Telerik Software Academy – C# Fundamentals Part 1 – Exam

Problem 1 – Fighter Attack

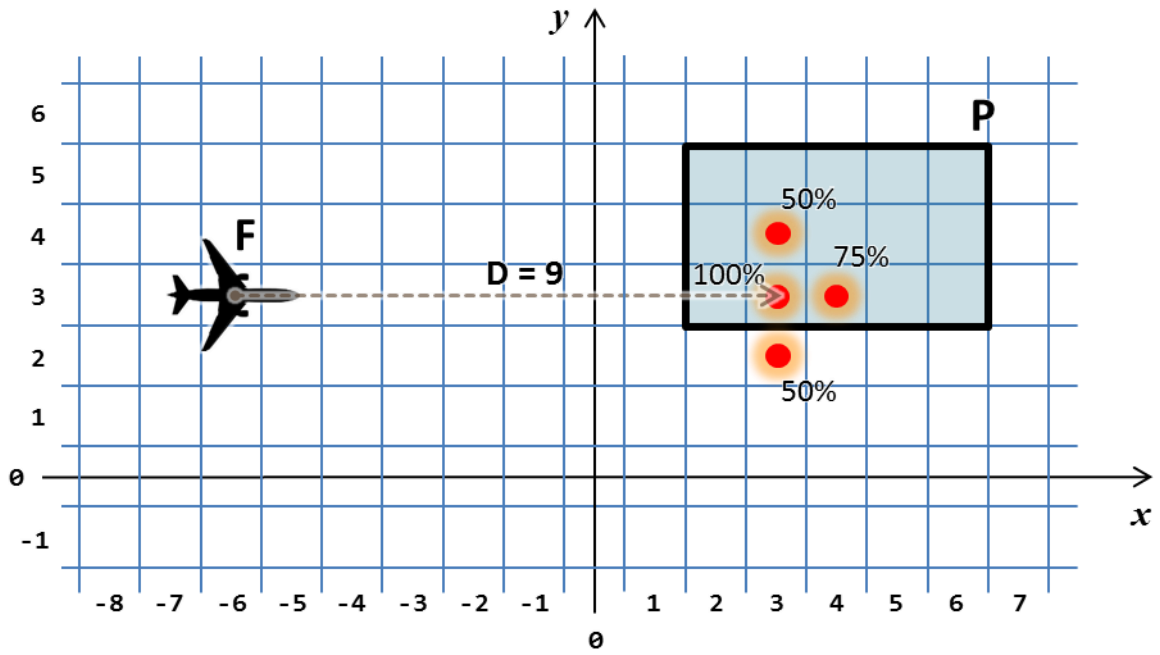
A rectangular plant **P** located in east-west direction is under attack by a fighter aircraft flying over it on the west. When the fighter launches a missile we have its coordinates **F**. It is assumed that the missile's direction is always straight on the west and the missile always hits the target after a fixed distance **D** in front of the fighter.

To simplify our model we assume the land is built of square cells of size 1 x 1 located in east-west direction and each cell has integer Cartesian coordinates {x, y}. In this model the plant can be represented by a rectangular area of cell and the missile always hits some of the square cells (inside or outside of the plant).

When the missile hits a certain cell, the damage over it is 100%, on the cells staying on the left and on the right of it the damage is 50% and in front of it the damage is 75%. The **total damage** is sum of the separate damages and can exceed 100%.

You are given the location of the plant P, the location of the fighter **F** and the distance **D**. Write a program that calculates the damage over the plant after the attack. Note that the missile could hits the plant partially of fully or can hit some area outside of the plant and cause no damage.

At the figure below a plant **P**, a fighter **F**, a distance **D** and the missile hit point are shown along with the damage caused over the cells by the hit. Note that some of the damaged cells are outside of the plant and thus the total damage is 225%:



Your task is to write a program that calculates the total damage caused after the attack over the plant.

Input

The input data should be read from the console.

There will be exactly **7** lines holding the integer numbers P_{X1}, P_{Y1}, P_{X2}, P_{Y2}, F_X, F_Y, and **D**. The plant **P** is given by the coordinates of any two of its opposite corners and is non-empty (consists of at least one

cell). The location of the fighter is given as cell coordinates **F_X**, and **F_Y** and the distance **D** is given as an integer number.

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

Output

The output should consist of a single line holding the total damage given as percentage.

Constraints

- The numbers **P_{X1}**, **P_{Y1}**, **P_{X2}**, **P_{Y2}**, **F_X**, **F_Y**, and **D** are all integers in the range [-100 000 … 100 000].
- Allowed work time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Input	Output
2 5 6 3 -6 3 9	225%

Input	Output
2 5 6 3 -6 5 7	75%

Input	Output
6 5 2 3 0 1 -3	0%

Problem 2 – Astrological Digits

The astrological digit of a given number **N** is a digit calculated by the number's digits by a special algorithm. The algorithm performs the following steps:

(1) Sums the digits of the number **N** and stores the result back in N.

(2) If the obtained result is bigger than 9, step (1) is repeated, otherwise the algorithm finishes.

The last obtained value of **N** is the result, calculated by the algorithm.

Input

The input data should be read from the console.

The only line in the input contains a number **N**, which can be integer or real number (decimal fraction).

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 calculated astrological digit of the number **N** on the first and only line of the output.

Constraints

- The number **N** will be in range $[-1.7 \times 10^{-308} \ldots 1.7 \times 10^{308}]$
- **N** will have no more than 300 digits before and after the decimal point.
- The decimal separator will always be the "**.**" symbol.
- Allowed working time for your program: 0.2 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
8	8
-1337	5
1234567.8900	9

Problem 3 – Sand-glass

Once upon a time a powerful wizard was born. His name was Gwenogfryn and soon he became a great sorcerer. Kind-hearted he was. He would only use his magic to protect humans from the evil witches that would come at night. Gwenogfryn, however was a pacifist and did not want to fight or hurt the witches, so he came up with another solution. He would catch the witches and throw them into a sand-glass (the only prison a witch cannot escape from). Unfortunately, he is running out of sand-glasses. Help Gwenogfryn catch all witches by making your own sand-glasses.

Input

The input data should be read from the console.

You have an integer number **N** (always odd number) showing the height of the sand clock.

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

Output

The output should be printed on the console.

You should print the hourglass on the console. Each row can contain only the following characters: “.” (dot) and “*” (asterisk). As shown in the example: the middle row must contain only one ‘*’ and all other symbols must be “.”. Every next row (up or down from the middle one) must contain the same number of ‘*’ as the previous one plus two. You should only use “.” to fill-in the rows, where necessary.

Constraints

- The number **N** will be a positive integer number between 3 and 101, inclusive.
- Allowed working time for your program: 0.25 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
5	***** .***. ..*.. .***. *****
7	******* .*****. ..***.. ...*... ..***.. .*****. *******

Problem 4 – Dancing Bits

Gergana loves dancing and she also likes bits (she doesn't know what bits really are, but she knows that she likes them). Few days ago she accidently invented a new term - “**dancing bits**”.

If you ask her what “dancing bits” mean she will tell you that it’s a sequence of identical bits (so the bits can dance together – zeros can only dance with other zeros, the same applies for ones).

You are given **N** positive integer numbers that are converted to binary numeral system and are concatenated together in one big sequence of bits.

For example: if we have 4 numbers: **5** (`101` in binary numeral system), **6** (`110` in binary numeral system), **14** (`1110` in binary numeral system) and **143** (10001111 in binary numeral system) their concatenation will be **`101110111010001111`.**

You are also given a positive integer **K** - the number of identical bits (zeroes or ones that can dance together).

Write a program that finds the number of all “dancing bits” (the sequences of equal bits) with a length of exactly **K bits**. Your program should search in the concatenation of the given **N** numbers.

For example, if we have **4** numbers (**5**, **6**, **14** and **143**, the concatenation of their binary representation is 10**111**0**111**01**000**1111) and we are searching for the total number of all sequences of equal bits with an exact length of **3** bits, the answer will be **3** (the sequences are bolded in the concatenation above).

In this example we have two sequences of “dancing bits” - "`111`" consisting of only ones and one sequence of “dancing bits” - "`000`" consisting of only zeros. Note that the sequence "`1111`" is not a sequence of exact 3 identical bits.

Input

The input data should be read from the console.

At the first input line there will be one positive integer – the number **K**.

At the second input line there will be another positive integer – the number **N**.

At each of the next **N** lines there will be one positive integer – the N numbers that represent the input sequence of bits.

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.

The only output line must contain the answer – the number of “dancing bits” sequences found.

Constraints

- The number **K** will be positive integer number between 1 and 25 600, inclusive.
- The number **N** will be positive integer number between 1 and 800, inclusive.
- Each of the **N** numbers will be positive integer numbers between 1 and 2 147 483 647, inclusive.
- Allowed working time for your program: 0.25 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
3 4 5 6 14 143	3
1 4 2 10 42 170	20

Problem 5 – Lines

You are given a list of **8** bytes (positive integers in the range [0…255]) $\mathbf{n_0}$, $\mathbf{n_1}$, …, $\mathbf{n_7}$.

These numbers represent a square grid consisting of **8** lines and **8** columns.

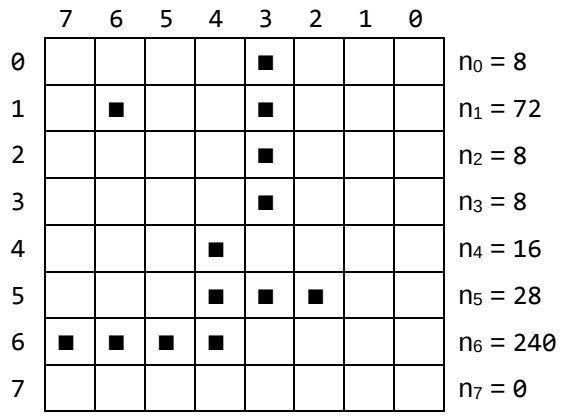
Each cell of the grid could either be empty or full. The first line is represented by the bits of $\mathbf{n_0}$, the second – by the bits of $\mathbf{n_1}$ and so on, and the last line is represented by the bits of $\mathbf{n_7}$.

Each bit with value 1 denotes a **full cell** and each bit with value 0 denotes an **empty cell**.

The lines are numbered from the first (top) to the last (bottom) with the numbers 0, 1, …, 7.

The columns are numbered from right to left with the indices 0, 1, …, 7.

The figure shows a sample square grid and its representation by a sequence of 8 numbers $\mathbf{n_0}$, $\mathbf{n_1}$, …, $\mathbf{n_7}$:



A **line** is any sequence of full cells staying on the same row or column.

At the figure above we have two lines of 4 cells and two lines of 3 cells and one line of 1 cell.

You need to create a program that finds the longest line in the grid and the number of lines with the longest length.

At the figure we have two largest lines with length of 4 cells.

Input

The input data is should be read from the console.

There will be exactly 8 lines each holding the integer numbers n_0, n_1, …, n_7.

It is guaranteed that there exists at least one line in the grid (the grid is not empty).

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

Output

The output consists of two integers placed on separate lines.

The first line should hold the length of the longest line in the grid.

The second line should hold the number of lines with the maximal length.

Constraints

- The numbers n_0, n_1, …, n_7 are positive integers in the range [0…255].
- Allowed work time for your program: 0.25 seconds.
- Allowed memory: 16 MB.

Example

Input Example	Output Example
8 72 8 8 16 28 240 0	4 2
246 247 248 249 250 251 252 253	8 4