**Problem 1 – Second Nature**

Some unknown universal power has given you the idea to create a software program, which oversees the blooming of the legendary Edel flowers. The Edel flowers blooms when their Weiss dust is completely gone. Weiss dust is reduced by watering an Edel flower.

You will be given a **sequence of integers** – each indicating the Weiss dust in **each flower**. After that you will be given **another sequence of integers**. This time, each indicating – a bucket and the water in it.

Watering is done by picking **exactly one** bucket at a time. You must start picking from **the last received bucket** and start watering from **the first entered flower**. If the current bucket has **N**water, you **give** the **first entered flower** **N** water, **reducing**its Weiss dust by **N** – therefore **reducing** its integer value by **N**.

When an Edel flower’s **integer value** reaches **0 or less**, it blooms and **gets removed**. It is **possible** that the current flower’s value is **greater** than the current bucket’s value. **In that case** you **pick buckets** **until** you reduce its integer value to **0 or less**. If a bucket’s value is **greater** than the flower’s **current** value, you water the flower, and **add the remaining water** to the **next bucket in order**. In case there is no such, keep the remaining water in the same bucket.

If the **current bucket**’s value is **equal** to the **current flower**’s **current value**, the flower develops a **second nature**.   
By doing that, the flower becomes eternally bloomed – it cannot be affected by water. The bucket however **gets removed**.  
The watering is **continued** with the **next flower in order**.

If you **have managed** to **water all the flowers**, print the **remaining water buckets**, from the **last entered** **– to the first**.   
If you **haven’t managed** to water **all the flowers** with the given water, you must print the **remaining** **flowers**, by **order of entrance** – from the **first entered – to the last**.   
In both cases, you must also print the **second nature** flowers, if there are such, by **order of their appearance**.

**Input**

* On the **first line** of input you will receive the integers, representing the **flowers**, and the Weiss dust in them – each with each, **separated** by a **single space**.
* On the **second line**of input you will receive the integers, representing the **buckets with water** – each with each, **separated** by a **single space**.

**Output**

* On the first line of output you must print the remaining water buckets, or the remaining flowers, depending on the case you are in. Just **keep** the **orders of printing** exactly as **specified**.
* On the second line of output you must print the second nature flowers, by the order of their appearance.  
  If there are no second nature flowers, just print **“None”**on the second line of output.

**Constraints**

* All of the given numbers will be valid integers in the range [1, 500].
* It is safe to assume that there will be **NO** case in which the water is **exactly as much** as the flowers’ values, so that at the end there are no flowers and no water.
* Allowed time/memory: 100ms/16MB.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| **4 2 10 5**  **3 5 5 11 6** | **9**  **None** | **We take the first entered flower, and the last entered bucket, as we are told by the description.**    **6 – 4 = 2 – we have 2 more so we add it to the next bucket in order and it becomes 13**    **13 – 2 = 11 -**again more, so we add it to 5 and it becomes 16    **16 – 10 = 6 – we add it to the next in order**    **11 – 5 = 6 – adds its value to the last bucket**    **We’ve managed to water all the flowers so we print what we have, remaining, from the water and the second nature flowers we found, in our case - None** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **1 5 28 11 4**  **3 8 1 9 30 4 5** | **8 3**  **4** |

**Problem 2 –Nature’s Prophet**

Furion loves nature and that is why he has a beautiful square garden. He wants to plant it with magical flowers, so it can be even more beautiful. No one knows why, but he actually needs a software program to do that, that is why you’ll write one for him.

You will be given **N** and **M** – **integers**, indicating the **dimensions**of the **square garden**. The garden is **empty at the beginning**– it has no flowers. Furion wants every place for a flower to be presented with a **zero (0)** when it is **empty**. After you’ve finished creating the garden, you will start receiving two integers – **Row** and **Column**, **separated** by a **single space** – which represent the **position** at which Furion **currently plants a flower**. This happens until you receive the command **“Bloom Bloom Plow”**. When you receive that input, **all planted flowers** should **bloom**.

The flowers are **magical**. When a flower **blooms** it instantly **blooms flowers** to **all places** to its **left**, **right**, **up**, and **down**, **increasing** their **value** with **1**. Flowers can bloom **multiple times**, and **each time** the flower blooms – it becomes more and more beautiful, which means its **value increases**. The blooming of flowers is done from the **top-left** corner of the garden to the **bottom-right** one.

**Note**: If one flower blooms and affects several places, and then another flower blooms and affects one of the **first flower’s affected places**, it does **NOT** override their values with 1 again. Instead it blooms them one more time – **increasing** their value with **1**.

**Input**

* On the first line of input you will receive two integers, separated by a single space – indicating the dimensions of the garden.
* On the next several lines you will be receiving two integers separated by a single space – indicating the position at which Furion currently plants a flower.
* When you receive the input line **“Bloom Bloom Plow”**the input sequence should end.

**Output**

* The output is simple. Print the whole garden – each row of it on a new line, and each column – separated by a **single space**.

**Constraints**

* The dimensions of the matrix (**N** and **M**) will be integers in the range [3, 500].
* The integers received as position of planting a flower will **always** be inside the matrix.
* The amount of input commands will be in the range [0, **N \* M**].
* Flowers will **always** be planted on **empty**
* Allowed time/memory: 100ms/16MB

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| **5 5**  **1 1**  **3 3**  **Bloom Bloom Plow** | **0 1 0 1 0**  **1 1 1 2 1**  **0 1 0 1 0**  **1 2 1 1 1**  **0 1 0 1 0** | **The garden has 5 rows and 5 columns.**  **They are all 0 at the beginning.**  **The planted flowers are at [1, 1] and [3, 3].**  **The affected places are:**  **0 0 0 0 0**  **0 0 0 0 0**  **0 0 0 0 0**  **0 0 0 0 0**  **0 0 0 0 0**  **We receive the blooming command and we bloom the flowers from top-left to bottom-right corner of the garden.**  **First we reach the first flower, and we bloom it, increasing all affected fields’ value with 1.**  **0 1 0 0 0**  **1 1 1 1 1**  **0 1 0 0 0**  **0 1 0 0 0**  **0 1 0 0 0**  **Then we bloom the second flower, and we increase all affected fields’ (even those from the first flower) value with 1.**  **0 1 0 1 0**  **1 1 1 2 1**  **0 1 0 1 0**  **1 2 1 1 1**  **0 1 0 1 0** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **4 4**  **0 0**  **3 3**  **1 1**  **2 2**  **Bloom Bloom Plow** | **1 2 2 2**  **2 1 2 2**  **2 2 1 2**  **2 2 2 1** |

**Problem 3 – NMS**

Furion has created the NMS – Natural Messaging Service, for all the flowers. They communicate through special messages which only they understand. You want to know what the flowers are saying to each other, that’s why you’ve decided to create a software program which translates the messages.

You will be given several input lines of random, **upper case** and **lower case, English alphabet letters**. You need to find **all words** in that message. A word in the Natural language is an **increasing sequence of letters**.

Тhe message **“abc”** is a **single** word, because **“b”>“a”**and **“c”>“b”**, therefore it **IS** an increasing sequence and it is counted as a word. The message **“abca”** consists of **2** words – **“abc”** and **“a”** because **“a”<“c”**and it **breaks**the increasing sequence and **begins a new one.   
Equal letters** do **NOT** break the subsequence. The **comparison** is **case-insensitive**.

The input **ends** when you receive the command **“---NMS SEND---“**. After it you will receive a **specified delimiter**. You need to **break the whole message** into **words** and print them as a text, each separated with **the given delimiter**. The delimiter can be a line of **random ASCII characters** with **random length**.

NOTE: The **comparison** is **case-insensitive**, as specified above, therefore **“B”>“a”**, and **“A”=”a”**.

**Input**

* You will be receiving lines of input containing random English alphabet letters, until you receive the line –   
  **“---NMS SEND---“**.
* After you receive the ending command, you will receive the delimiter on the next line, as **the last line of input**.

**Output**

* As output you need to print all the words you’ve found, **separated by the given delimiter**.

**Constraints**

* All of the input lines, except the input-terminating one and the delimiter, will consist only of uppercase and lowercase English alphabet letters.
* The maximum lines of input you can receive is 1000.
* The delimiter will be a string, which can consist of any ASCII character.
* Allowed time/memory: 100ms/16MB

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Foxtrot**  **Uniform**  **Charlie Kilo**  **---NMS SEND---**  **(space)** | **Fox t r otU n i for m Ch ar l i eK ilo** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **abcdefghijklmnopqrstuvwxyz**  **ABCDEFGHIJKLMNOPQRSTUVWXYZ**  **---NMS SEND---**  **---NMS SEND---** | **abcdefghijklmnopqrstuvwxyz---NMS SEND---ABCDEFGHIJKLMNOPQRSTUVWXYZ** |

**Problem 4 – Ashes of Roses**

The time has come for the great Rosen Flame Festival. Million Roses must be prepared for the reincarnation of the Icarus, The Great Phoenix God. Different regions that participate in the Festival will grow roses for Icarus to burn, upon resurrecting. You are asked to create a software program which oversees that process.

Regions can **grow** a **specific amount** of **specifically colored** roses.

The **valid input** will come in the following format:

* **Grow <{regionName}> <{colorName}> {roseAmount}** – grows the given amount of the given color roses in the given region.

The **region names** must always **start** with a **capital** **English alphabet letter**, and **consist**only of **lowercase** letters. The **color’s name** must consist only of **English alphabet letters** and **digits**. The **rose amount** must be an **integer**.   
Any input that **does NOT** **follow**the, specified above, rules is to be treated as **invalid**, and must be **ignored**.

If you receive an input with **existent** region, you should **add** the new color and roses to it. If even **the** **color exists, increase**their **current value** with the**given one**.

The input ends when you receive the command **“Icarus, Ignite!”**. That means that the Festival is ready to begin, and the roses are all ready and set. You must **print all the regions** and their roses, but in a **specific order**.

Regions must be **ordered by amount** of roses they have, in **descending order**, as prime criteria and in **alphabetical order**, as secondary criteria. Their colors must be **ordered by amount of roses**, in **ascending order**, as prime criteria and in **alphabetical order**, as secondary criteria.

**Input**

* The input will consist only of the commands specified above.
* The input ends when you receive the command **“Icarus, Ignite!”**.

**Output**

* As output you must print information about each region and its roses in the specified order.
* The format of output is:
  + **“{region1Name}**
  + **\*--{color1Name} | {roseCount}**
  + **\*--{color2Name} | {roseCount}**
  + **…”**

**Constraints**

* The input lines can consist of any ASCII character. You must find only the valid ones.
* The input count of roses will be a valid integer in range [0, 231– 1].
* Allowed time/memory: 100ms/16MB

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Grow <Dorne> <Indigo> 2000**  **Grow <Dorne> <Purple> 5000**  **Grow <Dorne> <Purple> 1000**  **Grow <Ironislands> <Blue> 20000**  **Grow <North> <Scarlet> 1000000**  **Icarus, Ignite!** | **North**  **\*--Scarlet | 1000000**  **Ironislands**  **\*--Blue | 20000**  **Dorne**  **\*--Indigo | 2000**  **\*--Purple | 6000** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Grow <Valeofarryn> <White> 1**  **Grow <Stormlands> <White> 1**  **Grow <TheGift> <White> 1**  **Grow     <Yiti>      <White> 1**  **Grow <Valeofarryn> <Blue> 1**  **Grow <Stormlands> <Green> 1**  **Icarus, Ignite!** | **Stormlands**  **\*--Green | 1**  **\*--White | 1**  **Valeofarryn**  **\*--Blue | 1**  **\*--White | 1** |