Telerik Software Academy 2012/2013 – C# Part 2 – Practical Exam – Variant 1

Exam terms – Read carefully!

During the exam you **are allowed** to use any teaching materials, lectures, books, existing source code, and other paper or Internet resources.

Direct or indirect communication with anyone in the class or outside the room is absolutely **forbidden**.

The address of the judge system is: http://bgcoder.com. Trainers will give you the contest password. Only your **last** code submission will be evaluated. All decimal separators in all tasks will be "." (Dot).

You have exactly **6 hours** to solve all given problems. Good luck!

After the end of the exam you should leave all used papers (including this one) in the room!!!

Problem 1 – Kaspichan Numbers

In Kaspichan we have a special way to write numbers. We use the following 256 digits:

0	A
1	B
…	…
25	Z

26	aA
27	aB
…	…
51	aZ

52	bA
53	bB
…	…
77	bZ

…

234	iA
235	iB
…	…
255	iV

We write the numbers as sequences of digits. The last digit of the number (the most right one) has a value as shown in the above table. The next digit on the left has a value 256 times bigger than the shown in the above table, the next digit on the left has 256*256 times bigger value than the shown in the table and so on. Your task is to write a program to **convert a decimal number into its corresponding representation in Kaspichan**.

Input

The input data consists of a single integer number.

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

Output

The output data consists of a single text line holding the result and should be printed at the console.

Constraints

- The input number is in the range [0…18 446 744 073 709 551 615] inclusively.
- Allowed work time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
20	U

Input	Output
30	aE

Input	Output
280	BY

Input	Output
1000	DhY

Problem 2 – Greedy Dwarf

Joro is a very nice dwarf, but he is a little greedy. He and his friend Gosho heard about a valley that they can collect lots of coins. Unfortunately they can also lose some of the coins.

Joro and Gosho are given a set of maps which show a correct pattern of steps through the valley. Since Joro and Gosho are greedy dwarfs, they want you to help them find the correct pattern, so they can collect most coins.

You are given a sequence of numbers, the coins in the valley, and a set of patterns. The coins in the valley are numbers from -1000 to 1000, meaning the dwarfs should either get 1000 coins, or leave 1000 coins. The set of patterns is represented as many arrays, each containing numbers.

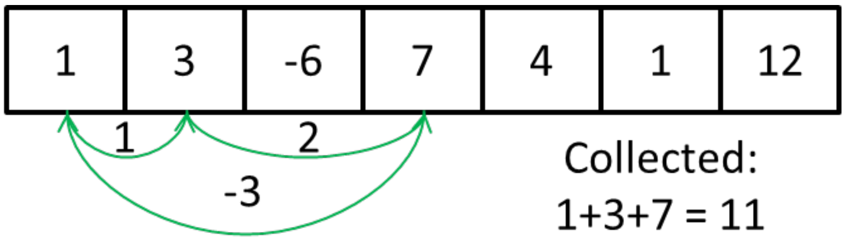
The patterns consist of numbers and show where the dwarfs should go from their current position in the valley. The dwarfs should follow the patterns exactly.

Given the valley:

1	3	-6	7	4	1	12

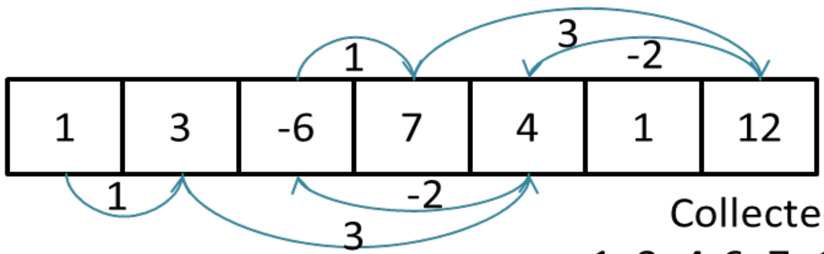
And the following patterns, the routes to escape are as follows:

1	2	-3



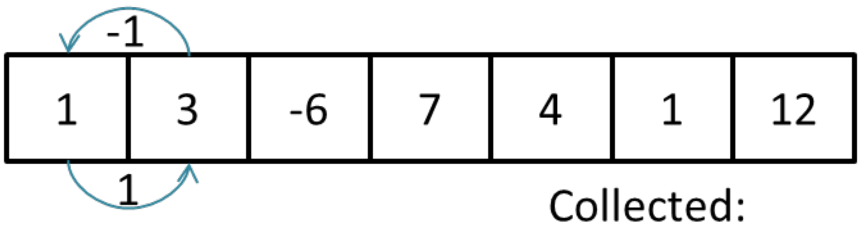
Collected:
1+3+7 = 11

1	3	-2



Collected:
1+3+4-6+7+12 = 21

1	-1



Collected:
1+3 = 4

The pattern is used as follows:

Start from the first number in the valley (valley[0]) and add the number to the coins collected. Then use the first number in the pattern (pattern[0]) and go to position (0 + pattern[0]) in the valley. Collect the coins from position 0 + pattern [0] and go to position (0 + pattern[0]) + pattern[1]m collect the coins, etc…

When all the numbers from the pattern are used, start the pattern from the top.

The dwarfs escape the valley when they step on already visited position in the valley, or when they go out the valley (i.e. their current position + the number in the pattern is leading outside of the valley).

Help the dwarfs and find the best pattern to use, so they can collect most coins.

Input

The input data should be read from the console.

On the first line you will be given the valley - numbers separated with ", " (comma and space).

On the second line you will be given **M** – the number of patterns.

On each of the next **M** lines you will be given numbers, separated with ", "(comma and space), representing the patterns.

Output

The output data should be printed on the console.

The output should contain the maximal number of coins, that the dwarfs can collect using one of the patterns

Constraints

- **The numbers in the valley** will be between 1 and 10 000 inclusive.
- **M** will be between 1 and 500
- Each pattern will contain at most **100 numbers**
- Each of the numbers in the valley or in the patterns will be between -1000 and 1000
- Allowed work time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Example

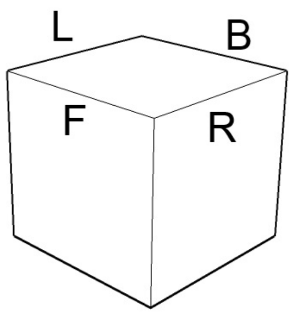
Input example	Output example

`1, 3, -6, 7, 4 ,1, 12`
`3`
`1, 2, -3`
`1, 3, -2`
`1, -1`
	21

Problem 3 – Slides

You are given a **rectangular cuboid** of size **W** (width), **H** (height) and **D** (depth) consisting of **W** * **H** * **D** cubes. The top row (layer) of the cube is considered to be with height 0. Each cube can contain one of the following things – slide, teleport, basket or just be an empty cube. A small ball that can fit through a single cube is dropped over the cuboid. The ball can pass through cubes according to some rules and can exit the cuboid only through the bottom side, but not through the walls. Your task is to find if the ball can exit the cuboid and also the coordinates of the final cube that it got to (either before exiting or when stuck somewhere). The elements of the cuboid are given as strings:

- Slide – The ball slides on it to the next level of the cuboid in the given direction. Each slide is given in the following pattern "S X" where X is the direction. The directions are: "L" – left, "R" – right, "F" – front, "B" – back, "FL" – front left, "FR" – front right, "BL" – back left, "BR" – back right. E.g. if the ball is in cube [2, 0, 0] with slide (S BL) it slides to cube[1, 1, 1]. If the ball can't slide in the given direction (i.e. to a wall), it's stuck and can't exit the cuboid. If the ball is on a slide on the last row it is considered that it can slide and exit regardless of the direction.



- Teleport – Surprisingly the ball gets teleported to another cube on the same row. Each teleport is given in the following pattern "T W D" where "W" and "D" are the width and the depth of the destination cube e.g. (T 1 2) Two teleports will not point to each other.
- Empty cube – The ball falls through it and goes directly to the next level. It is given as a single letter "E" e.g. (E)
- Basket – If the ball falls in a basket it's stuck there and can't exit. The basket is given as single letter "B" e.g. (B)

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 cubes are given as **D** sequences of exactly **W** strings each one in brackets "( )". Each sequence of **W** strings is separated from the next one by " **|** " (space + vertical line + space). At the last line there will be two integer numbers **ballW** and **ballD** that specify the location where the ball is initially dropped.

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.

On the first output line you should print "Yes" if the ball can exit the cuboid or "No" otherwise.

On the second line print the 3 coordinates (w, h, d) of the last cube that the ball got to.

Constraints

- The numbers **W**, **H** and **D** are all integers in the range [3…50].
- The numbers **ballW** and **ballD** specify the indexes in the cuboid array.
- 0 ≤ ballW < W, 0 ≤ ballD < D
- Allowed working time for your program: 0.3 seconds. Allowed memory: 16 MB.

Examples

Example input	Example output
3 3 3 (S L)(E)(S L) | (S L)**(S R)**(S L) | (B)(S F)(S L) (S B)(S F)(E) | (S B)**(S F)(T 1 1)** | (S L)(S R)(B) (S FL)**(S FL)**(S FR) | (S FL)(S FL)(S FR) | (S F)(S BR)(S FR) 1 1	Yes 1 2 0

Example input	Example output
3 3 3 (S L)**(E)**(S L) | (S L)(S R)(S L) | (B)(S F)(S L) (S B)**(S R)**(E) | (S B)(S F)(T 1 1) | (S L)(S R)(B) (S FL)(S FL)**(B)** | (S FL)(S FL)(S FR) | (S F)(S BR)(S FR) 1 0	No 2 2 0

Problem 4 – Console Justification

Joro is having problems. He broke his PC (and his bed, and another bed, and several glasses, and several other things…) on his birthday party and has to write the description of the next task for the PC Magazine & Telerik programming contest. Now, Joro has a very old PC, which can only run a console. Sure, he can use that to write the description, but there's a small problem. The console can't justify the text, like MS Word can – Joro needs that functionality, and he needs it fast!

He decided that he can do justification by placing additional whitespaces between the words, by following some rules regarding the justified text:

1. A word is separated from other words by at least one whitespace (or new line)
2. Each line must have a fixed width **W** – number of symbols including whitespaces – Joro knows this width in advance
   - This width will be at least the length of the longest word in the text
3. Each line must have at least one word, and each word can be on exactly one line
   - E.g. a word cannot have several characters on one line and several on the next
4. If a line contains exactly one word, there must be no whitespaces after it – this is the only case in which the line can have a width less than **W**
5. We will call the whitespaces between two consecutive words "gaps"
6. One gap **can** have at **most 1 whitespace more** than any other gap
7. One gap **cannot** have **more whitespaces** than any other **gap to its left**
   - So, a gap can have a whitespace more than another gap "on its right", but not vice versa
8. Justification happens by fitting as much words as possible on one line, and continuing with the next

- o The first line is formed by taking all words that can fit on that line from the text
- o The next line is formed by taking all words that can fit on that line from the remaining text, and so on…
- o The order of the letters in the text (excluding whitespaces) must remain the same

Write a program which justifies text, by using whitespaces, according to the rules described. Note that just placing each word from the text on a separate line could be a violation of rule 8.

Input

The input data should be read from the console.

On the first line of the input there will be the number **N** – the total number of lines in Joro’s initial text.

On the next line of the input there will be the number **W** – the number of symbols, of which each of the justified text lines must consist

Each of the next **N** lines will contain a line of Joro’s text, consisting of Latin letters and whitespaces (a line in the input text will have at least one word on it)

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 print **the justified text**, **line by line**, with **each line having exactly W characters** (unless it contains exactly one word, in which case it must have exactly as many characters as there are in the word).

Constraints

- 0 < N < 1000
- 0 < W < 10000, no word will have more than W symbols
- There will be no empty lines in the input
- There can be sequences of whitespaces in the input
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Input example	**Output example**

`5`
`20`
`We happy few        we band`
`of brothers for he who sheds`
`his blood`
`with`
`me shall be my brother`
	`We   happy   few  we`
	`band   of   brothers`
	`for   he  who  sheds`
	`his  blood  with  me`
	`shall  be my brother`

```
10
18
Beer beer beer Im going for
   a
beer
Beer beer beer Im gonna
drink some beer
I love drinkiiiiiiiiing
beer
lovely
lovely
beer
```

```
Beer  beer beer Im
going  for  a beer
Beer  beer beer Im
gonna  drink  some
beer     I    love
drinkiiiiiiiiing
beer        lovely
lovely        beer
```

Problem 5 – One Task is Not Enough

Solving one task is too mainstream. You are given two tasks. Solve them both.

First task

A strange street is filled with lamps numbered 1 through **N**, initially all turned off. Joro has birthday and wants to **turn on all the lamps**. He makes multiple actions turning the lights on, beginning with lamp 1. On the first action, he turns on the first lamp, and every 2nd lamp that is turned off thereafter. On the 2nd action, he turns on the first lamp that is turned off, and every third lamp that is turned off thereafter. In general, on the **i**th action, he turns on the first lamp that is turned off, and every (i+1)th lamp that is still turned off thereafter.

For example, with 8 lamps, on the first action Joro turns on 1, 3, 5 and 7, leaving 2, 4, 6, and 8. On the second action he turns on 2 and 8, leaving 4 and 6. On the third action he turns on lamp 4, and on the final action lamp 6.

You will be given **N**, the number of lamps. Find the number of the lamp turned on last.

Second task

Robot Joro (“Joro the bot”) is standing on some arbitrary point in the infinite plane.

Joro is given a **sequence of commands** that he has to execute. The commands are given as a string. Each character of this string is a single command. The commands must be executed in the given order.

There are 3 types of commands: '**S**' means "step forward", '**L**' means "turn 90 degrees to the left", and '**R**' means "turn 90 degrees to the right". All steps that Joro makes have the same length.

Since Joro is a bot, he will be executing the commands forever: after he executes the last character of the commands, he will always start from the beginning again.

We say that Joro’s path is **bounded** if there is some positive real number R such that while executing the infinite sequence of steps he will never leave the circle with radius R steps and center at his starting location.

Given the list of commands (as a single line string), your program should determine whether Joro’s path will be bounded or not. Print the string "*bounded*" (quotes for clarity) if the path is bounded and "*unbounded*" if it is not.

Input

The input data should be read from the console.

Your program should solve one input of the first task and two inputs of the second task.

The number of lamps **N** will be given on the first input line.

A string of commands for Robot Joro will be given on the second input line.

Another string of commands for Robot Joro will be given on the third input line.

Joro executes the two strings of commands separately!

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 first output line print the solution of the first task.

On the second output line your program should print the solution of the second task with the input data from the second input line.

On the third output line your program should print the solution of the second task with the input data from the third input line.

Constraints

- **N** will be between 1 and 2 000 000, inclusive.
- The number of commands for Robot Joro will be between 1 and 2500, inclusive (for each test).
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Example input	**Example output**
8 SRSL SSSSR	6 unbounded bounded

Example input	**Example output**
9 SSSSR L	6 bounded bounded

Example input	**Example output**
12 L SRSL	12 bounded unbounded