# Problem 1

First you will be given **a sequence of integers representing males**. Afterwards you will be given another **sequence of integers representing females**.

You have to start from the **first female** and try to match it with the **last male**.

- If their **values** are **equal**, you have to **match them** and **remove both** of them. Otherwise you should **remove only the female** and **decrease** the **value** of the **male** by **2**.
- If someone's value is **equal to or below 0**, you should **remove him/her** from the records **before** trying to **match** him/her with anybody.
- Special case - if someone's **value divisible by 25 without remainder**, you should **remove him/her and** the **next person** of the **same gender before trying to match** them with anybody.

You need to **stop matching** people when you have **no more females or males**.

## Input

- On the **first line** of input you will receive the integers, representing the **males**, **separated** by a **single space**.
- On the **second line** of input you will receive the integers, representing the **females**, **separated** by a **single space**.

## Output

- On the first line of output - print the number of successful matches:
  - "`Matches: {matchesCount}`"
- On the second line - print all males left:
  - If there are no males: "`Males left: none`"
  - If there are males: "`Males left: {maleN}, … , {male3}, {male2}, {male1}`"
- On the third line - print all females left:
  - If there are no females: "`Females left: none`"
  - If there are females: "`Females left: {female1}, {female2}, {female3},…, {femaleN}`"

## Constraints

- All of the given numbers will be valid integers in the range [-100, 100].

## Examples

| Input | Output | Comment |
|---|---|---|
| 4 5 7 3 6 9 12<br>12 9 6 1 | Matches: 3<br>Males left: 1, 7, 5, 4<br>Females left: none | The first pair is the **first female** with value of 12 and the **last male** of value 12, their **values are equal**, so we **match them,** therefore - **remove them** from the **records**. Then we have **two more matches** (9 == 9 and 6 == 6). But the value of the **next male is 3** and the value of the **next female is 1**, it's **not a match** and we **remove** the **female** and **reduce** the **male's value** by 2.Then, we **print** the desired **output**. |
| 3 0 3 6 9 0 12<br>12 9 6 1 2 3 15 13 4 | Matches: 4<br>Males left: none<br>Females left: 15, 13, 4 | |

# Problem 2

You will be given a **string**. Then, you will be given an **integer N** for the **size** of the field with **square** shape. On the next **N** lines, you will receive the **rows** of the field. The player will be placed on a **random position**, marked with **"P"**. On **random positions** there will be **letters**. **All of the empty positions** will be marked with **"-"**.

Each turn you will be given commands for the **player's movement**. If he moves to a **letter**, he **consumes** it, **concatenates** it to the **initial string** and the letter **disappears from the field**. If he tries to move **outside** of the field, he **is punished** - he **loses** the **last** letter **in the string**, **if there are any**, and the **player's position** is **not changed**.

At the end **print all letters and the field**.

## Input

- On the **first line**, you are given the **initial string**
- On the **second line**, you are given the integer **N** - the size of the **square** matrix
- The **next N lines** holds the values for every **row**
- On the next line you receive a **number M**
- On the **next M lines** you will get a move **command**

## Output

- On the first line the **final** state of the **string**
- In the end print **the matrix**

## Constraints

- The size of the **square** matrix will be between **[2…10]**
- The **player position** will be **marked** with "**P**"
- The **letters** on the field will be **any letter except** for "**P**"
- Move commands will be: **"up"**, **"down"**, **"left"**, **"right"**

## Examples

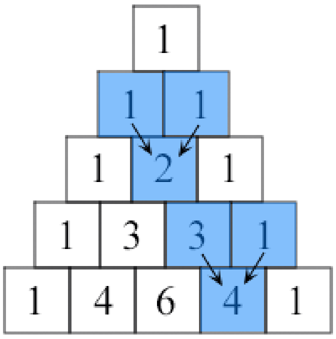
| Input | Output | Comments |
|---|---|---|
| Hello<br>4<br>P---<br>Mark<br>-l-y<br>--e-<br>4<br>down<br>right<br>right<br>right | HelloMark<br>----<br>---P<br>-l-y<br>--e- | The initial string we receive is "Hello". Then we receive 4x4 field and the player is on index [0;0].<br>Then, we start receiving commands. First the player moves to [1;0], where he consumes 'M', and then all letters on the right. Our string is "HelloMark" and the player is on index [1;3]. |
| Initial<br>5<br>-----<br>t-r--<br>--Pa-<br>--S--<br>z--t-<br>4<br>up<br>left<br>left<br>left | Initialr<br>-----<br>P----<br>---a-<br>--S--<br>z--t- | The initial string we receive is "Initial". Then we receive 5x5 field and the player is on index [2;2]. The player consumes 'r' and 't', but also tries to go out of the matrix once, so he loses the last character of his string – 't'. |

# Problem 3

Create a function called **`get_magic_triangle`** which will receive a **single parameter** (integer **n**) and it should create a magic triangle which follows those **rules**:

- We start with this simple triangle **`[[1], [1, 1]]`**
- We generate the **next rows** until we reach **n** amount of rows
- **Each number** in each row is equal to the **sum** of the **two numbers** right **above it** in the triangle
- If the current number has **no neighbor** to the upper **left/rigth**, we just take the **existing neighbor**

After you create the magic triangle, **return** it as a **multidimensional list**. Here is an example with **`n = 5`**



***<u>Note: Submit only the function in the judge system</u>***

## Input

- There will be **no inputs**
- The function will be tested by passing different values of n
- You can test your function with the test code below

## Constraints

- N will be in range **`[2, 100]`**

## Examples

| Test Code | Output |
|---|---|
| `get_magic_triangle(5)` | `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]` |