# **Problem 1 – Cubic Artillery**

Cubic has taken charge of creating places where the, the weapons, armor and basically all the resources for the Cubic army, can be stored. He is creating massive bunkers which are capable of storing weapons. However every weapon has a different capacity, and the bunkers can only take up to a specified total capacity. That is why you have taken the task of supervising and managing the process.

You will be given **n** – an integer specifying the **bunkers’ maximum capacity**. Then you will be given input lines which will contain **English alphabet letters** and **numbers**, separated by a **single space**. The **letters** represent the **bunkers** and the **numbers** – the **weapons** and their **capacity**. Weapons must be **stored** in the bunkers. The **first entered** bunker is the **first in which weapons are getting stored**. Every weapon takes **specific capacity**, equal to its number.

When a bunker **overflows** (it **cannot contain** a given weapon due to lack of enough **free capacity)**, it passes the weapon to the **next entered bunker**. If the **next bunker cannot contain** the weapon, it passes it to the next-next, and so on till the **last entered bunker**. If there is **no bunker** that **can hold** the given weapon, **ignore** that weapon.

If there are **no other bunkers** besides the current one, you must check if current weapon can actually be contained (its **needed capacity** is **less than** the **maximum capacity specified for the bunkers**). If the weapon **can be contained**, you must **make enough free capacity** to hold that weapon, **if there isn’t already**. That is done by **removing weapons**, starting from **the first entered**, till the last. If the weapon **cannot be** contained, **ignore** the weapon.

If a bunker overflows you must **remove it**, and print it on the console, along with all of the weapons it **currently contains**. If there are no weapons, just print “**Empty**”. After you’ve removed that bunker, **the next one** becomes the **first in the order** – weapons will **first be passed to it**. If there are **no other bunkers**, you must **NOT** remove the one that overflowed.

The input sequence **ends** when you receive the command “**Bunker Revision**”.

### **Input**

* The input will come in lines of letters and digits separated by a space.
* The input ends when you receive the command “**Bunker Revision**”.

### **Output**

* For output, you must print a bunker, every time it overflows, after removing it.
* The format is the following: **{bunker}** **-> {weapon1}, {weapossn2}…**
* Where {bunker} is the letter that corresponds to that bunker, and the weapons are the numbers.
* In case a bunker has no weapons, just print “**Empty**”.

### **Constraints**

* The bunkers will only be English alphabet letters.
* Each bunker’s letter will always be unique.
* The integer n will be In the range [0, 500].
* The weapons will always be valid integers in the range [0, 500].
* Allowed time/memory: 100ms/16MB.

### **Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **60**  **a 20 20 b 20 1**  **Bunker Revision** | **a -> 20, 20, 20** |
| **Comment** | |
| **“a” is the first entered bunker. Then we receive the weapons 20 and 20 which are passed to “a”. Then we get the bunker “b”. Then again we receive a weapon 20. “a” still has enough capacity to hold the weapon so we store it there. Then we get the weapon 1. “a” has capacity 60/60 – it overflows, so we pass the weapon to the next bunker. We find “b” and we pass the weapon to “b”. “a” is then removed and printed on the console. “b” becomes the first bunker now.** | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **50**  **b 10 15 20 30**  **c 100**  **a 65**  **Bunker Revision** | **b -> 20, 30**  **c -> Empty** |

# **Problem 2 – Cubic’s Rube**

Several years ago, while Cubic was researching a new quantum technology, to design a weapon he can use against the Spherical Race, he discovered a magical sub-dimension which stands in the cross-road of all other dimensions. The dimension was completely empty but it had a perfect cubic form and Cubic liked that a lot, so he named it after himself – The Cubic’s Rube.

Cubic noticed that the Rube gets frequently bombarded with particles which fill it, so he decided to research its volume to see how it reacts with particles. He asked for help from The Great Cubical Army, and, guess what? They sent you.

You will be given **n** – an integer specifying the 3 dimensions of The Rube. Only 1 integer is used for all 3 dimensions because The Rube is a perfect cube. After that you will start receiving lines containing 4 integers separated by a single space. The **first three integers** will represent a **point** in the 3D space, and **the last integer** will represent the particles. You must bombard that cell at that point, **if there is such cell**, with the **given particles**, adding to it – the value corresponding to the given 4th integer. Note that the properties specified above apply **only** for cells **INSIDE** The Rube. Also, there will be **NO** cell that is **hit** more than **1 times** by particles.

The input ends when you receive the command “**Analyze**”. Then you must print the sum of all the cells in The Rube, along with the number of cells which’s value has not been changed. See the output section for more info.

### **Input**

* The first line of input will hold an integer **n**.
* After that you will begin receiving lines of input containing 4 integers separated by a space.
* The input ends when you receive the command “**Analyze**”.

### **Output**

* The output consists of two lines.
* On the first line print the sum of all the cells in the Rube.
* On the second line print the amount of cells which’s value has not been changed.

### **Constraints**

* The dimensions of the matrix – n will be in the range [0, 25].
* The integers from the input lines will be in the range [-231 + 1, 231 – 1].
* Allowed time/memory: 100ms/16MB.

### **Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **4**  **2 2 2 2**  **Analyze** | **2**  **63** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **5**  **3 2 3 10**  **-1 -1 -1 0**  **1 4 0 20**  **2 2 2 5**  **Analyze** | **35**  **122** |

# **Problem 3 – Cubic’s Messages**

Cubic is a veteran soldier from The Great Cubic Army. He has even participated in the Spherical Invasion as a Sergeant First Class. As a veteran, Cubic has some personal security issues – he communicates only trough text messages and sends them in a specific encrypted way, which you must decrypt in order to understand what he is saying.

You will begin receiving lines of input, which will consist of random ASCII characters – Cubic’s encrypted lines. After each line you will receive a number – the length of the message he sent. Cubic might send false messages, in an act to confuse his “enemies”. You must capture only the messages that follow a certain format.

According to that format the **valid** messages:

* Consist of **m** characters, where **m** is the integer entered after each encrypted line.
* Has only digits before itself in the encrypted line
* Consists only of English alphabet letters
* Has no English alphabet letters after itself in the encrypted line

**Any** message that **does not follow** the, specified above, rules, is **invalid**, and you must **ignore it**.

After you find **all valid** messages, you need to find their **verification code**. Every message has its own verification code, which Cubic gives in order to verify the message. **Take all the digits before the message** and all the digits **after the message** and consider them as **indexes**. If they are **valid existing** indexes **in the message**, **form a string** with those indexes **taking characters from the message**. If an index is **nonexistent**, put a **space** there. The string you form up is the verification code for the current message.

### **Input**

* The input will always come in the form of 2 lines, except when it is the line terminating the input sequence.
* The first input line will contain random ASCII characters, and the second – a number.
* When the line “**Over!**” is entered, the input sequence ends.

### **Output**

* The output is simple. You must print all the valid messages you’ve found, each on a new line, and their verification codes, if they have such.
* The format of output is “**{message} == {verificationCode}**”.

### **Constraints**

* The input lines can consist of **ANY ASCII** character.
* There will be **NO** such cases as an encrypted message without a number before it.
* The number will be a valid integer in the range [0, 100].
* Allowed time/memory: 100ms/16MB

### **Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **1234test4321**  **4**  **0000oooo0000**  **4**  **Over!** | **test == est tse**  **oooo == oooooooo** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **1wat!**  **3**  **#23asd33**  **3**  **333asd3a**  **3**  **100dun2**  **3**  **Over!** | **wat == a**  **dun == uddn** |

# **Problem 4 – Cubic Assault**

After countless of hours of preparation – artillery storage, quantum research, and planning trough encoded messages, time has finally come for a war with the Spherical Race. Cubic is on the front lines devastating the enemy forces. Someone, however, must give statistics to Cubic about the count of enemies on each front. You are best for this job.

You will be given as input lines containing, a region, the type of the soldiers at that region and their amount. You must **store statistics** about the **amount of meteors** Cubic needs to defeat in **every region**. Note that if at one region 1 000 000 Green Meteors gather, they **combine** into 1 Red Meteor. By the same logic, 1 000 000 Red Meteors get **combined** into 1 Black Meteor. Note also, that you might receive **several input lines** with information about **1 region**. In that case just **update that region’s statistics**. Multiple values to one type **increase** that type’s value each time.

The input data will come in the following format **{regionName} -> {meteorType} -> {count}.**

When you receive the command “**Count em all**”, you must **end** the input sequence. You must print all the regions ordered by the **amount of Black Meteors** – descending, then by the **length of their names** – ascending, and lastly **alphabetically**. For every region you must print how many green, red and black meteors there. Order the printing of the types by **amount of defeated units** in them – descending, and if two are with the same value, order them **alphabetically**.

### **Input**

* As input you will receive random amount of input lines containing information about Cubic’s statistics.
* The input ends when you receive the command “Count em all”.

### **Output**

* The output is simple. You must print each region and the statistics about the 3 types of meteors in it.
* The format of output is :

**{regionName}**

**-> {firstType} : {firstTypeCount}**

**-> {secondType} : {secondTypeCount}**

**-> {thirdType} : {thirdTypeCount}**

* The order of each type depends on its count as specified above. All data must be ordered correctly.

### **Constraints**

* The input numbers will be valid integers in the range [-231 + 1, 231 – 1].
* The input will always be in the format specified above. There is no need to check it explicitly.
* Allowed time/memory: 100ms/16MB

### **Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Cubica -> Black -> 1**  **Cubica -> Red -> 1000**  **Spherica -> Green -> 1000000**  **Count em all** | **Cubica**  **-> Red : 1000**  **-> Black : 1**  **-> Green : 0**  **Spherica**  **-> Red : 1**  **-> Black : 0**  **-> Green : 0** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Triangula Canyon -> Black -> 100**  **Diagonalica -> Red -> 999999**  **Ellipsetica -> Red -> 100000000**  **Diagonalica -> Black -> 99**  **Diagonalica -> Green -> 1000000**  **Count em all** | **Diagonalica**  **-> Black : 100**  **-> Green : 0**  **-> Red : 0**  **Ellipsetica**  **-> Black : 100**  **-> Green : 0**  **-> Red : 0**  **Triangula Canyon**  **-> Black : 100**  **-> Green : 0**  **-> Red : 0** |