Sample Output 0

```
YES 1
YES 9
YES 99
NO
NO
NO
NO
```

Explanation 0

The first three numbers are beautiful (see the diagram above). The remaining numbers are not beautiful:

- For $s = 101103$, all possible splits violate the first and/or second conditions.
- For $s = 010203$, it starts with a zero so all possible splits violate the second condition.
- For $s = 13$, the only possible split is $\{1, 3\}$, which violates the first condition.
- For $s = 1$, there are no possible splits because s only has one digit.

Sample Input 1

```
4
99910001001
7891011
9899100
999100010001
```

Sample Output 1


```
            }
            }
        System.out.println(isValid ? "YES" + subString : "NO");

        }


    }

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader
(System.in));

        int q = Integer.parseInt(bufferedReader.readLine().trim());

        for (int qItr = 0; qItr < q; qItr++) {
            String s = bufferedReader.readLine();

            Result.separateNumbers(s);
        }

        bufferedReader.close();
    }
}
```

Function Description

Complete the separateNumbers function in the editor below.

separateNumbers has the following parameter:

- s: an integer value represented as a string

Prints

- string: Print a string as described above. Return nothing.

Input Format

The first line contains an integer q, the number of strings to evaluate.

Each of the next q lines contains an integer string s to query.

Constraints

- $1 \leq q \leq 10$
- $1 \leq |s| \leq 32$
- $s[i] \in [0 - 9]$

Sample Input 0

```
7
1234
91011
99100
101103
010203
13
1
```

Change Theme Language Java 7

```
            {
                validString+=Long.toString(++num);
            }
            if(s.equals(validString))
            {
                isValid = true;
                break;
            }
            }
        System.out.println(isValid ? "YES" + subString : "NO");

        }


    }

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader
(System.in));

        int q = Integer.parseInt(bufferedReader.readLine().trim());

        for (int qItr = 0; qItr < q; qItr++) {
            String s = bufferedReader.readLine();
```

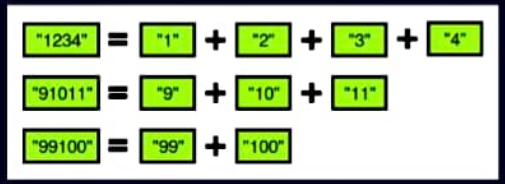
Line: 58 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

A numeric string, s, is beautiful if it can be split into a sequence of two or more positive integers, $a[1], a[2], \ldots, a[n]$, satisfying the following conditions:

1. $a[i] - a[i-1] = 1$ for any $1 < i \leq n$ (i.e., each element in the sequence is 1 more than the previous element).
2. No $a[i]$ contains a leading zero. For example, we can split $s = 10203$ into the sequence $\{1, 02, 03\}$, but it is not beautiful because 02 and 03 have leading zeroes.
3. The contents of the sequence cannot be rearranged. For example, we can split $s = 312$ into the sequence $\{3, 1, 2\}$, but it is not beautiful because it breaks our first constraint (i.e., $1 - 3 \neq 1$).

The diagram below depicts some beautiful strings:



Perform q queries where each query consists of some integer string s. For each query, print whether or not the string is beautiful on a new line. If it is beautiful, print `YES x`, where x is the first number of the increasing sequence. If there are multiple such values of x, choose the smallest. Otherwise, print `NO`.

Function Description

Change Theme Language Java 7

```java
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;

class Result {

    /*
     * Complete the 'separateNumbers' function below.
     *
     * The function accepts STRING s as parameter.
     */

    public static void separateNumbers(String s) {
    // Write your code here
        String subString = "";
        boolean isValid = false;
        for(int i=1;i<=s.length()/2;i++)
        {
            subString = s.substring(0,i);
            Long num = Long.parseLong(subString);
            String validString = subString;
            while(validString.length() < s.length());
```

Line: 58 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Sample Output 1

```
5
```

Explanation 1

The special triplets are $(1, 2, 1), (1, 2, 2), (1, 3, 1), (1, 3, 2), (1, 3, 3)$

Sample Input 2

```
4 3 4
1 3 5 7
5 7 9
7 9 11 13
```

Sample Output 2

```
12
```

Explanation 2

The special triplets are

$(1, 7, 7), (1, 9, 7), (1, 9, 9), (3, 7, 7), (3, 9, 7), (3, 9, 9), (5, 7, 7), (5, 9, 7), (5, 9, 9), (7, 7$

.


```java
    static int getValidIndex(int[] distinctA, int key) {
        int low = 0;
        int high = distinctA.length - 1;
        int count = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (distinctA[mid] <= key) {
                count = mid;
                low = mid + 1;
            }
                else
                    high = mid - 1;

            }
            return count;
        }


    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) throws IOException {
        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.
```

Sample Input 0

```
3 2 3
1 3 5
2 3
1 2 3
```

Sample Output 0

```
8
```

Explanation 0

The special triplets are

$(1,2,1),(1,2,2),(1,3,1),(1,3,2),(1,3,3),(3,3,1),(3,3,2),(3,3,3)$.

Sample Input 1

```
3 3 3
1 4 5
2 3 3
1 2 3
```

Sample Output 1

```
5
```



```
            distinctTripleCount += c1 * c3;
        }
        return distinctTripleCount;


    }

    private static int[] removeDuplicates(int[] a) {
        Set<Integer> set = new HashSet<>();
        for (int item : a) {
            set.add(item);
        }
        int len = set.size();

        int result[] = new int[len];
        int i = 0;
        for (int item : set) {
            result[i++] = item;
        }
        return result;

    }
```

Given 3 arrays a, b, c of different sizes, find the number of distinct triplets (p, q, r) where p is an element of a, written as $p \in a$, $q \in b$, and $r \in c$, satisfying the criteria: $p \leq q$ *and* $q \geq r$.

For example, given $a = [3, 5, 7]$, $b = [3, 6]$, and $c = [4, 6, 9]$, we find four distinct triplets: $(3, 6, 4)$, $(3, 6, 6)$, $(5, 6, 4)$, $(5, 6, 6)$.

Function Description

Complete the triplets function in the editor below. It must return the number of distinct triplets that can be formed from the given arrays.

triplets has the following parameter(s):

- a, b, c: three arrays of integers .

Input Format

The first line contains 3 integers $lena$, $lenb$, *and* $lenc$, the sizes of the three arrays. The next 3 lines contain space-separated integers numbering $lena$, $lenb$, *and* $lenc$ respectively.

Constraints

$1 \leq lena, lenb, lenc \leq 10^5$

$1 \leq$ all elements in $a, b, c \leq 10^9$

Output Format

Print an integer representing the number of distinct triplets.

Change Theme Language Java 7

```java
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;

public class Solution {

    // Complete the triplets function below.
        static long triplets(int[] a, int[] b, int[] c) {
        long distinctTripleCount = 0;
        int[] distinctA = removeDuplicates(a);
        int[] distinctB = removeDuplicates(b);
        int[] distinctC = removeDuplicates(c);
        Arrays.sort(distinctA);
        Arrays.sort(distinctB);
        Arrays.sort(distinctC);

        for(int q : distinctB) {
            long c1 = getValidIndex(distinctA, q) + 1;
            long c3 = getValidIndex(distinctC, q) + 1;
```

Line: 119 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Congratulations

You solved this challenge. Would you like to challenge your friends?

- Test case 0
- Test case 1
- Test case 2
- Test case 3
- Test case 4
- Test case 5
- Test case 6

Compiler Message

Success

Input (stdin) Download

```
3 2 3
1 3 5
2 3
1 2 3
```

Expected Output Download

```
8
```

Congratulations

You solved this challenge. Would you like to challenge your friends?

Next Challenge

- Test case 0
- Test case 1
- Test case 2
- Test case 3
- Test case 4
- Test case 5
- Test case 6

Compiler Message

Success

Input (stdin) Download

```
2
0
1
```

Expected Output Download

```
1
2
```