01.Flower Finder

You will be given **two sequences of characters, representing vowels and consonants**. Your task is to start checking if the following words could be found:

- `"rose"`
- `"tulip"`
- `"lotus"`
- `"daffodil"`

Start by taking the **first character** of the **vowels collection** and the **last character** from the **consonants collection.** Then **check** if these letters are present in one or more of the given words. If a letter is present, that **part of the word** is considered **found**. The word is gradually revealed with each letter found. Continue processing the **next couple of letters** until you find **one of the given words above.**

A **letter (vowels or consonants) could participate in more than one word or more than one time in a word, for example:**

- The letter "**o**" is present in "**rose**", "**lotus**", and "**daffodil**".
- The letter "**l**" is present in "**tulip**", "**lotus**", and "**daffodil**".
- The letter "**f**" is present in the word "**daffodil**" twice.

The consonant and **the vowel** are **always removed** from the collection after trying to match them with the letters in the given words (whether successful or not). In the end, the program **stops** when **a word is found, or there are no more vowels or consonants**.

As a result, if you **found a word**, print **it** and **the remaining letters** in each collection in the format described below. Otherwise, print `"Cannot find any word!"` on the first line and **the remaining letters** in each sequence in the format described below.

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

Input

- On the **first line**, you will receive **vowels**, **separated** by a single space (`" "`).
- On the **second line**, you will receive **consonants, separated** by a single space (`" "`).

Output

- On the first line:
  - If a word is found, print it in the format: `"Word found: {word_found}"`
  - Otherwise, print: `"Cannot find any word!"`
- On the next lines, print the remaining letters in each collection (if there are any left):
  - `"Vowels left: {vowel_one} {vowel_two} … {vowel_N}"`
  - `"Consonants left: {consonants_one} {consonants_two} … {consonants_N}"`

Constraints

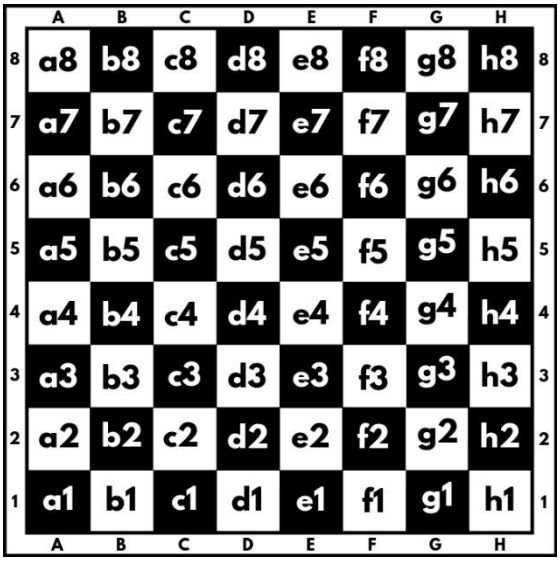
- All letters will be lowercase.
- The letter `'y'` **will always be a vowel.**
- The letter `'w'` **will always be a consonant.**

Examples

Input	Output
o e a o e a i p r s x r	Word found: rose Vowels left: o e a i Consonants left: p r
Comment	
Start by taking the first volew "o" and the last consonant "r". They are found in words "rose", "lotus", and "daffodil". Then, take "e" and "x". They are found in thr word "rose". Then, take "a" and "s". They are found in words "rose", "lotus", and "daffodil". The word "rose" is found, so we print it. Then we print the remaining letters in each sequence.	

Input	Output
a a a x r l t p p	Cannot find any word! Consonants left: x r l
u a o i u y o e p m t l	Word found: tulip Vowels left: u y o e

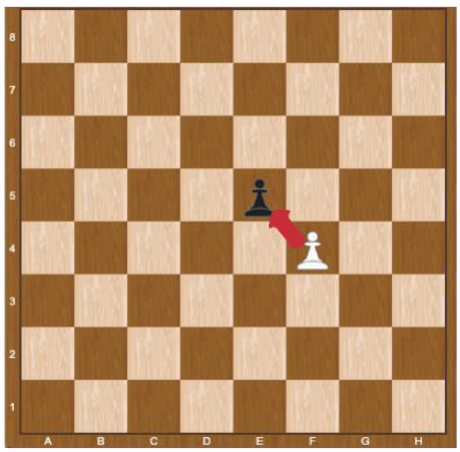
02. Pawn Wars



A chessboard has 8 rows and 8 columns. Rows, also called ranks, are marked from number 1 to 8, and columns are marked from A to H. We have a total of 64 squares. Each square is represented by a combination of letters and a number (a1, b1, c1, etc.). In this problem colors of the board will be ignored.

We will play the game with two pawns, **white (w)** and **black (b)**, where they can:

- **Only** move **forward** in a **straight line**:
  - White (**w**) moves from the 1st rank to the 8th rank direction.
  - Black (**b**) moves from 8th rank to the 1st rank direction.
- Can move only 1 square at a time.
- Can **capture** another pawn **in from of them only diagonally**:



When a pawn reaches the **last rank** (for the **white one - this is the 8th** rank, and **for the black one - this is the 1st** rank), can be **promoted** to a **queen**.

Two pawns (**w** and **b**) will be placed on two random squares of the bord. The **first move is always made by the white pawn** (**w**), then black moves (b), then white (w) again, and so on.

Some rules apply when moving paws:

- If the **two pawns interact diagonally**, the player, in turn, **must capture** the opponent's pawn. When a pawn **captures another pawn**, the **game is over**.
- If no capture is possible, the pawns **keep on moving** until **one** of them **reaches the last rank**.

Input

- On 8 **lines**, you will receive **each row with its 8 columns, each element separated by a single space:**
  - **Empty positions** are marked with **"-"**.
  - **White** pawn is marked with **"w"**
  - **Black** pawn is marked with **"b"**

Output

Print either one of the following:

- **If a pawn captures the other**, print:
  - **"Game over! {White/Black} win, capture on {square}."**
- **If a pawn reaches the last rank**, print:
  - **"Game over! {White/Black} pawn is promoted to a queen at {square}."**

Constraints

- The input will always be valid.
- The matrix will always be 8x8.
- There will be no case where two pawns are placed on the same square.
- There will be no case where two pawns are placed on the same column.
- There will be no case where black/white will be placed on the last rank.

Examples

Input	Output	Comments
- - - - - - b - w -	Game over! White pawn is promoted to a queen at b8.	We start by pushing the white pawn to **b4**, next, we push the black pawn to **g7**: - - - - - - - - - - - - - - b - - - - - - - - - - - - - - - - - - w - Then white play **b5**, black play **g6**: - b - - w - … Capturing is not possible here, so after a few more

		moves, the white pawn is promoted to a queen on **b8**.
- b - - - - - - - - w - - - - - - - - - - - - - -	Game over! White win, capture on a3.	A white pawn always start first, so it must capture the black one on **a3** in the first move: - w -

03. Springtime

Spring is the season of new beginnings. Fresh buds bloom, animals awaken and the earth seems to come to life again. Farmers and gardeners plant their seeds and temperatures slowly rise.

Write a function called **`start_spring`** which will **receive a different number of keyword arguments.**

Each **keyword** holds a **key** with a **name of the spring object** (string), and each **value** holds **its type** (string). For example, **`dahlia="flower"`**, **`shrikes="bird"`**, **`dogwood="tree"`**.

The function should **sort** the given spring objects in collections **by their type**:

- The collections **sorted by their number of elements** in descending order. If two or more **collections** have **the same number of elements** in them, return them in **ascending order** (alphabetically) by the **type's name**.
- **Each collection's elements** should be sorted in **ascending order** (alphabetically) by the **object's name**.

Note: Submit only the function in the judge system

Input

- There will be **no input**. Just parameters passed to your function.

Output

- **Return** the result, sorted as **described above** in the **format:**
  - **`"{type_one}:`**
    `-{spring_object_of_this_type_one}`
    `-{spring_object_of_this_type_two}`
    `…`
    `-{spring_object_of_this_type_N}`
    `{type_two}:`
    `…`
    `{type_N}:`
    `…`
    `-{last_spring_object_of_typeN}"`

Examples

Test Code	Output

`example_objects = {"Water Lilly": "flower",`
`"Swifts": "bird",`
`"Callery Pear": "tree",`
`"Swallows": "bird",`
`"Dahlia": "flower",`
`"Tulip": "flower",}`
`print(start_spring(**example_objects))`
	`flower:`
	`-Dahlia`
	`-Tulip`
	`-Water Lilly`
	`bird:`
	`-Swallows`
	`-Swifts`
	`tree:`
	`-Callery Pear`

`example_objects = {"Swallow": "bird",`
`"Thrushes": "bird",`
`"Woodpeckers": "bird",`
`"Swallows": "bird",`
	`bird:`
	`-Shrikes`
	`-Swallow`
	`-Swallows`

"Warblers": "bird", "Shrikes": "bird",} print(start_spring(**example_objects))	-Thrushes -Warblers -Woodpeckers
example_objects = {"Magnolia": "tree", "Swallow": "bird", "Thrushes": "bird", "Pear": "tree", "Cherries": "tree", "Shrikes": "bird", "Butterfly": "insect"} print(start_spring(**example_objects))	bird: -Shrikes -Swallow -Thrushes tree: -Cherries -Magnolia -Pear insect: -Butterfly