Problem A. Casino

Input file:	blackjack.in
Output file:	blackjack.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Eugene owns a small casino, and there is only game in that casino. This game uses an infinite deck of cards: for each card type, the probability of such card being the next one in the deck is known, and the probabilities stay constant. Each card type also has an integer value associated with it.

The game is played in the following way. First, the dealer takes cards one by one. As soon as the sum of values of taken cards becomes greater than integer S, he gives the last taken card to the player and stops taking new cards.

After that, the player takes cards one by one, and he could not drop the card if he already took it. The player does not know the values of the dealer's cards, but knows how many cards the dealer took. Also, he knows the algorithm which the dealer uses.

The goal for the player is to have cards with the sum of values not greater than S, but greater than the sum of values of dealer's cards. In such case, he will win and receive one dollar. If the sum of the player's cards and the sum of the dealer's cards are equal, the game will end in a draw, and the player will receive nothing. in all other cases, the player will lose one dollar.

In these latter days, things are going bad in the casino, so Eugene decided to study the game more thoroughly. He wants to find the expected value of player's gain if he plays optimally.

Input

The first line contains two integers n and S ($2 \le n \le 50$, $2 \le S \le 400$), the amount of different types of cards and the limit for the sum of card values.

The second line contains n different integers a_1, a_2, ..., a_n ($2 \le a_i \le 400$), the values of the cards.

The third line contains n real numbers p_1, p_2, ..., p_n ($0.0001 \le p_i \le 1.0000$), the probabilities of appearance of card types with values a_1, a_2, ..., a_n respectively. It is guaranteed that each p_i has at most four digits after the decimal point, and the sum of all p_i is equal to 1.

Output

Output a single real number: the expected value of player's gain. The absolute or relative error should not exceed 10^{-6}.

Examples

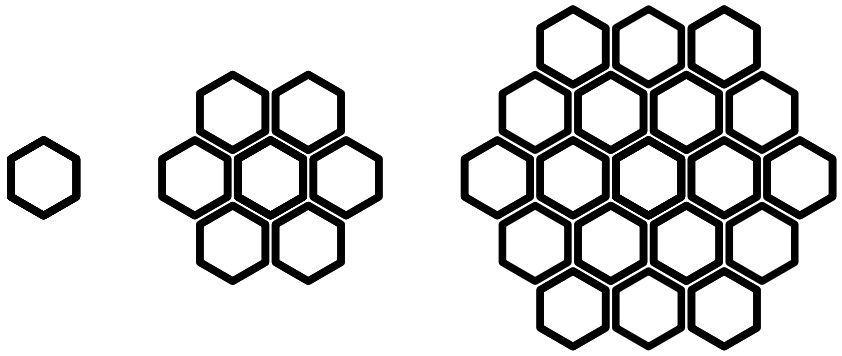
blackjack.in	blackjack.out
2 4 2 4 0.25 0.75	0.0351562500
2 6 2 4 0.25 0.75	-0.2614746094

Problem B. Caps and Cakes

Input file:	caps-and-cakes.in
Output file:	caps-and-cakes.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

In the kindergarten "Learn by Playing", children go for a walk in the park every day. There, they stop to eat a light meal: each kid gets one pouch of delicious fruit puree. When the puree is eaten, children throw empty pouches in a trash can but keep the hexagonal pouch caps. The caps are special: they have the form of identical regular hexagons with connectors at the sides, so that they can be assembled together as construction toys.

Six friends from the kindergarten met today at a celebration, and each of them brought m caps to play cake-building. A cake is a planar figure which consists of caps and resembles a regular hexagon. A cake is characterized by its size which is a positive integer. A cake of size 1 is just a lonely cap. A cake of size $s > 1$ is assembled as follows: take a cake of size $(s - 1)$ and add a new layer of caps along its border. Cakes of sizes 1, 2 and 3 are shown on the picture.



The six friends want to know how they can make the least possible number of cakes using all the caps they got. Help them find that out.

Input

The first line of input contains an integer n: the number of caps each of the friends brought to the party ($1 \le n \le 10^{18}$).

Output

On the first line, print an integer k: the number of cakes. This number must be as small as possible.

On the second line, print k integers separated by spaces: the sizes of the cakes in arbitrary order. If there are several solutions with the same k, print any one of them.

Example

caps-and-cakes.in	caps-and-cakes.out
3	6 1 2 1 2 1 1

Explanation

In the example, the friends have 18 caps in total. They can be arranged to form six cakes: for example, four cakes of size 1 and two cakes of size 2. Using all caps, the children cannot form less than six cakes.

Problem C. Circles

Input file:	circles.in
Output file:	circles.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

There are n random circles on a plane. Construct a circle that intersects as many of the given circles as possible.

Input

The first line of input contains one integer n: the number of circles ($1 \le n \le 80$).

Each of the next n lines contains three integers x_i, y_i, r_i: coordinates of the center of i-th circle and its radius.

It is guaranteed that all tests except the example satisfy $n = 80$. In addition, they are all generated as follows. First, three integers a_0, c, d are chosen randomly ($1 \le a_0, c, d \le 10^9 + 7$). Then sequence a_i of length $3 \cdot n$ is generated following the rule $a_i = (a_{i-1} \cdot c + d) \bmod (10^9 + 7) + 1$. Finally, coordinates of center and radii of circles are defined as $x_i = a_{3 \cdot i - 2}$, $y_i = a_{3 \cdot i - 1}$, $r_i = a_{3 \cdot i}$.

Output

On the first line, print one integer: the number k of given circles intersecting with the circle you constructed.

On the second line print three real numbers x, y, r: coordinates of the center and radius of the circle you constructed.

Your answer will be considered correct if k is maximum possible and the ring with center (x, y) and radii $r - 10^{-9}, r + 10^{-9}$ of inner and outer circles respectively intersects exactly k of the given circles.

We guarantee that any ring of breadth 10^{-6} will intersect no more given circles than the optimal answer.

Example

circles.in	circles.out
3 1 1 1 3 1 1 1 3 1	3 2 2 2.41421356237

Problem D. C-plus-minus

Input file:	`cpm.in`
Output file:	`cpm.out`
Time limit:	5 seconds
Memory limit:	512 mebibytes

Four pages long statement?!

Inexperienced participant

It's an implementation problem, sit down and code

Experienced participant who did not read the statement

How do I code that?

Experienced participant who did read the statement

Modern IDEs (integrated development environments) do a lot of stuff. Say, instantly suggest functions and variables based on types, show relevant documentation snippets, detect mistakes, rearrange code according to the style guide, consume all available RAM... In this problem you have to implement small piece of IDE for the $C\pm$ language.

In the beginning you have $C\pm$ program's source code called S_0. Your program should reformat the code according to rules below with adding and removing of spaces and newlines, yielding program F_0, which you have to print.

Afterwards you should perform m queries which change the source code. Let's denote the source code after performing i-th query ($1 \le i \le m$) as S_i. For each S_i we can yield a reformatted source code F_i. For each modification query you should print one integer: $L(F_i)$, where $L(x)$ stands for number of lines in source code x (see below for precise definition).

There are three types of queries:

- "`add` p `@`t`@`" query inserts string t into source S_{i-1}, starting at character number p (here $|t|$ stands for the length of t):

$$S_i = S_{i-1,1}S_{i-1,2}\ldots S_{i-1,p-1}t_1t_2\ldots t_{|t|-1}t_{|t|}S_{i-1,p}S_{i-1,p+1}\ldots S_{i-1,|S_{i-1}|}$$

- "`del` p l" query removes a string of length l from S_{i-1}, starting at character number p:

$$S_i = S_{i-1,1}S_{i-1,2}\ldots S_{i-1,p-1}S_{i-1,p+l}S_{i-1,p+l+1}\ldots S_{i-1,|S_{i-1}|}$$

- "`get` r s k" query does not affect the source ($S_i = S_{i-1}$), but you should print a substring of source F_i. The desired substring is characterized with a line number r, number of the first desired character in that line s and length of the substring k.

  If F_i has less than r lines, we consider an empty string to be the answer for the `get` query. If r-th line in F_i does not have some characters with numbers $s, s+1, \ldots, s+k-1$, we consider these missing characters to be equal to "`#`" (octothorpe, ASCII code is 35).

It's not guaranteed that $S_0, S_1, \ldots, S_n$ are correct $C\pm$ programs. However, formatting rules described below can be applied to arbitrary sources.

Before defining formatting rules we'll introduce several supporting definitions:

- *Identifier* is a maximal by inclusion sequence of Latin letters, digits, square brackets "`[`" and "`]`" (ASCII codes 91 and 93), dots "`.`" (ASCII code 46) and underscores "`_`" (ASCII code 95). Some correct $C\pm$ identifiers: `hi`, `hello_world`, `bob[er9]5`, `12c.hairs`, `12_3`.
- *Binary operator* is one of the following characters:

Character	ASCII code
+	43
-	45
*	42
/	47
<	60
>	62
=	61
&	38
|	124

- *$C\pm$ allows the following characters*: parts of identifiers, binary operators, round brackets ("`(`", ASCII code 40 and "`)`", ASCII code 41), curly brackets ("`{`", ASCII code 123 and "`}`", ASCII code 125), comma "`,`" (ASCII code 44), semicolon "`;`" (ASCII code 59), space (ASCII code 32) and new line character (ASCII code 10, typically denoted by "\n" in modern programming languages). It's guaranteed that all sources $S_0, \ldots, S_n$ contain allowed characters only.
- *Lexem* is either identifier or any other allowed character, except for space and new line character. For example, source `hello+ world` contains three lexems: `hello`, `+` and `world`.
- *R-balance of brackets* (round or curly) in a some string S is the result of the following pseudocode (it counts balance for round brackets, code for curly brackets is similar):

```
B <- 0
for each C in S do {
  if C is "(" then B <- B + 1
  if C is ")" then B <- B - 1
  if B < 0 then B <- 0
  if B > R then B <- R
}
return B
```

  Not strictly speaking, this code calculates difference between number of opening and closing brackets of a specific type in S, but if this balance "gets beyond" $[0, R]$ segment, then it's "truncated" (R is either positive integer or ∞). For example, string "`{{{){{}})}}(`" has ∞-balance of both round and curly brackets equal to 1, and string "`(((((`" has 4-balance of round brackets equal to 3.
- Position p is said to be *inside round brackets* if a substring t which ends at p and starts right after last curly bracket before p (or in the beginning of the source if there are no curly brackets before p), has 4-*balance of round brackets* strictly greater than zero. For example, if $t =$`(( ((( )))`, then p is *inside round brackets* (as 4-balance is 1), and if $t =$`(( ((( ))) )`, it is not (as 4-balance is 0).

Formatting is performed as follows:

1. All spaces and new line characters are removed from the source. Neighboring lexems are **not** considered as a single lexem after this step.

2. New line characters are added in the following places (unless empty lines appear):
   - After opening curly bracket "`{`".
   - Before closing curly bracket "`}`".
   - After closing curly bracket "`}`", unless it's immediately followed by either "`else`" identifier or semicolon.
   - After semicolon "`;`", unless it is located *inside round brackets*.
   - In the very end of the source.

   Line in the resulting program is defined as a sequence of characters ending with new line character. So, $L(x)$ is defined as the number of new line characters.

3. Spaces are added in each line between neighboring lexems according to the table below (plus on the intersection of row A and column B means adding of space between A and B):

	Identifier	Binary operator	,	{	(	)	;
Identifier	+	+	-	+	-	-	-
Binary operator	+	-	+	+	+	+	-
,	+	+	-	+	+	-	+
}	+	+	-	+	+	-	-
(	-	+	-	-	-	-	-
)	+	+	-	+	+	-	-
;	+	+	-	+	+	-	+

   Note that opening curly bracket `{` is always the last character in the line. Similarly, `}` is always the first character in the line.

4. If one of the identifiers `if`, `for`, `while`, `do` is immediately followed by opening round bracket, an additional space is added between them.

5. Afterwards an *indent* is calculated for each line. *Indent* is equal to ∞-balance of curly brackets in all previous lines.

6. Spaces are prepended to each line in the amount of doubled *indent* of the corresponding line.

Input

First lines of the input file contain original source code S_0 in the format `@`S_0`@` (at sign has ASCII code of 64). S_0 contains *allowed* characters only ($1 \le |S_0| \le 10^5$). Note that S_0 is not allowed to contain at sign, although it can contain spaces and/or new line characters. The second at sign is immediately followed by new line character.

The next line contains a single integer m, the number of queries ($0 \le m \le 10^5$). Next lines contain queries separated by at least one new line character, and i-th query is formatted as follows:

- "`add` p `@`t`@`": add non-empty string t into current source, starting at position p ($1 \le p \le |S_{i-1}| + 1$). It's guaranteed that t contains *allowed* characters only. In particular, t may contain spaces and/or new line characters. The second at sign is immediately followed by new line character. It's guaranteed that sum of all $|t|$ does not exceed 10^5.
- "`del` p k": erase a substring of length k from current source, starting at position p ($1 \le p \le p + k - 1 \le |S_{i-1}|$).
- "`get` r s k": print k characters of line number r of current source, starting at position s ($1 \le r, s, k \le 10^5$). It's guaranteed that sum of all k does not exceed 10^5.

Please see example for better understanding of input format.

Output

At the beginning of the output file, print one integer $L(F_0)$: number of lines in the formatted source F_0. Afterwards print another space, at sign, $\min(10^5, |F_0|)$ characters of F_0, followed by another at sign and new line character.

Next, print the answer of i-th query on a separate line:

- For `add` and `del` queries: print number of lines in the source after corresponding change ($L(F_i)$).
- For `get` queries: print string `@`t`@` where t is the requested substring of length l of the formatted source F_i. If there is no line of number r in F_i (that is, $r > L(F_i)$), then you should print empty string instead of t. If $r \le L(F_i)$, but some of requested characters do not exist in F_i, you should print "`#`" character (ASCII code 35) instead of them. We consider new line character to be not included in t.

Please see example for better understanding of output format. Follow the format as close as possible: your answer must be identical to the required answer.

Example

cpm.in	cpm.out

`@if (a) { foo } else { bar; };`
`for (int a = 0; a< 5; a+=1)`
`{ some strange(command, wow); }}`
`else int foo bar {`
`  {(hi; strange; world)}`
 	
`   (hi; strange; world() {there;}`
`}@`
`14`
`get 3 2 10`
`get 100 1 1`
`add 32 @NEW`
`LINE@`
`get 6 1 30`
`del 32 8`
`add 18 @se if @`
`get 2 1 10`
`get 3 1 10`
`get 4 1 10`
`del 20 1`
`get 2 1 10`
`get 3 1 10`
`get 4 1 10`
`get 5 1 10`
	`16 @if (a) {`
	`  foo`
	`} else {`
	`  bar;`
	`};`
	`for (int a = 0; a < 5; a += 1) {`
	`  some strange(command, wow);`
	`}`
	`} else int foo bar {`
	`  {`
	`    (hi; strange; world)`
	`  }`
	`  (hi; strange; world() {`
	`    there;`
	`  }`
	`}`
	`@`
	`@ else {###@`
	`@@`
	`16`
	`@fNEW LINEor(int a = 0; a < 5; @`
	`16`
	`16`
	`@  foo#####@`
	`@} else if @`
	`@  bar;####@`
	`17`
	`@  foo#####@`
	`@}##########@`
	`@elseif se @`
	`@  bar;####@`

Problem E. Maharajah

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

Maharajah is a chess piece which can move either as a queen or as a knight. It is used in some chess-like games. Its mobility makes maharajah an extremely powerful piece. For example, in the game called "Maharajah and the Sepoys", White maharajah fights against a complete army of Black chess pieces (called sepoys). To win the game, the maharajah must checkmate Black's king.

Vasya heard that there exists a strategy which allows Black to win the game regardless of White's moves. He thinks that's not fair and now tries to remove some black pieces and analyze the games he obtains. At first, he tried the most simple case: maharajah must checkmate a lone Black King.

Vasya's friend Petya said that this game is even more unfair. Petya stated that he can checkmate a lone Black King with maharajah even without any information about King's moves and position.

Your task is to repeat Petya's achievement. Your program will play as White maharajah against the evil interactor. It means that the interactor is able to choose an arbitrary valid initial position of the Black King (that is, any position different from maharajah's one and such that the King is not in check) and its valid sequence of moves not only before the start of the game, but at any moment of the game process later. If at least one sequence exists in which the maharajah is taken by the King, or the King is stalemated, your solution is considered wrong. Also, you have only 50 moves to checkmate the invisible King, otherwise you lose too (remember that in classic chess, 50 moves without capturing a piece or moving a pawn is considered a draw). If your solution makes more than 50 moves, it is considered wrong. Maharajah moves first. Good luck!

Interaction Protocol

The game is started by specifying the initial position of maharajah. You should read the position from a single line of the standard input stream. After that, you need to make your moves. Each move is made by writing to the standard output stream a single line containing the position you selected for maharajah. Interactor responds with a single line with `-1` if you lost or made an invalid move, `0` if the game continues, and `1` if you win. You program must immediately terminate gracefully when it has received anything different from `0`.

The position (either in the input or in the output) is always specified by a single lowercase letter (`a`...`h`) followed by a single digit (`1`...`8`).

To prevent output buffering, flush the output buffer after each move: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

Example

standard input	standard output
a1 0 -1	 b1 d2

Explanation

Note that, in this example, the solution loses. Your task is to write a winning one!

Problem F. Matryoshka Dolls

Input file:	matryoshka-dolls.in
Output file:	matryoshka-dolls.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

A matryoshka is a set of wooden dolls of different sizes. All dolls except the smallest one are hollow and can be separated into two parts: upper and lower. When the matryoshka is assembled, the smallest doll lies inside the second smallest, which lies inside the third one, and so on.

Little Ignat had a few identical matryoshkas each of which consisted of n dolls. Unfortunately, at the moment, some dolls may have gone missing, and the remaining ones are scattered across the floor.

Now Ignat wants to assemble the matryoshkas so that they don't occupy too much of the floor. As some of the dolls may be missing, he decided to follow more liberal rules: a matryoshka is a doll which either is empty or contains one doll of any lesser size which, in turn, can either be empty or contain one doll of any size even lesser, and so on.

It is known how many dolls of each size are on the floor. What is the minimum possible number of matryoshkas Ignat can assemble according to these rules?

Input

The first line of input contains an integer n: the number of sizes of dolls ($1 \le n \le 100$). The second line contains n space-separated integers a_1, a_2, ..., a_n: the number of dolls of each size ($1 \le a_i \le 100$). The sizes are listed from largest to smallest: for example, a_1 is the number of largest dolls, and a_n is the number of smallest dolls.

Output

On the first line, print an integer k: the minimum possible number of matryoshkas.

Example

matryoshka-dolls.in	matryoshka-dolls.out	Illustration
3 3 2 5	5	

Explanation

The picture to the right of the example shows one possible configuration of the dolls where the number of matryoshkas is 5.

Problem G. Non-Extremal Value

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

Consider four pairwise distinct numbers: `a`, `b`, `c` and `d`. The numbers are fixed but kept secret. One may ask at most three questions of the form "is it true that one of these numbers is less than another?" and get the answer for each question immediately. After that, one must name a number which is neither the smallest nor the greatest of these four numbers.

Write a program which will solve this problem for any possible `a`, `b`, `c` and `d`.

Interaction Protocol

Your program must communicate with the jury program via standard input and standard output streams. The program may ask from zero to three questions, and after that, it must provide an answer.

The questions are accepted in the form "is it true that $x < y$?". To ask such a question, print the line "`Is` x `<` y`?`" to the standard output stream. The variables x and y must be replaced by different letters from letters "`a`", "`b`", "`c`" and "`d`". The line must end with a newline.

To prevent output buffering, flush the output buffer after each request: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

The answer to the question in the form "`Yes.`" or "`No.`" on a separate line is provided at your program's standard input stream.

If your program is ready to give an answer, do it by printing the line "`Value` z `is non-extremal.`" to the standard output stream. The variable z must be replaced by letter "`a`", "`b`", "`c`" or "`d`". This line must also end with a newline. After that, your program must immediately terminate gracefully.

Example

Please note that the participant's **output** is displayed to the **left**, and the following **input** to the **right**.

participant's actions	jury's answers
`Is a < b?`	`Yes.`
`Is b < c?`	`Yes.`
`Is c < d?`	`No.`
`Value b is non-extremal.`	

Problem H. Parallelograms

Input file:	parallelograms.in
Output file:	parallelograms.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Little Igor likes to build geometric shapes. Igor has n segments. Igor is still a little boy so he does not know that segments can be rotated. He knows only how to translate them.

Today Igor learned a new shape: the parallelogram. Of course, now he wants to build as many of these wonderful shapes as he can. He will use each segment in at most one parallelogram. Each side of each parallelogram will consist of just one segment. Also, Igor is interested only in shapes with positive area. Some segments may be left unused.

Help Igor!

Input

The first line of input contains an integer n: the number of segments Igor has ($1 \le n \le 10^5$). Each of the next n lines contain four integers $x_{i,1}$, $y_{i,1}$, $x_{i,2}$, $y_{i,2}$: coordinates of ends of the i-th segment ($-10^9 \le x_{i,1}, y_{i,1}, x_{i,2}, y_{i,2} \le 10^9$).

Output

Print one integer k: the maximum possible number of parallelograms Igor can build with his segments.

Example

parallelograms.in	parallelograms.out
5 0 0 1 2 2 2 5 3 5 7 4 5 0 1 3 2 3 5 3 5	1

Problem I. Posters

Input file:	posters.in
Output file:	posters.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

The upcoming World Congress will gather together n delegates. A large hall contains a round table where exactly n people can take seats. However, there is a problem: there is a poster above each of the n seats at the table. The posters have different contents, and the delegates have different preferences, so in order to avoid awkward situations, certain delegates can not seat under certain posters.

The Chief Administrator of the Congress got the full list of restrictions. In this list, the delegates are numbered by integers from 1 to n. The seats at the table are also numbered by integers from 1 to n. Each restriction is a pair (d, s) which means that the delegate numbered as d can not take seat numbered as s.

Help the Chief Administrator by listing every possible seating plan which satisfies all restrictions. The history of previous congresses shows with absolute certainty that the number of possible seating plans is no more than 1000.

Input

The first line contains two integers n and m separated by a space: the number of delegates and the number of restrictions ($1 \le n \le 100$, $0 \le m \le 10\,000$). Each of the next m lines contains two integers d_i and s_i which mean that delegate d_i can not take seat s_i ($1 \le d_i, s_i \le n$). It is guaranteed that all pairs (d_i, s_i) are distinct.

Output

On the first line, print a single integer k: the number of possible seating plans. Each of the following k lines must contain one possible seating plan: n integers a_1, a_2, …, a_n separated by spaces. The number a_i denotes the seat which should be taken by delegate number i. Each possible seating plan must be printed exactly once. Print the seating plans in any order.

It is guaranteed that in the right answer $0 \le k \le 1000$.

Example

posters.in	posters.out
4 6 1 2 1 3 2 3 2 4 4 4 2 1	3 4 2 1 3 1 2 4 3 4 2 3 1

Explanation

In the example, the second delegate can take only seat number 2. If we choose which of the remaining delegates takes seat number 1, it turns out that the delegates which occupy the remaining two seats can be uniquely determined.

Problem J. Quaternary Squares

Input file:	quaternary.in
Output file:	quaternary.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

It is well-known that one can write any number in quaternary numeral system using only digits 0, 1, 2 and 3.

Vasya once wrote an integer in quaternary numeral system and then decided to calculate its square root on his computer. Occasionally, in his code he had parsed that number as if it was in decimal representation but did not notice he had done something wrong, because the square root was also an integer consisting in its decimal representation only of digits 0, 1, 2 and 3!

Later, when he has realized his mistake, he has called such numbers *quaternary squares*. More formally, a *quaternary square* is a decimal integer without leading zeros consisting only of digits 0, 1, 2 and 3 which is a perfect square and its square root is also an integer consisting only of digits 0, 1, 2 and 3 in its decimal representation.

Your task is quite simple. Let us sort all quaternary squares of length n in ascending order. Find the k-th of them (numeration is 1-based).

Input

The input consists of one or more test cases.

The only line of each test case contains two integers n and k ($1 \le n \le 40$, $k \ge 1$). It is guaranteed that k-th quaternary square exists. The sum of all values of n in the input does not exceed 239.

The last line contains two zeros.

Output

Output one line for each test case: k-th quaternary square of length n.

Example

quaternary.in	quaternary.out
3 1 3 2 0 0	100 121

Problem K. Transmission by Parts

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

Sasha is a scout. She wants to transmit an important message to the headquarters. The message is concealed in a string consisting of lowercase English letters. Today, the cipher is as follows: the message is the most frequent substring of the transmitted string. In case there are several most frequent strings, the message is the lexicographically greatest of them.

Sasha knows that transmitting messages is a dangerous practice. Therefore, she is going to split the string into k parts and transmit them one after another: each transmission will take place when the conditions are favorable. Still, a variety of things might happen between two consecutive transmissions, so after transmitting any part, transmitting the rest might become impossible. Therefore, Sasha wants to check that in this case, the message will not turn out to be something catastrophic.

Given k string parts in the order they have in the string, print k strings: what will be deciphered message if the transmission stops after the first, second, ..., k-th part.

String s is lexicographically greater than string t if either t is a proper prefix of s or there is a position where s and t do not match, and in the leftmost of such positions i, the relation is $s_i > t_i$.

Interaction Protocol

Your program must communicate with the jury program via standard input and standard output streams.

At first, your program gets the number k on a separate line ($1 \le k \le 10\,000$). After that, your program gets k string parts. Each part is given on a separate line, consists of lowercase English letters and has length from one to one million letters inclusive. Moreover, it is guaranteed that the total length of the string is from one to one million letters inclusive. Each part starting from the second one is given as soon as your program provides an answer for the previous part.

After getting each of the k string parts, print one line: what will be the deciphered message if the transmission ends now. The line must end with a newline.

To prevent output buffering, flush the output buffer after each request: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

After printing k answers, your program must immediately terminate gracefully.

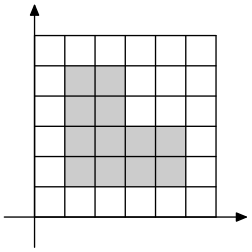
Example

standard input	standard output
4 a c ba cb	*(reading...)* a c a cb

Problem L. UFO Driver

Input file:	ufo.in
Output file:	ufo.out
Time limit:	5 seconds
Memory limit:	256 mebibytes

UFO driver Arseny wants to land his UFO onto a plaform which is a square of size $L \times L$ meters. Arseny's UFO has the form of a square of size $2A \times 2A$ meters without one quarter. Its relative position to the platform is shown on the picture. Here, $L = 6$ and $A = 2$.



The landing module established a coordinate system on the platform. Its axes are going along the sides of platform, one corner has coordinates $(0, 0)$, and the opposite corner is at (L, L).

For technical reasons, UFO can land only so that all its corners are in points with integer coordinates. Additionally, UFO can't be rotated.

There are several cameras on the platform. Each of them sees a rectangle inside the platform. Each camera has picture quality characterized by an integer q_i. The *visibility* of Arseny's landing is the maximal quality of a picture where a part of his UFO with non-zero area can be seen. The visibility of landing where no cameras can see the UFO is zero. The picture below shows the first sample test. For each cell, the visibility of landing on it is displayed in the cell. The only way to land with visibility 2 is shown.

Arseny wants his UFO to stay unidentified, so he needs to land at some position where visibility is minimum possible. Help him find such position. Note that the UFO should land fully inside the platform.

Input

Input data consists of one or more test cases given one after another.

The first line of a test case description contains two integers A and L ($1 \le A$, $2 \cdot A \le L \le 10^9$) which are the sizes of UFO and the platform.

The second line contains an integer n ($1 \le n \le 50\,000$), the number of cameras. The next n lines describe the cameras, one per line. Each camera is described by five integers $x_{i,1}$, $y_{i,1}$, $x_{i,2}$, $y_{i,2}$ and q_i ($0 \le x_{i,1} < x_{i,2} \le L$, $0 \le y_{i,1} < y_{i,2} \le L$, $1 \le q_i \le 10^9$): the coordinates of two opposite corners and picture quality.

The sum of n for all test cases in one test is no more than 50 000.

Output

For each test case, print two lines. On the first line, print one integer: the minimum possible visibility of landing. On the next line, print two integers: the coordinates of the UFO corner closest to the origin after landing. If there are several possible answers, print any one of them.

Example

ufo.in	ufo.out	Notes
1 4 4 0 0 4 1 10 0 0 1 4 10 1 1 3 3 2 2 2 3 3 5 1 2 1 1 1 2 2 10	2 1 1 0 0 0