Problem 1 – Genome Decoder

In modern molecular biology and genetics, the genome is the entirety of an organism's hereditary information. A genome sequence is a combination of the 4 Latin letters A, G, T and C.

An encoded genome is a genome sequence where all sub-sequences of same consecutive letters (with length at least 2) are replaced with the length of the sub-sequence followed by the letter that is repeating. For example the genome sequence "*AAGGGCTTT*" will be encoded as "*2A3GC3T*". The decoded genome is the original genome that is used for generating the encoded genome. In the given example the encoded genome is *2A3GC3T* and the decoded genome is *AAGGGCTTT*.

You will be given an encoded genome and your task is to decode it and then output it in a special format. You should output only **N** letters per line. On each line every **M** letters must be separated by a single space. At the start of each line you should print the line number (starting from 1) followed by a space. The line numbers should be aligned to the right using empty spaces (as shown in the second example). The last output line should not contain any spaces at the beginning nor the ending of the line.

Input

The input data should be read from the console.

From the first input line you should read 2 integer numbers – **N** and **M** separated by a single space.

From the second input line you should read the encoded genome sequence.

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 decoded genome in the format described. See the examples below.

Constraints

- **N** will be an integer number between 1 and 1000, inclusive.
- **M** will be an integer between 1 and 1000, inclusive. **M** will be always lower than or equal to **N**.
- The encoded genome will contain only digits and the capital Latin letters 'A', 'C', 'G' and 'T'.
- The length of the decoded genome will be between 1 and 100 000, inclusive.
- Allowed working time for your program: 0.15 seconds. Allowed memory: 16 MB.

Examples

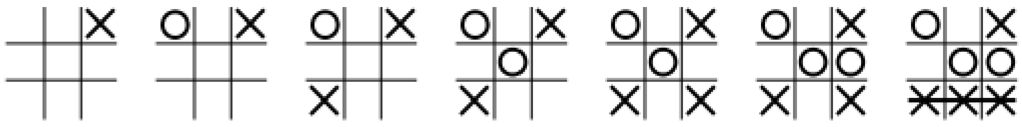
Input example	Output example
`6 3`	`1 AAA AAA`
`10A15G3CA6T19C`	`2 AAA AGG`
	`3 GGG GGG`
	`4 GGG GGG`
	`5 GCC CAT`
	`6 TTT TTC`
	`7 CCC CCC`
	`8 CCC CCC`
	`9 CCC CCC`

Input example	Output example
`9 4`	` 1 AAAA AAAA A`
`18A13C10T10GA18GT17C`	` 2 AAAA AAAA A`
	` 3 CCCC CCCC C`
	` 4 CCCC TTTT T`
	` 5 TTTT TGGG G`
	` 6 GGGG GGAG G`
	` 7 GGGG GGGG G`
	` 8 GGGG GGGT C`
	` 9 CCCC CCCC C`
	`10 CCCC CCC`

Problem 2 – Tic-Tac-Toe

You all know the game called Tic-Tac-Toe. Tic-Tac-Toe is a pencil-and-paper game for two players, **X** and **O**, who take turns marking the spaces in a 3×3 grid. The **X** player always starts first. The player who succeeds in placing three respective marks in a horizontal, vertical, or a diagonal row wins the game.

The following example game is won by the first player, **X**:



You will be given the current state of the game and your program should find:

- the number of the games from the current state that end with a win for the first player (**X**),
- the number of the games from the current state that end with no winner and
- the number of the games from the current state that end with a win for the second player (**O**).

Input

The input data should be read from the console.

The current state of the game will be written in the console in 3 lines with exactly 3 symbols, each representing one of the nine cells of the current state of the game with '**X**', '**O**' (capital Latin letter O) and '**–**' (dash) – for the empty cells. 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.

On the first output line you should print the number of the games that the first player (**X**) is going to win.

On the second line you should print the number of the even games.

And on the third output line you should print the number of the games that the player **O** is going to win.

Constraints

- The input data will always be a valid Tic-Tac-Toe game state where the player **X** starts first.
- Allowed working time for your program: 0.2 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
OO- XXO XXO	1 0 0

Input example	Output example
OO- X-X ---	32 0 35

Problem 3 – Airplane Drinks

Antonio and Bobbi (also known as A&B) are flying on an airplane and are anxiously waiting for their coffee.

There are **N** seats in the plane numbered from 1 to **N**, one seat in each row. All seats are occupied, thus there are **N** passengers overall (including Antonio and Bobbi). A stewardess will serve a cup of coffee or tea to each passenger. She needs to serve tea to all passengers whose seat numbers are given to your program, and she needs to serve coffee to all other passengers.

A coffee and tea machine is located just in front of the first seat of the plane. The stewardess has a flask that can contain enough coffee or tea to serve at most **7** passengers. Initially, the stewardess is located near the coffee and tea machine and the flask is empty.

The stewardess can perform the following kinds of actions:

- Move from the coffee and tea machine to seat 1 or move from seat 1 to the coffee and tea machine. Each of these two actions will take **1** second.
- Move from seat **i**, **i > 1**, to seat **i-1**. It will take **1** second.
- Move from seat **i**, **i < N**, to seat **i+1**. It will take **1** second.
- If she is near seat **i**, the passenger at this seat has not yet been served and the current type of drink in the flask is the same as the ordered one, she can serve this passenger with a cup of drink he/she wants. It will take **4** seconds.
- If she is near the coffee and tea machine and the flask is empty, she can fill the flask with either tea or coffee (but not both simultaneously). It will take **47** seconds. Note that she can fill the flask partially (to serve less than 7 passengers), but it will still take **47** seconds.

Given **N** and the seat numbers of the passengers who must be served with tea, return the minimal time needed for the stewardess to serve all passengers with proper drinks and return to the coffee and tea machine.

Input

The input data should be read from the console.

The number **N** of all seats will be written on the first input line.

The number of the passengers that must be served tea will be written on the second line.

In each of the next lines you must read the respective passenger that needs to be served tea. 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.

On the only output line you must print the minimal time needed for the stewardess to serve all passengers with proper drinks and return to the coffee and tea machine.

Constraints

- **N** will be between 2 and 44 777 777, inclusive.
- The number of the passengers that must be served with tea will be between 1 and 47, inclusive.

- Each passenger who must be served tea will be between 1 and **N**, inclusive. All passenger numbers will be distinct.
- Allowed working time for your program: 0.4 seconds.
- Allowed memory: 64 MB.

Examples

Input example	Output example
9 2 3 1	154

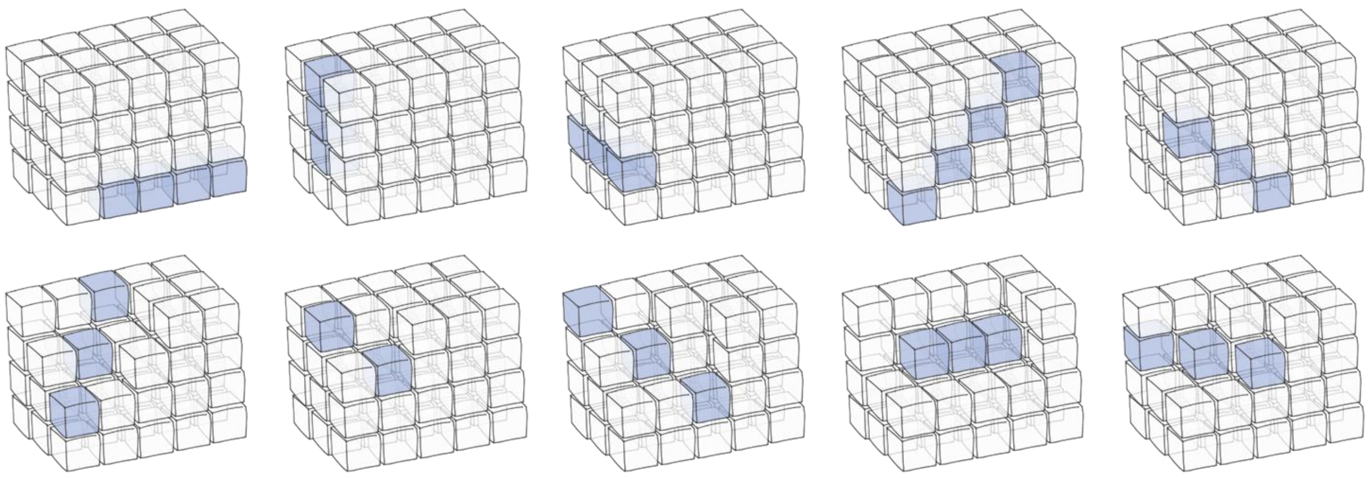
Input example	Output example
7 2 3 2	142

Input example	Output example
5 4 4 2 1 5	130

Problem 4 – 3D Lines

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.

Two cubes are **neighbors** when they share a common side, common edge or common point. A **3D line** is a sequence of neighbor cubes in the same color staying on the same horizontal, vertical, or diagonal in the cuboid. At the figure below few examples of 3D lines are shown:



Your task is to write a program that finds the length of the longest 3D line and how many 3D lines with this length exist in the cuboid.

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. The colors of the cubes in the cuboid are given as **D** sequences of exactly **W** letters in the next **H** lines. Each sequence of **W** letters is separated from the next one with a single space.

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.

In the first line of the output print the **length of the longest 3D line** in the cuboid followed by a space and **how many 3D lines** with this length exist in the cuboid. If **no lines exist** in the cuboid print "**-1**" at the first line of the output.

Constraints

- The numbers **W**, **H** and **D** are all integers in the range [1…50].
- The letter sequence in the input consists of capital Latin latters only
- Allowed work time for your program: 0.3 seconds.
- Allowed memory: 32 MB.

Examples

Input	Output
4 3 5 **A**AAA **A**AAA **A**BAA **A**AAB **A**ABA AAA**A** BBB**A** AAB**A** AAB**A** ABA**A** B**B**BB A**B**BB A**B**BB A**B**AB B**B**AA	5 3

Input	Output
7 4 3 BR**YYYYY** RYYYYGY YR**YYYYY** YYYGBGY YYYYGGG YYYGGGY R**BBBBB**Y RYYYYGY RYBYGBB RYBYGYY RBYYGYY **RRRRR**YB	5 4

Input	Output
2 2 2 AB EF CD GH	-1

Input	Output
3 3 3 AAD AXD BXD AXX XAX BXA CCC XXA BAA	3 7

Problem 5 – Guitar

Bobi is a guitar player and he is going to play a concert. He don't like to play all the songs at the same volume, so he decides to change the volume level of his guitar before each new song. Before the concert begins, he makes a list of the number of intervals he will be changing his volume level by before each song. For each volume change, he will decide whether to **add** that number of intervals to the volume, or **subtract** it.

You are given a list of integers **C**, the i-th element of which is the number of intervals Bobi will change his volume by before the **i**-th song. You are also given an integer **B**, the initial volume of Bobi's guitar, and an integer **M**, the highest possible volume setting of Bobi's guitar. Bobi cannot change the volume of his guitar to a level above **M** or below 0 (but exactly 0 and exactly **M** is possible). Your program should print the maximum volume Bobi can use to play the last song. If there is no way to go through the list without exceeding **M** or going below 0, print -1.

Input

The input data should be read from the console.

The elements of the list **C** will be on the first input line separated by a comma and an interval (`", "`).

On the second line there will be the number **B** and on the third line there will be the number **M**.

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 **-1** or the maximum volume **Bobi** can use to play the last song.

Constraints

- **C** will contain between 1 and 50 elements, inclusive.
- In 95% of the tests cases **C** will contain no less than 34 elements.
- Each element of **C** will be between 1 and **M**, inclusive.
- **M** will be between 1 and 1000, inclusive.
- **B** will be between 0 and **M**, inclusive.
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example

`5, 3, 7`
`5`
`10`
	`10`

Input example	Output example

`15, 2, 9, 10`
`8`
`20`
	`-1`

Input example	Output example

`74, 39, 127, 95, 63, 140, 99, 96, 154, 18, 137, 162, 14, 88`
`40`
`243`
	`238`