hilbert curves can be defined as follows. H_1 is a unit square with open top side (fig. 1a), H_n consists of four copies of H_{n-1}: bottom left and bottom right are copied without changes, top left is rotated 90° counter-clockwise and top right is rotated 90° clockwise. These small copies are connected by three segments of unit length (fig. 1b,c,d).

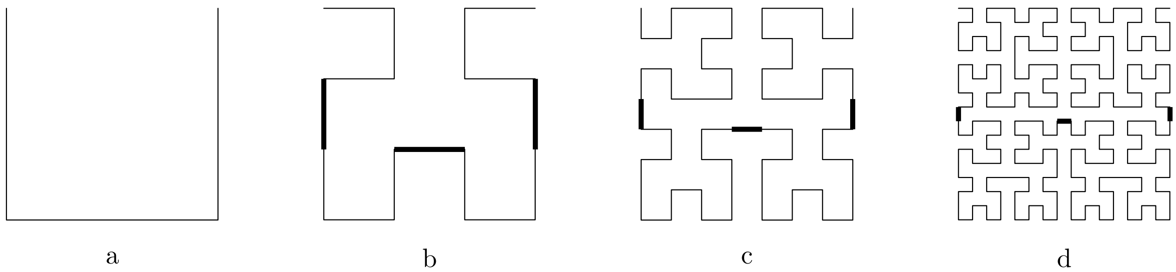


Fig. 1. Hilbert curves, order 1 to 4.

Trying to exterminate the mole, Mr. Hilbert fills the burrow with water (fig. 2). But air inside the burrow prevents water from filling it entirely. In this problem we suppose that air and water are incompressible and cannot leak throw the borders of the burrow. Your task is to find the total area of the burrow, filled with water.

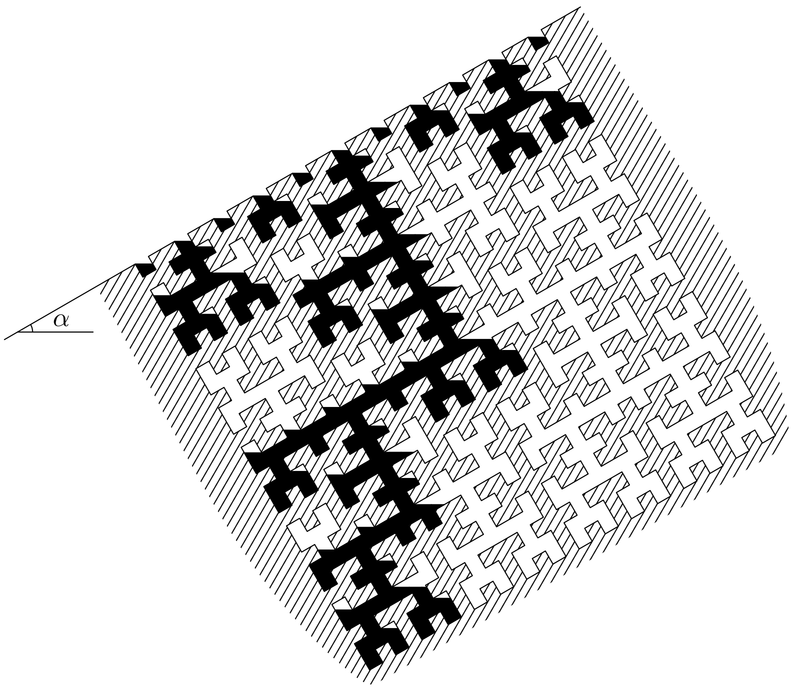


Fig. 2. Burrow, filled with water.

Note that water can flow over the obstacle only when its level is *strictly higher*. See examples on fig. 3 for further clarification.

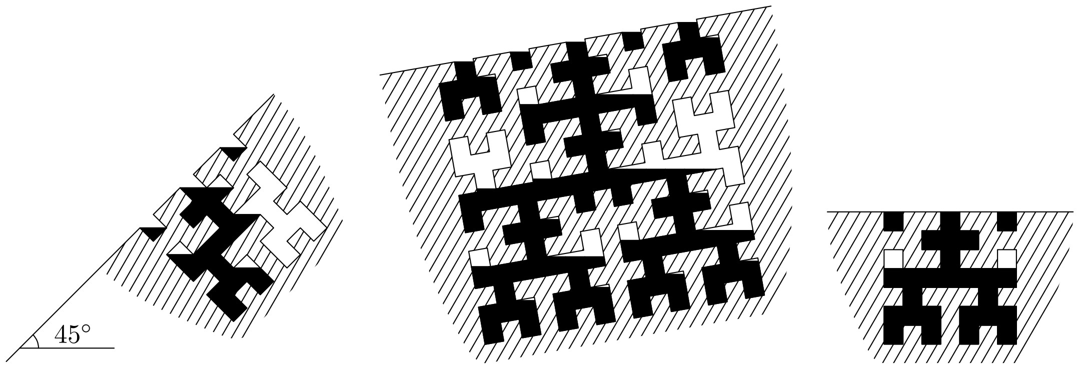


Fig. 3. More examples of filled burrows.

Input

The first line of the input file contains two integer numbers: n and α — order of Hilbert curve and slope angle of surface in degrees ($1 \le n \le 12$, $0 \le \alpha < 90$).

Output

The first line of the output file must contain a single real number — the total area of the burrow, filled with water. The relative error of the answer must not exceed 10^{-7}.

Example

ground.in	ground.out
5 30	190.803847577293
3 45	15.5
4 10	91.573591766702
3 0	26.0

Problem H. Holes

Input file:	`holes.in`
Output file:	`holes.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

You may have seen a mechanic typewriter — such devices were widespread just 15 years ago, before computers replaced them. It is a very simple thing. You strike a key on the typewriter keyboard, the corresponding type bar rises, and the metallic letter molded into the type bar strikes the paper. The art of typewriter typing, however, is more complicated than the art of computer typing. You should strike keys with some force otherwise the prints will not be dark enough. Also you should not overdo it otherwise the paper will be damaged.

Imagine a typewriter with very sharp letters, which cut the paper instead of printing. It is clear that digit 0 being typed on the typewriter makes a nice hole in the paper and you receive a small paper oval as a bonus. The same happens with some other digits: 4, 6, 9 produce one hole, and 8 produces two touching holes. The remaining digits just cut the paper without making holes.

The Jury thinks about some exhibition devoted to the oncoming jubilee of Pascal language. One of the ideas is to make an art installation, consisting of an empty sheet of paper with exactly h $(0 \le h \le 510)$ holes made by typing a non-negative integer number on the cutting typewriter described above. The number must be minimal possible and should not have leading zeroes. Unluckily we are too busy with preparing the ACM quarter- and semifinals, so we need your help and ask you to write a computer program to generate the required number.

Input

A single integer number h — the number of holes.

Output

The integer number which should be typed.

Example

holes.in	holes.out
0	1
1	0
15	48888888
70	88888888888888888888888888888888888

Problem I. Important Wires

Input file:	`important.in`
Output file:	`important.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Nick bought a new motherboard for his computer and it seems that it does not work properly. The motherboard is pretty complicated but it has only few important wires that have binary states: live or dead. Nick wants to know the states of these wires.

Unfortunately, important wires are not directly accessible. But Nick found a maintenance socket. Each output pin of this socket is connected to some of important wires via an integrated circuit. Fortunately, Nick found the circuit layout in the Internet. To specify it, he marked important wires by lowercase letters and socket's output pins by uppercase letters. After that he wrote down Boolean formula for each output pin. In these formulae live wires and pins are represented by *true* and dead wires — by *false*.

Nick used following notation for formulae (operations are listed from the highest priority to the lowest):

- Pin names — letters from ‘`a`’ to ‘`k`’;
- Parentheses — if E is a formula, then (E) is another;
- Negation — $\neg E$ is a formula for any formula E;
- Conjunction — $E_1 \wedge E_2 \wedge \cdots \wedge E_n$;
- Disjunction — $E_1 \vee E_2 \vee \cdots \vee E_n$;
- Implication — $E_1 \Rightarrow E_2 \Rightarrow \cdots \Rightarrow E_n$. Implication is evaluated from right to left: $E_1 \Rightarrow E_2 \Rightarrow E_3$ means $E_1 \Rightarrow (E_2 \Rightarrow E_3)$;
- Equivalence — $E_1 \equiv E_2 \equiv \cdots \equiv E_n$. This expression is by definition computed as follows: $(E_1 \equiv E_2) \wedge (E_2 \equiv E_3) \wedge \cdots \wedge (E_{n-1} \equiv E_n)$.

Nick has lots of various gates at hand, so he can build a new circuit that implements any formula. The variables of this formula are states of maintenance socket's pins. First of all, Nick wants to construct a circuit that takes all maintenance socket's pins as inputs and has a single output wire that is always live. Write a program to help him.

Input

The first line of the input file contains a single integer number n — the number of pins in the maintenance socket ($1 \le n \le 10$). The following n lines contain description of one pin each.

Each pin description consists of a pin name and corresponding formula delimited by ‘`:=`’ token. Pin name is a uppercase English letter. Formula is represented by a string consisting of tokens ‘`a`’..‘`k`’, ‘`(`’, ‘`)`’, ‘`~`’, ‘`&`’, ‘`|`’, ‘`=>`’, and ‘`<=>`’. The last five tokens stand for $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\equiv$ respectively. Tokens can be separated by an arbitrary number of spaces. Each pin description contains 1 000 characters at most.

Output

The first line of the output file must contain “`Yes`” if there exists a circuit which output wire is always live and “`No`” otherwise.

In the former case the following line must contain the formula for the constructed circuit in the same format as in the input file. Remember that the formula must contain each of pin names at least once and it must not contain the wire names. The line must not exceed 1 000 characters.

Example

`important.in`	`important.out`
`3` `A := (a=>c )& (b<=>d)` `C:= a	b` `B := c	d`	`Yes` `C&A => B	~A`

Problem J. Just Too Lucky

Input file:	`just.in`
Output file:	`just.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Since mass transit was invented, people who buy tickets look for lucky ticket numbers. There are many notions of lucky tickets, for example sometimes tickets are considered lucky if the sum of first half of the digits is equal to the sum of the second half, sometimes the product is used instead of the sum, sometimes permutation of digits is allowed, etc.

In St Andrewburg integer numbers from 1 to n are used as ticket numbers. Bill considers a ticket lucky if its number is divisible by the sum of its digits. Help Bill to find the number of lucky tickets.

Input

The first line of the input file contains n ($1 \le n \le 10^{12}$).

Output

Output one number — the number of lucky tickets.

Example

just.in	just.out
100	33

Problem K. Key to Success

Input file:	`key.in`
Output file:	`key.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

The organizers of the TV Show "Key to Success" are preparing a set of prizes for the winner of the game. If the score of the winner is X, she must choose prizes with a total value of exactly X dollars.

The organizers have a couple of spare prizes from the previous competitions that have values $a_1, a_2, \ldots, a_n$ dollars, respectively. Unfortunately they don't know what the score of the winner will be. So the organizers decided to buy m more prizes in order to maximize the minimal integer score that the winner of the show wouldn't be able to collect prizes for.

For example, if they already have prizes for 2, 3 and 9 dollars, and they want to buy 2 prizes, they should buy prizes for 1 and 7 dollars. Then the winner of the show would be able to collect prizes for any score from 1 to 22.

Input

The first line of the input file contains two integer numbers: n and m — the number of prizes the organizers have and the number of prizes they are ready to buy ($0 \le n \le 30$, $1 \le m \le 30$). The second line contains n integer numbers ranging from 1 to 10^9 — the values of the prizes organizers have.

Output

Output m integer numbers — the values of the prizes the show organizers should buy. Output numbers in non-decreasing order. If there are several optimal solutions, output any one.

Example

key.in	key.out
3 2 2 3 9	1 7