## Rosetta Stone

A team of archeologists has unearthed a set of very special stone tablets – it is believed they contain texts from the lost language of the Olmec and the means to translate them. The team has a large number of tablets, some of which contain texts, and others contain a template for decoding the text into English. Deciphering all of them by hand would be tedious, and this is why the lead professor of the team of archeologists has submitted an inquiry with you for a JS program to do it automatically.

You will be given an **encoded message** and a **template matrix** for **decoding** it. The decoding is done by overlaying the template on top of the stone with the message and performing an **arithmetic operation** with the numbers that overlap. Each pair of numbers (one from the message and one from the template matrix) is **added** together and the resulting number is located on the wheel of letters (pictured to the right), by counting from the beginning (**zero**) and going **clockwise**. You may count **more than one full revolution** around the wheel (e.g. 6 is the same as 33 and so on). Repeat this process for all symbols of the message.

If the decoding template matrix is smaller than the message, **shift the template** **horizontally** by as many cells as it is wide, so it covers the next section of the message (see example). Once you reach the last column of the message matrix, if there are more rows left, **shift** the template back to the **first column**, but **offset it vertically** by as many cells as it is high. See the example’s explanation for more information.

### Input

You will receive an **array of strings**. The first element represents a number **n**, which is the length of the **template matrix** for decoding. The next **n** elements represent rows in the decoding template matrix, with columns **separated by space**. The rest of the elements are all rows of the **encoded message matrix**, with columns in each row **separated by space**.

### Output

Print the decoded message in **uppercase** on a single line on the **console**. The final message is joined from all resulting cells, starting from **top left**, going right, **row by row**. Trailing spaces are **ignored**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [ '2',  '59 36',  '82 52',  '4 18 25 19 8',  '4 2 8 2 18',  '23 14 22 0 22',  '2 17 13 19 20',  '0 9 0 22 22' ] | I CAME I SAW I CONQUERED |

#### Explanation

The line highlighted in red in the input is the number of lines of the decoding template matrix. The lines highlighted in blue are the decoding template itself. The remaining lines are the encoded message matrix. When we overlay the template over the message, 59 and 36 from the first line are on top of 4 and 18, 82 and 52 are on top of 4 and 2.



|  |  |  |
| --- | --- | --- |
| We add each pair to get 63 and 54 on the first line and 86 and 54 on the second. Looking up these numbers on the wheel of letters yields the following:   * For 63 we count two full revolutions and then we count 9 more, which corresponds to the letter ‘I’ * For 54 we count exactly two full revolutions, ending at 0, which is empty space * For 86 we count 3 full revolutions and 5 more, ending at ‘E’ * For 54 we get empty space again | |  |
| We’ve ran out of slots on the template, so we shift it 2 indexes to the right (the width of the template is 2), to decode the next segment of the message. This time we overlay it on top of 25, 19, 8 and 2 and after adding the numbers, we get 84, 55, 90 and 54. They correspond to ‘C’, ‘A’, ‘I’ and empty space, respectively. | |  |
| We keep repeating this for the entire message. Once we reach the end of the row, we shift the template down by its height (2 cells) and back to the beginning of the message. Any slots of the template that overhang at the end of the row are ignored. |  | | |
| The process is repeated for all remaining cells of the message. Note any parts of the template matrix that overhang bellow are ignored. | | | |
|  | | | |
|  | | | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| [ '2',  '31 32',  '74 37',  '19 0 23 25',  '22 3 12 17',  '5 9 23 11',  '12 18 10 22' ] | WE COME IN PEACE |