# Multidimensional Arrays

## Diagonal Difference

Create a program that finds the **difference between the sums of the square matrix diagonals** (absolute value).



### Input

* On the **first line**, you are given the integer **N** – the size of the square matrix
* The next N **lines** hold the values for **every row** – N numbers separated by a space

### Output

* Print **the absolute** difference between **the sums** of the primary and the secondary diagonal

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  11 2 4  4 5 6  10 8 -12 | 15 | **Primary diagonal:** sum = 11 + 5 + (-12) = 4  **Secondary diagonal:** sum = 4 + 5 + 10 = 19  **Difference:** |4 - 19| = 15 |

## 2X2 Squares in Matrix

Find the count of **2 x 2 squares of equal chars** in a matrix.

### Input

* On the **first line**, you are given the integers **rows** and **cols –** the matrix’s dimensions
* Matrix characters come at the next **rows** lines (space separated)

### Output

* Print the number of all the squares matrixes you have found

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3 4  A B B D  E B B B  I J B B | 2 | Two 2 x 2 squares of equal cells:  A **B B** D A B B D  E **B B** B E B **B B**  I J B B I J **B B** |
| 2 2  a b  c d | 0 | No 2 x 2 squares of equal cells exist. |

## Maximal Sum

Create a program that reads a rectangular integer matrix of size **N x M** and finds in it the square **3 x 3** that **has a maximal sum of its elements**.

### Input

* On the first line, you will receive the rows **N** and columns **M**. On the next **N lines,** you will receive **each row with its columns**

### Output

* Print the **elements** of the 3 x 3 square as a matrix, along with their **sum**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Matrix** | **Output** |
| 4 5  1 5 5 2 4  2 1 4 14 3  3 7 11 2 8  4 8 12 16 4 |  | Sum = 75  1 4 14  7 11 2  8 12 16 |

## Matrix Shuffling

Write a program that reads a string matrix from the console and performs certain operations with its elements. User input is provided in a similar way as in the problems above – first, you read the **dimensions** and then the **data**.

Your program should then receive commands in the format: "**swap row1 col1 row2 col2**" where row1, col1, row2, col2 are **coordinates** in the matrix. For a command to be valid, it should start with the "**swap**" keyword along with **four valid coordinates** (no more, no less). You should **swap the values** at the given coordinates (cell [row1, col1] with cell [row2, col2]) **and print the matrix at each step** (thus you'll be able to check if the operation was performed correctly).

If the **command is not valid** (doesn't contain the keyword "**swap**", has fewer or more coordinates entered or the given coordinates do not exist), print "**Invalid input!**" and move on to the next command. Your program should finish when the string "**END**" is entered.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 3  1 2 3  4 5 6  swap 0 0 1 1  swap 10 9 8 7  swap 0 1 1 0  END | 5 2 3  4 1 6  Invalid input!  5 4 3  2 1 6 |
| 1 2  Hello World  0 0 0 1  swap 0 0 0 1  swap 0 1 0 0  END | Invalid input!  World Hello  Hello World |

## Snake Moves

You are walking in the park and you encounter a snake! You are terrified, and you start running zig-zag, so the snake starts following you.

You have a task to visualize the snake’s path in a square form. A **snake** is represented by **a string**. The **isle** is a **rectangular matrix of size NxM**. A snake starts going down from the **top-left corner** and slithers its way down. The first cell is filled with the first symbol of the snake, the second cell is filled with the second symbol, etc. The snake is as long as it takes to **fill the stairs**– if you reach the end of the string representing the snake, start again at the beginning. After you fill the matrix with the snake’s path, you should print it.

### Input

* The input data should be read from the console. It consists of exactly two lines
* On the first line, you’ll receive the **dimensions** of the stairs in the format: **"N M"**, where **N** is the number of **rows**, and **M** is the number of **columns**. They’ll be separated by a single space
* On the second line, you’ll receive the string representing the **snake**

### Output

* The output should be printed on the console. It should consist of **N lines**
* Each line should contain a string representing the respective row of the matrix

### Constraints

* The **dimensions** N and M of the matrix will be integers in the range [1 … 12]
* The **snake** will be a string with length in the range [1 … 20] and **will not contain any whitespace characters**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 6  SoftUni | SoftUn  UtfoSi  niSoft  foSinU  tUniSo |

## Jagged Array Manipulator

Create a program that populates, analyzes, and manipulates the elements of a matrix with an unequal length of its rows.

First, you will receive an **integer N** equal to the **number of rows** in your matrix.

On the **next N lines**, you will receive **sequences of integers**, **separated** by a single **space**. Each sequence is a **row** in the matrix.

After populating the matrix, start analyzing it. If a **row** and the **one below** it have **equal length**, **multiply** each **element** in **both** of them by **2**, **otherwise** - **divide** by **2**.

Then, you will receive commands. There are three possible commands:

* "**Add {row} {column} {value}**" - **add** **{value}** to the element at the **given indexes**, if they are **valid**
* "**Subtract {row} {column} {value}**" - **subtract** **{value}** from the element at the **given indexes**, if they are **valid**
* "**End**" - print the **final state** of the **matrix** (all elements **separated by a single space**) and **stop** the program

**Input**

* On the first line, you will receive the **number of rows** of the matrix - integer **N**
* On the next **N** lines, you will receive **each row** - **sequence of integers**, separated by a single **space**
* **{value}** will always be an **integer** number
* Then you will be receiving commands until reading "**End**"

**Output**

* The output should be printed on the console and it should consist of **N lines**
* Each line should contain a string representing the **respective row** of the **final matrix**, elements **separated** by a single **space**

**Constraints**

* The **number of rows** N of the matrix will be an integer in the range [2 … 12]
* The **input** will always **follow** the **format above**
* **Think about data types**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  10 20 30  1 2 3  2  2  10 10  End | 20 40 60 1 2 3 2 2 5 5 |
| 5  10 20 30  1 2 3  2  2  10 10  Add 0 10 10  Add 0 0 10  Subtract -3 0 10  Subtract 3 0 10  End | 30 40 60  1 2 3  2  -8  5 5 |

## Knight Game

Chess is the oldest game, but it is still popular these days. For this task, we will use only one chess piece – the **Knight**.

The knight moves to the **nearest** square but **not on the same**[**row**](https://en.wikipedia.org/wiki/Glossary_of_chess#rank), [**column**](https://en.wikipedia.org/wiki/Glossary_of_chess#file), or [**diagonal**](https://en.wikipedia.org/wiki/Glossary_of_chess#diagonal). (This can be thought of as moving two squares horizontally, then one square vertically, or moving one square horizontally then two squares vertically— i.e. in an "**L**" pattern.)

The knight game is played on a board with dimensions **N x N** and a lot of chess knights **0 <= K <= N2**.

You will receive a board with **K** for knights and '**0'** for empty cells. Your task is to remove a minimum of the knights, so there will be no knights left that can attack another knight.

### Input

On the first line, you will receive the **N** side of the board

On the next **N** lines, you will receive strings with **Ks** and **0s**.

### Output

Print a single integer with the minimum number of knights that needs to be removed

### Constraints

* Size of the board will be 0 < N < 30
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  0K0K0  K000K  00K00  K000K  0K0K0 | 1 |
| 2  KK  KK | 0 |
| 8  0K0KKK00  0K00KKKK  00K0000K  KKKKKK0K  K0K0000K  KK00000K  00K0K000  000K00KK | 12 |