01. Ramen Shop

You will be given **two sequences** of **integers** representing **bowls of ramen** and **customers**. Your task is to find out **if you can serve all the customers**.

Start by taking **the last bowl** of ramen and **the first customer**. Try to serve every customer with ramen **until we have no more ramen or customers left**:

- **Each time** the value of the ramen **is equal** to the value of the customer, **remove them both** and **continue** with the next **bowl of ramen** and the **next customer**.
- **Each time** the value of the ramen **is bigger** than the value of the customer, **decrease** the **value of that ramen with** the **value of that customer** and **remove the customer**. Then try to **match** the **same bowl of ramen** (which has been decreased) with the **next customer**.
- **Each time** the customer's value **is bigger** than the value of the ramen bowl, **decrease** the **value of the customer with** the **value of the ramen bowl** and **remove the bowl**. Then try to **match** the **same customer** (which has been decreased) with the **next bowl of ramen**.

Look at the examples provided for a better understanding of the problem.

Input

- On the **first line**, you will receive **integers** representing the bowls of ramen, **separated** by a single space and a comma `", "`.
- On the **second line**, you will receive **integers** representing the customers**, separated** by a single space and a comma `", "`.

Output

- If all customers are served, print: **`"Great job! You served all the customers."`**
  - Print all of the left **ramen bowls** (**if any**) separated by **comma and space** in the format: **`"Bowls of ramen left: {bowls of ramen left}"`**
- Otherwise, print: **`"Out of ramen! You didn't manage to serve all customers."`**
  - Print all **customers** left separated by **comma and space** in the format **`"Customers left: {customers left}"`**

Examples

Input	Output
14, 25, 37, 43, 19 58, 23, 37	Great job! You served all the customers Bowls of ramen left: 14, 6

Comment
Start by taking the last bowl 19 and the first customer 58. The customer value is higher, so we remove the bowl and decrease the value of the customer by 19. Now the two lists should look like this: Bowls = [14, 25, 37, 43] Customers = [39, 23, 37] Next, we take the following bowl (43) and continue with the same customer who is 39 now. The value of the bowl with ramen is higher than the customer's value, so we remove the customer and decrease the value of the ramen bowl. Now the two lists should look like this: Bowls = [14, 25, 37, 4] Customers = [23, 37] We take the last bowl of ramen, which is 4 now, and compare it with the next customer (23). The value of the customer is higher, so we decrease his value by 4 and remove the last bowl. Now the two lists should look like this:

Bowls = [14, 25, 37]
Customers = [19, 37]
Then we continue with the ball 37 and customer 19. The bowl is higher. We remove the customer and decrease the bowl value with the value of the customer 19. Now the two lists should look like this:
Bowls = [14, 25, 18]
Customers = [37]
Then we continue with bowl 18 and customer 37. The customer value is higher. We remove the bowl and decrease the customer value with the value of bowl 18. Now the two lists should look like this:
Bowls = [14, 25]
Customers = [19]
Then we continue with the ball 25 and customer 19. The bowl is higher. We remove the customer and decrease the bowl value with the value of the customer 19. Now the two lists should look like this:
Bowls = [14, 6]
Customers = []
We see that we served all of the customers and print the appropriate string for that case. After that, we print the leftover bowls of ramen.

Input	Output
30, 13, 45 70, 25, 55, 15	Out of ramen! You didn't manage to serve all customers. Customers left: 7, 55, 15
Input	**Output**
30, 25 20, 35	Great job! You served all the customers.

02. Martian Explorer

Your rover has landed on Mars, and it needs to find resources to start humanity's first interplanetary colony.

You will receive a **6x6 field on separate lines** with:

- **One rover** - marked with the letter **"E"**
- **Water deposit** (one or many) - marked with the letter **"W"**
- **Metal deposit** (one or many) - marked with the letter **"M"**
- **Concrete deposit** (one or many) - marked with the letter **"C"**
- **Rock** (one or many) - marked with the letter **"R"**
- **Empty positions** will be marked with **"-"**

After that, you will be given the **commands for the rover's movement** on **one line** separated by **a comma and a space** (**", "**). Commands can be: **"up"**, **"down"**, **"left"**, or **"right"**.

For **each command**, the rover **moves in the given directions with one step**, and it can land on one of the given types of **deposit** or a **rock**:

- When it **lands on a deposit**, you must print the **coordinates of that deposit** in the format shown below and **increase its value by 1**.
- If the rover **lands on a rock**, it gets **broken.** Print the **coordinates where it got broken** in the format shown below, and **the program ends**.
- If the rover **goes out of the field**, it should **continue** from the **opposite side** in the **same direction**. Example: If the rover is at **position (3, 0)** and it needs to **move left** (outside the matrix), it should be placed at **position (3, 5)**.

The rover **needs to find at least one of each** deposit to **consider the area suitable** to start our colony.

Stop the program if you **run out of commands** or the **rover gets broken**.

Input

- On the **first 6 lines**, you will receive the **matrix**.
- On the **following line**, you will receive the commands for the rover **separated by a comma and a space.**

Output

- For each deposit found while you go through the commands, print out on the console: **"{Water, Metal or Concrete} deposit found at ({row}, {col})"**
- If the rover hits a rock, print the **coordinates where it got broken** in the format: **"Rover got broken at ({row}, {col})"**

After you go through all the commands or the rover gets broken, print out on the console:

- If the rover **has found at least one of each deposit**, print on the console: **"Area suitable to start the colony."**
- Otherwise, print on the console: **"Area not suitable to start the colony."**

See examples for more clarification.

Examples

Input	Output

- R - - - - - - - - - R - E - R - - - W - - - - - - - C - - M - - - - - down, right, down, right, down, left, left, left	Water deposit found at (3, 1) Concrete deposit found at (4, 3) Metal deposit found at (5, 0) Area suitable to start the colony.
R - - - - - - - C - - - - - - - M - - - W - - - - E - W - R - - - - - - up, right, down, right, right, right	Water deposit found at (3, 2) Water deposit found at (4, 3) Rover got broken at (4, 5) Area not suitable to start the colony.
R - - - - - - - C - - - - - - - M - C - M - R M - E - W - - - - - - - - right, right, up, left, left, left, left, left	Water deposit found at (4, 3) Metal deposit found at (3, 2) Concrete deposit found at (3, 0) Metal deposit found at (3, 5) Rover got broken at (3, 4) Area suitable to start the colony.

03. Words Sorting

Write a function **`words_sorting`** which receives a different number of words.

Create a dictionary, which will have as keys the words that the function received. For each key, create a value that is the sum of all ASCII values of that key.

Then, **sort the dictionary**:

- **By values** in **descending** order, if the **sum of all values** of the dictionary is **odd**
- **By keys** in **ascending** order, if the **sum of all values** of the dictionary is **even**

Note: Submit only the function in the judge system

Input

- There will be **no input**, just any number of words passed to your function

Output

- The function should **return a string** in the format **`"{key} - {value}"`** for each key and value on a **separate lines**

Constraints:

- There will be **no case** with **capital** letters.
- There will be **no case** with a string consisting of **other than letters**.

Examples

Test Code	Output	Comment

`print(`
`  words_sorting(`
`  'escape',`
`  'charm',`
`  'mythology'`
`  ))`
	`charm - 523`
	`escape - 625`
	`mythology - 1004`
		All of the ascii values of the 'escape' word are:
		e = 101, s = 115, c = 99, a = 97, p = 112, e = 101
		Their sum is 625.
		We add it in the dictionary {'escape': 625}.
		The ascii values of the 'charm' are:
		c = 99, h = 104, a = 97, r = 117, m = 109
		Their sum is 523.
		We add it in the dictionary {'escape': 625, 'charm': 625}
		The ascii values of the 'mythology' word are:
		m = 109, y = 121, t = 116, h = 104, o = 111, l = 108, o = 111, g = 103, y = 121.
		Their sum is 1004.
		We add it in the dictionary
		{'escape': 625, 'charm': 523, 'mythology': 1004}
		When we sum 625 + 523 + 1004 = 2152. The result is even, and we sort the dictionary by keys in ascending order.

`print(`
`  words_sorting(`
`  'escape',`
`  'charm',`
`  'eye'`
`  ))`
	`escape - 625`
	`charm - 523`
	`eye - 323`	

`print(`
`  words_sorting(`
`  'cacophony',`
`  'accolade'`
`  ))`
	`accolade - 812`
	`cacophony - 964`	