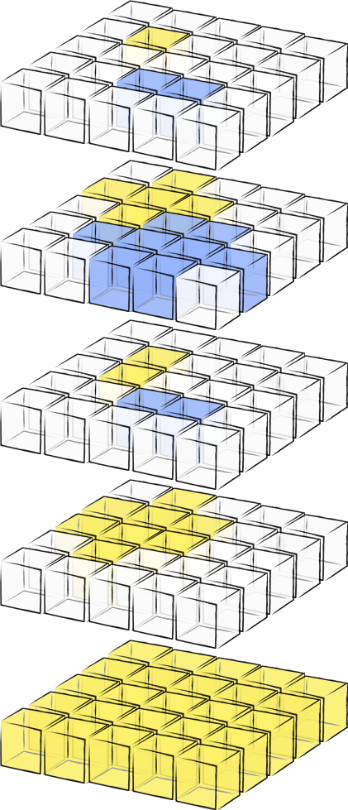
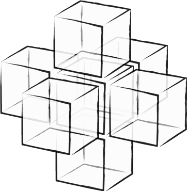
# Lab: Problem Solving

This document defines the **in-class exercise** assignments for the ["Algorithms" course @ Software University](https://softuni.bg/opencourses/algorithms).

# Stars in the Cube

We are given a **cube of Latin letters** of size **n** \* **n** \* **n** given as **n** layers (square matrices) of size **n** \* **n**. A **cube**, split into **layers**, is shown on the **right** (each letter is shown as different color).

Write a program to calculate how many **3D stars of 7 cells** (center, up, down, left, right, front, back) holding **equal letters** exist in the cube.

The form of the **3D star** is shown at the figure on the **left**. The same letter can be shared between several stars (stars can overlap inside the cube).

## Inputs

* The input is read from the console.
* The first line holds an integer **n** – the size of the cube.
* At the next **n** lines, the layers of the cube are given as sequence of **n** matrices separated by ‘**|**’.
* The cells in each matrix row are separated by space (see the examples below).

## Output

* At the **first line** at the console print the **total number of 3D stars** of equal letters in the cube.
* At the next few lines, for **each letter** in alphabetical order print the **number of its stars** found in the cube in format “letter -> count”. Skip the letters that don’t have any stars in the cube.

## Constraints

* The size of the cube **n** is integer in the range **[1…75]**.
* All cube **cells** hold lowercase **Latin letters** in the range **[‘a’… ‘z’]**.
* Time limit: **200 ms**. Allowed memory: **32 MB**.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  a a a a a | a p a a a | a a a a a | a p a a a | p p p p p  a p a a a | p p p a a | a p a a a | p p p a a | p p p p p  a a a a a | a p x x a | a p a a a | p p p z a | p p p p p  a a x x a | a x x x x | a a x x a | a p z z z | p p p z p  a a a a a | a a x x a | a a a a a | a a a z a | p p p p p | 6  a -> 1  p -> 3  x -> 2 |
| 3  x x x | x a x | x x x  x a x | a a a | x a x  x x x | x a x | x x x | 1  a -> 1 |
| 2  a a | a a  a a | a a | 0 |

# Guitar

Bobi is a guitar player and he is going to play a concert. He doesn’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** | **Output** |  | **Input** | **Output** |
| 5, 3, 7  5  10 | 10 |  | 15, 2, 9, 10  8  20 | -1 |
| **Input** | | | | | | **Output** |
| 74, 39, 127, 95, 63, 140, 99, 96, 154, 18, 137, 162, 14, 88  40  243  **2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2**  **50**  **69** | | | | | | 238 |

# Shortest Path

You are lost in the dark walking home. Luckily, you have a map for the shortest path available to your house. Well, sort of a map, you have turn directions. The only possible directions for each turn are straight (written with “S”), left (written with “L”) and right (“written with “R”). So, the map looks like the following: LSRLRSRLLR, which means – take left turn, straight, right turn, left turn, right turn, straight, right turn, left turn, left turn, right turn and you are home. Well… at least looked like that during the last century, because now the map is quite old and some of symbols cannot be read from it (written with “\*”). For example, you may have LR\*\*SR\*LL, which means – take left turn, right turn, all directions are possible, all directions are possible, straight, right turn, all directions are possible, left turn, left turn and you are home. Every “\*” can be either “S”, “L” or “R”. Your task is to find all possible different paths, which can be formed from the partial map.

## Input

The input data should be read from the console.

On the first and only input line there will be the partial map as sequence of “**S**”, “**L**”, “**R**” and “**\***”.

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 number of possible different paths.

On the next output lines, print every possible different path (each on separate line), sorted alphabetically.

## Constraints

* The length of the map will be maximum 16 symbols, inclusive.
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| LSLLRSRL | 1  LSLLRSRL |
| R\*S\*L | 9  RLSLL  RLSRL  RLSSL  RRSLL  RRSRL  RRSSL  RSSLL  RSSRL  RSSSL |
| \*\*RLR\* | 27  LLRLRL  LLRLRR  LLRLRS  LRRLRL  LRRLRR  LRRLRS  LSRLRL  LSRLRR  LSRLRS  RLRLRL  RLRLRR  RLRLRS  RRRLRL  RRRLRR  RRRLRS  RSRLRL  RSRLRR  RSRLRS  SLRLRL  SLRLRR  SLRLRS  SRRLRL  SRRLRR  SRRLRS  SSRLRL  SSRLRR  SSRLRS |