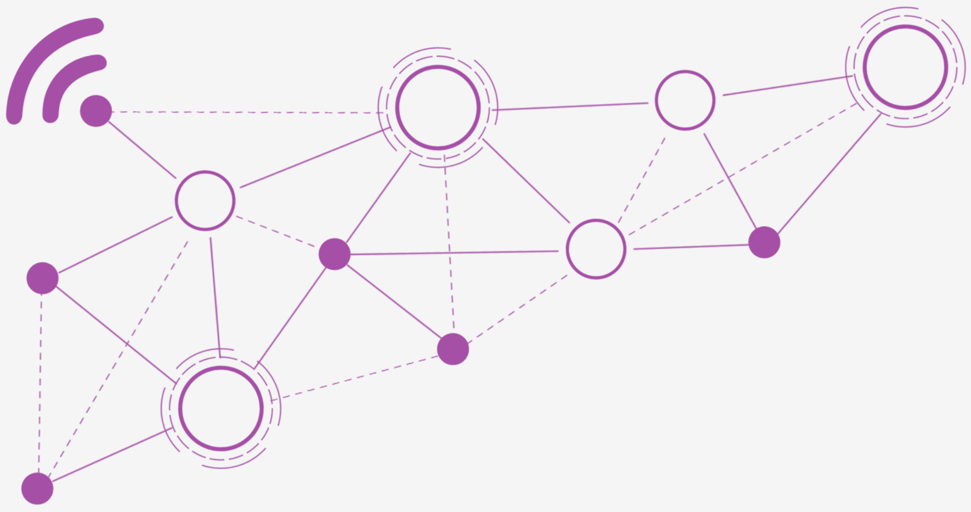


NOI.PH 2022

National Olympiad in Informatics

Final Round

Contents

Notes

- Many problems have large input file sizes, so use fast I/O. For example:
  - In C/C++, use `scanf` and `printf`.
  - In Python, `import sys` and then use `sys.stdin.readline()`
- Good luck and enjoy the problems!

Important! Read the following:

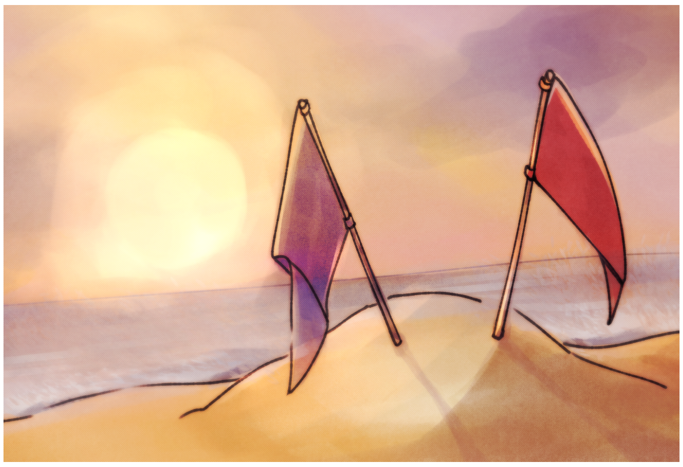
- Your solution will be checked by running it against one or more (usually several) hidden test cases. You will not have access to these cases, but a correct solution is expected to handle them correctly.
- The output checker is **strict**. Follow these guidelines strictly:
  - It is **space sensitive**. Do not output extra leading or trailing spaces. Do not output extra blank lines unless explicitly stated.
  - It is **case sensitive**. So, for example, if the problem asks for the output in lowercase, follow it.
  - Do not print any tabs. (No tabs will be required in the output.)
  - Do not output anything else aside from what's asked for in the Output section. So, do not print things like "Please enter t".

  Not following the output format strictly and exactly will likely result in the verdict "*Output isn't correct*".
- Do not read from, or write to, a file. You must read from the standard input and write to the standard output.
- Only include one file when submitting: the source code (.cpp, .py, etc.) and nothing else.
- Only use letters, digits and underscores in your filename. Do not use spaces or other special symbols.
- Many problems have large input file sizes, so use fast I/O. For example:
  - In C/C++, use `scanf` and `printf`.
  - In Python, `import sys` and then use `sys.stdin.readline()`

  We recommend learning and using these functions during the Practice Session.
- Good luck and enjoy the contest!

Problem A
Harry Palayok and the Sedimentary Stone

Time Limit: 2 seconds



Harry Palayok has been living on the skirts of Manila Bay for the past 6 years. Although he lives a simple life, his memories of grade school have taught him the power of magic. He particularly believes that good mental health and well-being are some of the most powerful forms of magic there is.

He therefore wants to uplift the spirits of everyone around him—and what better way to do this than by beautifying his surroundings? Having received a shipment of dolomite from an anonymous donor, he begins his work.

The area of Manila Bay he plans to work on can be described as an h by w grid. At the end of the reclamation process, each cell will either be empty, or be filled with dolomite.

Because Harry wants to share his joy with everyone, he wants as many people as possible to visit the bay. He defines a *viewing site* to be a cell such that:

- The cell does not contain dolomite. Obviously, they'd need some place to stand on!
- None of the 4 cells *orthogonally* adjacent (i.e. up, down, left, right) have dolomite in them. This lets visitors walk in and out of the site freely.
- Exactly **3** of the 4 *diagonally* adjacent cells have dolomite in them. This lets visitors experience its healing effects, without being overloaded too much.

Note that cells that are on a border or corner of the grid cannot be viewing sites.

Since he wishes to improve the mental health of as many citizens as possible, Harry will lay dolomite on the Bay such that the number of viewing sites is maximized. Can you provide a layout of which cells to fill, to help him fulfill this?

Input Format

The only line of input contains two space-separated integers h and w: the height and width of the grid, respectively.

Output Format

Output h lines, each containing w characters, representing the optimal layout. The jth character of the ith line must be "#" if the cell at row i and column j contains dolomite, and "." otherwise.

If there are multiple optimal layouts, you may output any of them.

Constraints and Subtasks

For all subtasks
$1 \le w \le 10^5$ $1 \le h \le 4$

Subtask	Points	Constraints
1	**25**	$1 \le h \le 2$
2	**20**	$1 \le w \le 5$
3	**25**	$1 \le h \le 3$
4	**30**	No further constraints.

Sample I/O

Input 1

```
3 3
```

Output 1

```
#.#
...
#..
```

Explanation

The cell at row 2 and column 2 is a viewing site, since all of its orthogonal neighbors have no dolomite, and only its bottom-right neighbor does not have dolomite. Since it can be shown that 1 is the maximum possible number of viewing sites, this output is considered correct.

Problem B
Bienvenido!

Time Limit: 2 seconds

Bienvenido, Fr Bueno!

All his life, Fr Bueno has always been very passionate about mathematics! He enjoys being directly involved in classroom work, even moreso than being bogged down in a "higher" administrative position. Currently, he is handling an advanced mathematics class — Topological Logics on Metalogical Manifolds. It's so advanced, that the class only has two students! And it's so advanced that... those two students aren't doing very well...

Still, Fr Bueno notices that they are working very hard and definitely putting a lot of effort into the class, which he doesn't want to go unacknowledged. But also, it's not like he can just give them free bonus points out of nowhere. What to do?

Fr Bueno decides that his next long test will have an interesting grading scheme. The long test is worth a total of HPS points. There are n questions, labeled 1 to n, whose point values are $p_1, p_2, p_3, \dots, p_n$. There are no partial points—so for some question i, either the student gets it correct and is awarded p_i points, or gets it incorrect and is awarded 0 points. A student's total score for the whole long test is the sum of the points that they were awarded for their correct answers. A student must get at least D points in order to pass.

The interesting thing is that **the point values are not decided behorehand**. Fr Bueno first looks at his two students' answers, and only after seeing their correct/incorrect answers does he decide how many points to allocate to each question, subject to the following constraints:

- Each p_i must be a positive integer.
- $p_1 + p_2 + p_3 + \cdots + p_n = HPS$.

His class only has two students, so Fr Bueno figures that he can work some magic on the point allocations so that *both* of them pass. Given the values of HPS and D, as well as the two students' checked long tests, **count** the number of different possible point allocations such that both students pass. This number can be quite large, so just compute it modulo 1503194023.

Two point allocations are considered different if there exists a question that is worth a different number of points between the two ways.

Input Format

The first line of input contains three integers: n representing the number of questions, HPS representing the total number of points, and D representing the passing score.

The second line of input contains n integers, the first student's checked long test. The ith integer in this line is 1 if the first student got a correct answer to the ith question in the long test, and 0 if the first student got an incorrect answer to the ith question.

The second line of input contains n integers, the second student's checked long test. The ith integer in this line is 1 if the second student got a correct answer to the ith question in the long test, and 0 if the second student got an incorrect answer to the ith question.

Output Format

Output a line containing a single integer, the number of ways to allocate points such that both students pass, modulo 1503194023.

Constraints and Subtasks

For all subtasks
$1 \leq HPS \leq 4000$ $1 \leq n \leq HPS$ $1 \leq D \leq HPS$

Subtask	Points	Constraints
1	**8**	$n \leq 12$
2	**20**	$n \leq 50$
3	**8**	$n \leq 500$
4	**8**	Both students got all questions correct.
5	**8**	Each question was answered correctly by at most one student.
6	**48**	No further constraints.

Sample I/O

Input 1

```
3 6 2
1 0 0
0 0 1
```

Output 1

```
3
```

Input 2

```
3 6 2
1 1 1
1 1 1
```

Output 2

```
10
```

Input 3

```
3 6 2
1 1 0
0 1 1
```

Output 3

```
10
```

Explanation

In the first sample, the long test has three questions worth 6 points in total, and the passing score is 2. The first student got the first question correct and the second and third questions incorrect. The second student got the first and second questions incorrect, and the third question correct. There are three ways to allocate points so that both students pass:

- 3 points for the first question, 1 points for the second question, and 2 points for the third question.
- 2 points for the first question, 2 points for the second question, and 2 points for the third question.
- 2 points for the first question, 1 points for the second question, and 3 points for the third question.

Problem C
Kelly Kawno and the Circuit Factory

Time Limit: None

This is an output-only problem.

Welcome to Kelly Kawno's Circuit Factory! Kelly Kawno is the mastermind behind the greatest circuit factory on Earth. Everywhere you look, it's different kinds of circuits, as far as the eye can see. Unfortunately, Ms Kelly is busy giving a tour right now to some other five children who found golden tickets in their graphics cards, so you'll have to settle for me as your guide today. Shall we begin?

Circuits are (more or less) just Boolean expressions. A Boolean expression is just like an arithmetic expression, except all values can only be either `1` or `0` (as such, we call these "Boolean values"). The circuits in our factory accept **at most** 10 variables $x_0, x_1, x_2, \dots, x_9$ as input. Then, we can perform *operations* on these values. The `AND`, `OR`, `NAND`, and `NOR` operators each accept two Boolean values as input, and produce one Boolean value as output. Each one's behavior is summed up in the following tables, where `a` and `b` are the input values.

a	b	a b AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a b OR
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a b NAND
0	0	1
0	1	1
1	0	1
1	1	0

a	b	a b NOR
0	0	1
0	1	0
1	0	0
1	1	0

The resulting value after evaluating some operator can be further used as input for *other* operators. This nesting allows us to create some pretty intricate expressions! For example, consider the following Boolean expression.

```
x0 x1 AND x2 x1 x2 NOR OR NAND x0 AND
```

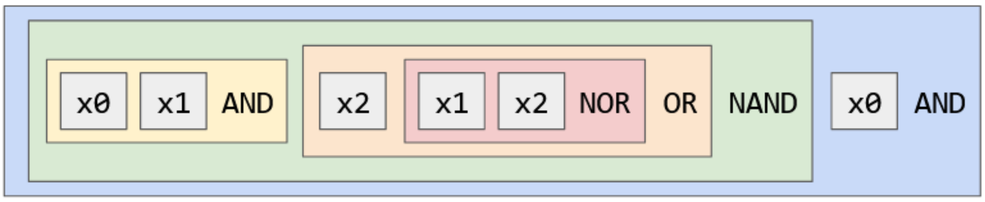
Note our use of *postfix* notation here, where we place the operator *after* its two arguments. This is in contrast to the *infix* notation that you might be more familiar with from arithmetic, where the operator goes *between* its two operands. Postfix notation has a lot of advantages over infix notation:

- Implementing a parser for it is significantly easier. Try it!
- Even without parentheses or an explicit PEMDAS-like convention, the order of operations is always completely unambiguous.

If we were to place explicit parentheses in the above expression, then it would be:

```
(((x0 x1 AND) (x2 (x1 x2 NOR) OR) NAND) x0 AND)
```

However, parentheses are not needed. The following diagram shows how to break down our example Boolean expression. See how the expressions' nestedness can be implied by the notation, even without explicit parentheses.



As an exercise, you can verify that if $x_0 = 1$, $x_1 = 0$, and $x_2 = 1$, then the above Boolean expression evaluates to 1.

We have many different kinds of circuits in this factory. But practically speaking, there is a sense in which two different circuits might be considered "the same". We say that two circuits are *equivalent* if they both produce the *same* output for any configuration of inputs. For example, you can verify that our example Boolean expression from earlier is equivalent to the following, simpler expression:

```
x0 x1 x2 NAND AND
```

In our factory, we partition all of our circuits into different *production lines*, where two circuits belong to the same production line if and only if they are equivalent.

Oh, Ms. Kawno is back! It seems that she was looking for someone to inherit her Circuit Factory, but all five children had failed her challenge. Perhaps you would like to try?

Ms. Kawno has selected twenty different circuits. Your challenge is to create a circuit that is equivalent to each one, using **as few** operators as possible.

Oh, and also, to make it more exciting, Ms. Kawno has required that you **only use** her new experimental operators, which she calls `BLAND` and `BLOR`. Their behavior is summed up in the following tables—be careful, the operators aren't commutative!

a	b	a b BLAND
0	0	0
0	1	1
1	0	0
1	1	0

a	b	a b BLOR
0	0	1
0	1	1
1	0	0
1	1	1

Why are they called `BLAND` and `BLOR`? Consider that an extra puzzle from Ms. Kawno, free of charge!

In summary, for each of the given circuits, construct an equivalent one that **only** uses `BLAND` and `BLOR`, and which does so using as few operators as possible. Specifically, Ms. Kawno herself was able to replicate each circuit, and her operator count will serve as a "target" which you need to try to match (or surpass!)

Input and Output Format

This is an output-only problem. You will be provided with twenty test files, each containing a test case. Each test file follows the file name format of `input_xx.txt`, and for each, you submit to it by uploading a file `output_xx.txt`, containing an equivalent `BLAND`/`BLOR` circuit.

Each test file and each submission should consist of a single line, containing a valid Boolean expression in postfix notation. To be absolutely explicit, we define valid Boolean expressions in postfix notation as follows:

- `xi` is a valid expression, where `i` is a digit from `0` to `9`. This represents the value of the variable x_i.
- If `a` and `b` are valid expressions, then `a b op` is also a valid expression, whose value is the result of evaluating the operator `op` with `a` as the first argument, and `b` as the second argument.
  - For test files, `op` is guaranteed to be one of `AND`, `OR`, `NAND`, `NOR`.
  - For your submissions, `op` must only be either `BLAND` or `BLOR`.

Each test file contains no more than 10^5 operators.

Your submissions should each contain no more than 10^5 operators.

Summaries of the twenty test cases are provided below.

- **Tests 1 to 4** contain exactly one operator.
- **Tests 5 to 8** contain only one *kind* of operator
- For the remaining tests, consider the production lines whose output values only depend on $x_0, x_1, \ldots, x_{k-1}$. Among them, four production lines were uniformly randomly sampled, and from each, we selected a circuit which only uses the variables $x_0, x_1, \ldots, x_{k-1}$.
  - **Tests 9 to 12** have $k = 2$
  - **Tests 13 to 16** have $k = 6$
  - **Tests 17 to 20** have $k = 10$

Submitting to CMS

In a given submission, you can upload your answers to any subset of the test cases.

If you don't upload a file to some test case, then your score for that test case will be computed using your most recently uploaded output file for it (or skipped, if you haven't submitted to this test case yet). Note that this means you **don't** have to re-upload all your output files for every submission—only the new output files need to be uploaded.

Scoring

Each correct test case is worth up to 5 points. Each test case has a "target value" T, meaning you should be aiming to use T or fewer operators in your submission.

If your submission to a test case is correct, then you are awarded a baseline of 2 points (yay!). Let c be the number of operators used in your submission. The remaining 3 points are then awarded based on how close you managed to bring c to T. In general, a smaller c is better, with full points being awarded when $c \leq T$. The precise scoring formula is given below.

$$\begin{cases} 0, & 10^5 < c, \\ 2, & 20T < c \leq 10^5, \\ 2 + 3 \times \dfrac{20T - c}{20T - T}, & T < c \leq 20T, \\ 5, & c \leq T. \end{cases}$$

If your submission is incorrect, then you will get 0 points for that test file.

The different targets T per test case are summarized in the following table.

Test Cases	T
1 to 4	3
5 to 6	12
7	1600
8	2000
9 to 12	3
13 to 16	150
17 to 20	3000

Sample I/O

Input

```
x1 x0 NOR x2 AND x1 OR
```

Output

```
x0 x1 BLAND x2 x0 BLOR BLAND x1 BLOR
```

How to do File I/O

Most modern programming languages have libraries for reading from and writing to files.

- For C++, you would use the `fstream` library
- For Python, you would use the `open()` function

However, it is possible to *redirect* the standard input and output of your program (i.e. `cin` and `input()`, and `cout` and `print()`) to make them read from or write to **files**, instead of the console (and this is how some online judges automatically test your programs).

When running your executable or Python program from the terminal, you can add the following command line arguments.

- `< filenameIn.txt` means that your program will read the contents of `filenameIn.txt` as the source of its standard input.
- `> filenameOut.txt` means that your program will write all its standard output to the file `filenameOut.txt`

Both arguments can be used in the same command. For example,

```
               circuit.exe < input_01.txt > output_01.txt
       python3 circuit.py < input_01.txt > output_01.txt
```

would run the respective executable or Python program, such that its standard input reads from `input_01.txt` and its standard output writes to `output_01.txt`

Reading until EOF

To keep reading input until end-of-file in C++, you can use the following code:

```cpp
std::string my_token;
while (std::cin >> my_token) {
    // use my_token
}
```

The program exits from this loop when there is no input left to read.

For Python, note that the entire circuit is given in one line, so you can use `input().split()` to read one line of input and then split it into a list of space-separated tokens.

Problem D
Goyo's Game

Time Limit: 1.5 seconds

Pusit Bulaga is the Philippines' hottest new game show, but many aren't aware of the original komiks that the creator had admitted had inspired his show—Goyo's Game. Randomly selected Filipinos from all over the country are invited to participate in the Goyo's Game, where contestants are encouraged to lie and cheat and manipulate their opponents in different psychological mind games. The stakes are high—winners earn a fortune, but losers are plunged into irrecoverable debt.

Our protagonist is also named Goyo, a genetically bio-engineered clone of Gregorio del Pilar. He joined the game to protect his great-great-great-grand-daughter, a trusting girl who was tricked into joining the game against her will. Will they triumph? Or will they lose and accrue a crippling debt. Or perhaps, they aim their sights higher... to fight the corrupt organizers of the Goyo's Game itself...

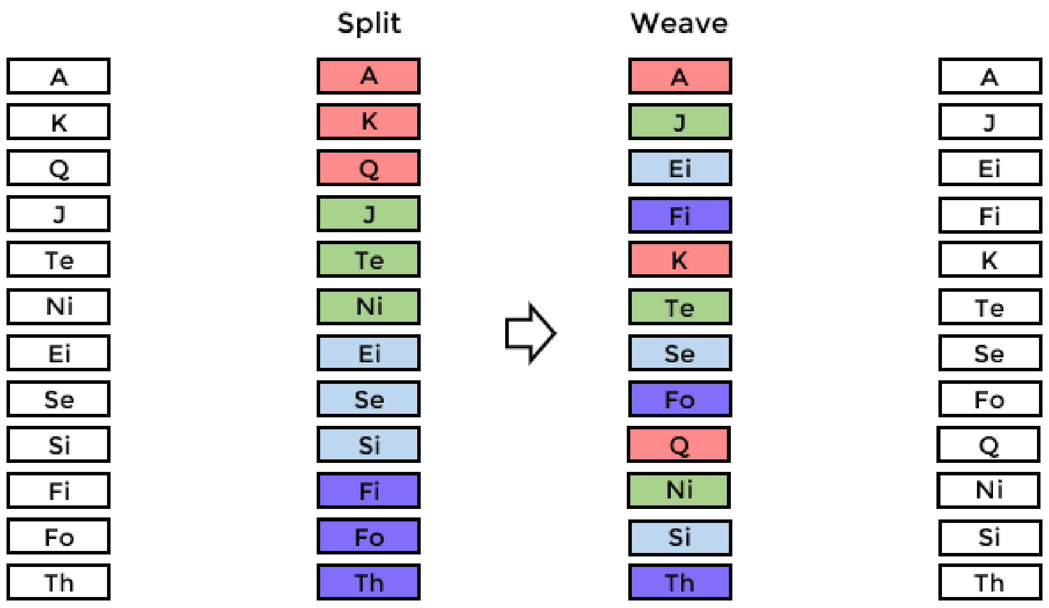
In one chapter, Goyo and his adversary play a game with a deck of n cards, each of which has a distinct label. The dealer shuffles the deck in a very peculiar way...

In one move, Goyo selects a positive integer d such that $1 < d < n$, and d divides n. He shouts it out, and the dealer then performs the following steps:

- Split the deck into d equally-sized stacks: the first n/d cards, then the next n/d cards, and so on.
- "Weave" the stacks together like this: Take the first cards of each stack, in order, then the second cards of each stack, in order, and so on.

Goyo may make as many moves as he wants (possibly zero) and he may select a different divisor d with each move.

In particular, note that if $d = 2$, then the dealer performs a perfect riffle shuffle. For another concrete example, consider the following illustration showing what happens when $n = 12$ and Goyo calls out $d = 4$.



By the end of the last chapter, it seemed like Goyo had been backed into a corner. But then out of nowhere, in the eleventh hour, Goyo was able to snatch victory from the jaws of defeat. How? The reveal is that Goyo was never in trouble, because he had been able to *force* the deck of cards into exactly the order that would ultimately ensure his victory.

He noticed that for their game, the dealer continued to use the same deck that they were using during the practice game, and so with his keen memory, Goyo *knew the initial ordering of the cards in the deck*. Recall that he also knows the *desired final ordering of the deck* that would lead to his victory. Therefore, all he had to do has compute the sequence of moves that he should call out to the dealer to successfully shuffle the deck of cards into his desired final ordering.

The mathematician in you calls BS (Big Sus) on this explanation. Can this strategy really work as described? Let's find out!

Given the initial ordering of the cards, and the desired final ordering of the cards, determine whether or not there actually exists a series of moves which can shuffle the former into the latter. Also, if yes, provide a sequence of moves which performs the shuffle using *as few moves* as possible.

Input Format

The first line of input contains a single integer n, the number of cards in the deck.

The second line describes the initial ordering of the deck. It contains n space-separated strings, each representing a card in the deck, given in the order they initially appear.

The third line describes the desired final ordering of the deck. It contains n space-separated strings, each representing a card in the deck, given in the order that Goyo wants them to appear.

Note that the cards' descriptions are case sensitive (e.g. `A` is different from `a`).

Output Format

First, output a line containing `YES` if the task is possible, or `NO` if it is not.

Then, if you answered `YES`, output a nonnegative integer k, the number of moves that you wish to perform.

If $k = 0$, there should be **no further output**. If $k > 0$, output a line with k space-separated integers, describing, in order, the d of each move of your solution. You are reminded that each d should be a divisor of n.

If there are multiple solutions, any that minimize k will be accepted. Given the constraints, it can be shown that k is always of a "reasonable" output size.

Constraints and Subtasks

For all subtasks
$2 \leq n \leq 2 \times 10^5$ Each card is described by a string of at most 4 upper and lowercase letters The desired final ordering consists of the same cards as the initial ordering

Subtask	Points	Constraints
1	**20**	$2 \leq n \leq 12$
2	**15**	$2 \leq n \leq 52$
3	**15**	$2 \leq n \leq 1000$
4	**10**	$n = 177147$
5	**10**	$n = 194477$
6	**10**	$n = 2 \times 10^5$
7	**20**	No further constraints.

Sample I/O

Input 1	Output 1

`6`
`AoS JoC KoH ToC QoD NoD`
`AoS KoH QoD JoC ToC NoD`
	`YES`
	`1`
	`3`

Input 2	Output 2

`8`
`Beg One Two Thr Fou Fiv Six End`
`Beg Six Fiv Fou Thr Two One End`
	`NO`

Input 3	Output 3

`2`
`a b`
`a b`
	`YES`
	`0`

Explanation

In the third sample test case, there exists no d such that $1 < d < 2$ and d divides 2, so Goyo has no valid moves. But, the answer is still `YES` because the empty sequence of moves solves the task. Because $k = 0$, we do not print any further output.