Data Structure – Session 4

Problem statement:

You have n elements, find all unique numbers and return them!

Ex: {1, 5, 1, 4, 5, 2}

Answer: {1, 5, 4, 2}

This problem can be solved using many methods:

1-Using Array

```
int x[]={1, 5, 1, 4, 5, 2}
int count=0;
for (int i = 0; i < x.length; i++)
{
            found=false;
            for (int j = 0; j < i;
j++) {
               if(x[j]==x[i]){
                   found=true;
                   break;
               }
            }
            if(!found)
                count++;
        }
```

```
int res[]= new int[count];
        int index=0;
      for (int i = 0; i < x.length;
i++) {
            found=false;
            for (int j =0; j < i;
j++) {
               if(x[j]==x[i]){
                   found=true;
                   break;
               }
            }
            if(!found)
                res[index++]=x[i];
        }
```

2-Using LinkedList

```
x[]= {1,5,1,4,5};
       LinkedList<Integer> list =
new LinkedList<Integer>();
        for (int i = 0; i <
x.length; i++) {

if(!list.contains(x[i]))
                list.add(x[i]);
        }
```

3-Using ArrayList

```
int x[]= {1,5,1,4,5};
       ArrayList<Integer> list = new
ArrayList<Integer>();
        for (int i = 0; i < x.length;
i++) {
            if(!list.contains(x[i]))
                list.add(x[i]);
        }
```

Problem:

The process of "**breaking**" an integer is defined as summing the squares of its digits.For example, the result of breaking the integer **125** is $(\mathbf{1^2 + 2^2 + 5^2}) = \mathbf{30}$. An integer **N** is **happy** if after "**breaking**" it repeatedly the result reaches 1. If the result never reaches 1 no matter how many times the "**breaking**" is repeated, then N is not a happy number.

TASK

Write a program that given an integer N, determines whether it is a happy number or not.

CONSTRAINTS

2 ≤ N ≤ 2,147,483,647

Input

- A single line containing a single integer **N**.

Output

- A single line containing a single integer **T** which is the number of times the process had to be done to determine that N is happy, or **-1** if **N** is not happy.

Example

Input:
19
Output:
4
1) 19 : $1^2 + 9^2 = 82$ 2) 82 : $82 + 2^2 = 68$
3) 68 : $6^2 + 8^2 = 10$ 4) 100 : $1^2 + 0^2 + 0^2 = 1$

The solution is 4 because we discovered that the integer 19 is happy after we repeated the process 4
times.
Example

Input:
204
Output:
-1

204 –> 20 –> 4 –> 16 –> 37 –> 58 –> 89 –> 145 –> 42 –> 20 –> 4 –> 16 –> 37 –> 58 –> 89 –> 145 ……..
204 is not a happy number because after breaking it several times the results start repeating so we
can deduce that if we continue breaking it, the result will never reach 1.

HashSet

HashSet is a unique data structure, which means it cannot contain duplicate elements

This class makes no guarantees as to the iteration order of the set

```
// Defining HashSet
        HashSet<Integer> hs = new HashSet<Integer>();

        // Adding elements
        hs.add(5); // true
        hs.add(1); // true
        hs.add(7); // true
        hs.add(5); // false (repeated)

        // Printing
        System.out.println(hs); // Order in hashet is NOT preserved, and
does NOT matter


        // removing elements
        hs.remove(5);// true
        hs.remove(3);// false

        System.out.println(hs); //[1,7]

        // check if it contains an element
      hs.contains(7);// true
      hs.contains(5);// false

      // Iterating :
      // first method : iterator
      Iterator<Integer> it = hs.iterator();
      while(it.hasNext())
            System.out.println(it.next());

      // second method : enhanced for
      for(int x:hs){
          System.out.println(x);
      }
```

HashMap

It's like a dictionary, which means every element has a key and a value

A key is always unique, and the value needs not to be

```
 // Defining HashMap
        HashMap<String,Integer> hm = new HashMap<String,Integer>();

        // Adding elements
        hm.put("Ahmed", 100);// null
        hm.put("Mona",15); // null
        hm.put("Ahmed",70); // 100 (old value for "Ahmed" )
        hm.put("mona",20); // null (new element)


        // Printing
        System.out.println(hm); // Order in HashMap is NOT preserved, and
does NOT matter
        // {Ahmed=70, mona=20, Mona=15}

        // removing elements
        hm.remove("Mona");// true
        hm.remove("Karim");// false

        System.out.println(hm); // {Ahmed=70, mona=20}


        // check if it contains an element
      hm.containsKey("Ahmed");// true
      hm.containsValue(100);// false

      // Iterating :
          for (Map.Entry<String, Integer> entry : hm.entrySet()) {
          System.out.println("Key : " + entry.getKey());
                System.out.println("Value : "+ entry.getValue());
     }
```

we remove by key

TreeMap

It's a hashmap that preserves the order of elements according to keys!

```
    // Defining TreeMap
        TreeMap<Integer,String> ht = new TreeMap<Integer,String>();

        // Adding elements
        ht.put(5, "R"); // null
        ht.put(15, "T"); // null
        ht.put(3, "M"); // null
        ht.put(5, "S"); // "R" (old value for s)


        // Printing
        System.out.println(ht); // Order in TreeMap is preserved!
        // {3=M, 5=S, 15=T}

        // removing elements
        ht.remove(3);// true
        ht.remove(10);// false

        System.out.println(ht); // {5=S, 15=T}


        // check if it contains an element
       ht.containsKey(5);// true
       ht.containsValue("M");// false

      // Iterating :
          for (Map.Entry<Integer, String> entry : ht.entrySet()) {
          System.out.println("Key : " + entry.getKey());
                System.out.println("Value : "+ entry.getValue());
     }


    }
}
```

10420 - List of Conquests

Time limit: 3.000 seconds

In Act I, Leporello is telling Donna Elvira about his master's long list of conquests:

> ``This is the list of the beauties my master has loved, a list I've made out myself: take a look, read it with me. In Italy six hundred and forty, in Germany two hundred and thirty-one, a hundred in France, ninety-one in Turkey; but in Spain already a thousand and three! Among them are country girls, waiting-maids, city beauties; there are countesses, baronesses, marchionesses, princesses: women of every rank, of every size, of every age.'' (*Madamina, il catalogo è questo*)

As Leporello records all the ``beauties'' Don Giovanni ``loved'' in chronological order, it is very troublesome for him to present his master's conquest to others because he needs to count the number of ``beauties'' by their nationality each time. You are to help Leporello to count.

Input

The input consists of at most **2000** lines, but the first. The first line contains a number **n**, indicating that there will be **n** more lines. Each following line, with at most **75** characters, contains a country (the first word) and the name of a woman (the rest of the words in the line) Giovanni loved. You may assume that the name of all countries consist of only one word.

Output

The output consists of lines in alphabetical order. Each line starts with the name of a country, followed by the total number of women Giovanni loved in that country, separated by a space.

Sample Input

```
3
Spain Donna Elvira
England Jane Doe
Spain Donna Anna
```

Sample Output

```
England 1
Spain 2
```