Problem 1 – Basic BASIC

Do you know Elio? Elio is a young man who hates almost everything. One of the things he hates the most is all old programming languages. Recently he found an old diskette with strange code on it. And because he is very curious he wants to execute the code and view the result. He hates old programming languages, so he asks you to write a C# program to execute that code and find its output result. Since Elio is a really good-natured boy, he deserves your help.

That old language actually was a dialect of the well-known programming language BASIC. We will call it Basic BASIC (BB). So what are the important things about the BB language?

First of all, every single line of the BB code starts with an integer number – the unique identifier of the line. Every line has unique identifier bigger than all its preceding lines. One BB code line can contain only one command. After executing the command (and if no **GOTO** executed), then the code execution continues from the line with the smallest unique identifier which is bigger than the unique identifier of the current line. The code execution starts from the line with the smallest unique identifier.

The second important thing about the BB is that you can only use 5 variables. These variables are named **V**, **W**, **X**, **Y** and **Z**. They can only have integer values between -2 000 000 000 and 2 000 000 000, inclusive. The default value of these 5 variables is 0. In other words, if no value is preliminarily assigned to them their value will be 0.

Finally, the most important thing in the BB language is the commands. In BB there are 5 types of commands listed below:

1. Changing the value of a variable. You can assign values to any of these variables by simply using the assign (**=**) command. BB also has 2 arithmetic operations available – addition (**+**) and subtraction (**-**). Here are four examples for assigning values to variables: **V=-5**, **X=Y**, **W=X-Y**, **Z=Z+1**. Only one arithmetic operation is allowed with a single assigned command. Note that these lines are invalid: **V=X+Y+Z**, **W=1-4+3**, **X=-1-X**, **X=-2--3**.
2. Conditional command execution (**IF** <condition > **THEN** <command>). The conditions in the **IF** operator can only be comparisons: bigger than (**>**), less than (**<**) and equal to (**=**) between *variable and variable*, *variable and number* or *number and number*. The **IF** statement executes the command after the **THEN** statement if and only if the condition is true. For example the following BB code will assign value 5 to the **X** variable only if the **Y** variable is greater than the **Z** variable: **IF Y > Z THEN X = 5**. If the condition is false, then the code execution continues exactly from the next line (the line after the **IF** command).
3. Unconditional branching (**GOTO**). The **GOTO** operator unconditionally jumps to a command line with the specified unique identifier. That line will always exist. For example the command **GOTO 10** will continue the code execution from a command with the unique identifier 10. **GOTO** "parameter" will be always a number, thus commands like **GOTO X** will be invalid.
4. Manipulating the output (**CLS** and **PRINT**).
    a. If the BB code reaches the **CLS** (clean screen) command, it automatically clears the content printed so far and starts printing from the beginning.
    b. The **PRINT** command gets the value of a variable and writes it in a single line in the output. For example **PRINT Z** will print the value of the **Z** variable on the output and then will write a new line.

5. Command for stopping the code execution – **STOP**. The **STOP** command immediately stops the execution of the BB code.

Note that the BB language **ignores all spaces** because they do not affect the semantics of the language.

Input

The input data should be read from the console.

The input will consist of valid code written in Basic BASIC (BB), always ending with a line containing the word "**RUN**" (see the examples below).

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

You must write on the console the output result from executing the BB code given in the input.

Constraints

- The given BB code will always be uppercased and valid in the syntax described above.
- Line identifiers will always be between 0 and 10 000, inclusive.
- It is guaranteed that the code will always reach its **STOP** command (or the end of the code) after no more than 1 000 000 command executions.
- None of the 5 variables (**V**, **W**, **X**, **Y** and **Z**) will have values smaller than -2 000 000 000 or bigger than 2 000 000 000 in any part of the code execution.
- Allowed working time for your program: 0.8 seconds.
- Allowed memory: 16 MB.

Examples

Input example	**Output example**

`5 X=-1`
`6 IF X=-1 THEN X=0`
`7 PRINT X`
`8 CLS`
`10 PRINT X`
`20 X=X+1`
`30 IF X < 4 THEN GOTO 10`
`40 STOP`
`50 PRINT X`
`RUN`
	`0`
	`1`
	`2`
	`3`

`0    X   =    1`
`1    Y     =     2`
`2 Z    =    Y    -    X`
`5    PRINT          X`
`6    PRINT      Z`
`10          X    =    X    +1`
`20   IF  X  =  Y  THEN  GOTO 2`
`RUN`
	`1`
	`1`
	`2`
	`0`

Problem 2 – Crossword

Write a program to create a crossword by a given list of words. The crossword must contain exactly **N** lines with **N** characters. You will be given **N*2** words containing exactly **N** capital Latin letters ('A' – 'Z').

The crossword is a combination of those words placed in a table (with no spaces or empty cells). You can combine the words either horizontally (in a row) or vertically (in a column), but each word should match one of the pre-given words. Each line and each column of the crossword should contain a word from the given list of words. The words should be placed from left to right and from the top to the bottom of the crossword. You are not obligated to use all words, you can also put the same word more than once but the only words you can use are the words you are given.

Input

The input data should be read from the console.

On the first input line you will be given an integer **N** – the width and the height of the crossword.

On the next **2*N** lines you will be given **2*N** words with exact length **N**. You must use these words for creating the crossword.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console. You should print the generated crossword.

If more than one solution exists you must print the crossword that is first in the lexicographical order. This means that if we concatenate the lines of the produced crossword from the first to the last, the obtained sequence of characters should be the smallest in the alphabetical order among all possible crosswords that can be generated using the given words.

If no crossword can be made with the given words then print "`NO SOLUTION!`" on the only output line as shown on the second example below.

Constraints

- **N** will be between 1 and 6, inclusive.
- Allowed working time for your program: 0.75 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
4 FIRE ACID CENG EDGE FACE ICED RING CERN	FACE ICED RING EDGE

Input example	Output example
3 ABC DEF GHI JKL MNO PQR	NO SOLUTION!

On the first example there is more than one solution but the one that is shown is the lexicographically lowest one.

On the second example there is no solution.

Problem 3 – Indices

You are given a zero-based array **ARR** with **N** integer numbers in it. Each element of **ARR** is an index in the **ARR** (seems like a recursive definition, right?).

You are also given the sequence that starts from the first element (**0**) then moves to the element with index **ARR[0]**, then moves to the element with index **ARR[ARR[0]]**, then moves to the element with index **ARR[ARR[ARR[0]]]**, and so on…

The full sequence is generated by performing these actions until you reach an index that is outside the bounds of the array **ARR**. Of course cycles are absolutely possible. When a cycle is started in the sequence it may never reach any index that is outside the bounds of the **ARR**.

Write a program that outputs the elements in the given sequence. When you find cycle you should output it in round brackets as shown in the examples below. Please note that no spaces should be printed between the brackets and the number.

Input

The input data should be read from the console.

In the first input line you are given the number **N** of the elements in **ARR**.

From the second line you should read the numbers in the array **ARR** separated by a single space.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

On the only output line you should print the described sequence. All the cycles should be printed with round brackets, with no spaces between the brackets and the numbers.

Constraints

- **N** will be between 1 and 200 000, inclusive.
- Numbers in ARR will be between -2 000 000 000 and 2 000 000 000, inclusive.
- Allowed working time for your program: 0.25 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example

`6`
`1 2 3 5 7 8`
	`0 1 2 3 5`

`6`
`1 2 3 5 7 1`
	`0(1 2 3 5)`

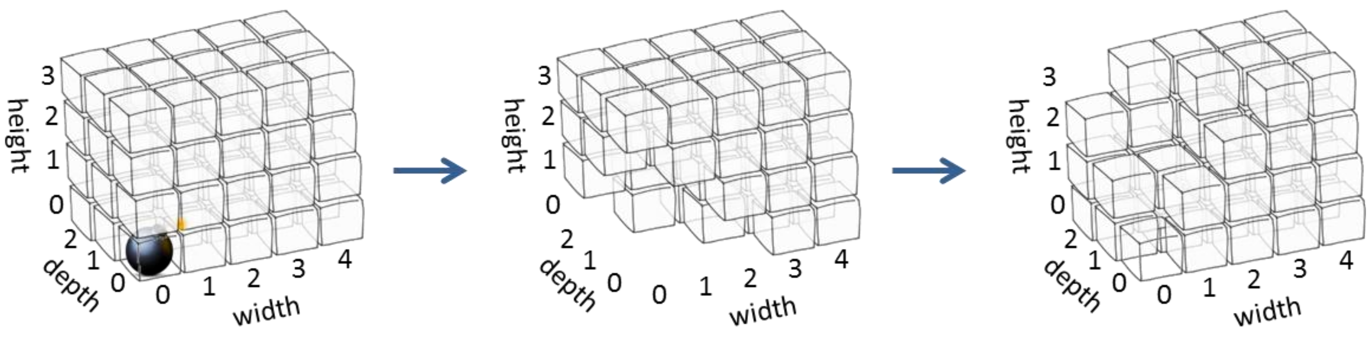
Problem 4 – Bombing Cuboids

You are given a **rectangular cuboid** of size **W** (width), **H** (height) and **D** (depth) consisting of **W** * **H** * **D** cubes, each colored in some color. Each **color** is denoted by a unique capital letter from the Latin alphabet: 'Y' is yellow, 'R' is red, 'B' is blue, 'G' green, etc. When a **bomb** with power **p** is detonated at certain position {**w**, **h**, **d**} inside the cuboid it destroys all the cubes around it which are located at distance ≤ **p** from the bomb, while all other cubes remain unaffected.

After a detonation the destroyed cubes leave an empty space in the cuboid and the cubes staying above them **fall down** due to the gravitation and fill the empty space. The fall down process moves the cubes to smaller coordinates by the 'height' axis.

The distance between the bomb and given cube is calculated as **standard Euclidian distance** in the 3D space from the bomb's center to the given cube's center. For example, the distance between a bomb located at position {5, 6, 2} and the cube at position {3, 0, 4} is approximately 6.63324958.

At the figure below a cuboid of size 5 x 4 x 3 is shown and the detonation of a bomb with power 2 at coordinates {0, 0, 0} is illustrated along with the cubes fall down after it:



Your task is to write a program which enters a cuboid from the console and simulates over it a sequence of N bomb explosions (at given coordinates and with given power) and calculates how many cubes of each color are destroyed by the bomb attack.

All explosions happen one after another: the first bomb is detonated, it creates an empty space in the cuboid, the cubes above the empty space fall down, then the second bomb is detonated, etc.

Input

The input data should be read from the console. At the first line 3 integers **W**, **H** and **D** are given separated by a space. These numbers specify the width, height and depth of the cuboid. At the next **H** lines the colors of the cubes in the cuboid are given as **D** sequences of exactly **W** letters. Each sequence of **W** letters is separated from the next with a single space. At the next line a single integer **N** is given – the number of bombs. At the next **N** lines the bombs are given, each as a sequence of 4 integers separated by a space: **w**, **h**, **d**, and **p**.

The input data will be correct and there is no need to check it explicitly.

Output

The output data should be printed on the console.

At the first line of the output print the **total number cubes destroyed**. On the next few lines print the cubes destroyed by colors in the following format: **color** followed by a space and by **amount**. Only colors with non-zero amount of cubes should be listed.

The colors should be listed in alphabetical order.

Constraints

- The numbers **W**, **H** and **D** are all integers in the range [1…100].
- The letter sequence in the input consists of capital Latin latters only
- The number **N** is an integer in the range [0…20].
- The value **w** is an integer in the range [0…**W**-1].
- The value **h** is an integer in the range [0…**H**-1].
- The value **d** is an integer in the range [0…**D**-1].
- The value **p** is an integer in the range [1…50].
- Allowed work time for your program: 0.15 seconds.
- Allowed memory: 16 MB.

Examples

Input	Output

`4 3 5`
`AAAA AAAA AAAA AAAA AAAA`
`AAAA AAAA AAAA AAAA AAAC`
`ABAA AAAA AAAA AAAA AAAA`
`3`
`1 2 3 1`
`0 0 0 2`
`0 0 0 2`
	`22`
	`A 21`
	`B 1`

Input	Output

`7 4 3`
`BRYYYYY RYYYYGY YRYYYYY`
`YYYGBGY YYYYGGG YYYGGGY`
`RYBYGYY RYYYYGY RYBYGBB`
`RYBYGYY RBYYGYY RYBYGBB`
`2`
`6 3 2 4`
`1 1 1 2`
	`72`
	`B 10`
	`G 14`
	`R 6`
	`Y 42`

Note that in the first example the first line of letters is the bottom of the cuboid (h=0) and the last line of letters (h=2) is the top of the cuboid. More precisely, the letter "B" at the first example is located at position {1, 2, 0} and the letter "C" is located at position {3, 1, 4}.

At the both examples the locations of the explosions are shown in gray background.

Problem 5 – Academy Tasks

As you know in our Academy we give you some problems to solve. You must first solve problem 0. After solving each problem **i**, you must either move on to problem **i+1** or skip ahead to problem **i+2**. You are not allowed to skip more than one problem. For example, {0, 2, 3, 5} is a valid order, but {0, 2, 4, 7} is not because the skip from 4 to 7 is too long.

You are given an array **pleasantness** (0-based), where **pleasantness[i]** indicates how much you like problem i. We will let you stop solving problems once the range of pleasantness you've encountered reaches a certain threshold. Specifically, you may stop once the difference between the maximum and minimum pleasantness of the problems you've solved is greater than or equal to the integer **variety**. If this never happens, you must solve all the problems. Return the minimum number of problems you must solve to satisfy our requirements.

Input

The input data should be read from the console.

On the first input line you will be given the list of numbers in **pleasantness** separated by a comma and a space (see the examples below).

On the second input line you will be given the integer **variety**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

On the only output line you must print the minimum number of problems you must solve to satisfy our requirements.

Constraints

- **pleasantness** will contain between 1 and 50 elements, inclusive.
- Each element of **pleasantness** will be between 0 and 1000, inclusive.
- **variety** will be between 1 and 1000, inclusive.
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Input	Output	Explanation

`1, 2, 3`
`2`
	2	Solve the 0-th problem and the 2-nd after it.

`1, 2, 3, 4, 5`
`3`
	3	Obviously, the first and the last problems should be solved. Skip a problem ahead twice in a row.

`6, 2, 6, 2, 6, 3, 3, 3, 7`
`4`
	2	You can stop after solving the first 2 problems.