2nd Programming Contest League 2025 - University of Birjand

Date: May 2025

Committee Members

Scientific Team
Technical Team
Executive Team

Problem A : Balanced Attendance

In the first round of the programming league, the event coordinator needs to evaluate whether the attendance behavior of the participants has been balanced. Since the number of participants is large, the coordinator uses the automated hall-management system to analyze the attendance logs.

Each participant is assigned an attendance code — a string consisting of the characters:

- **P** → Present
- **A** → Absent
- **L** → Late

The system considers an attendance record **balanced** if both of the following conditions hold:

- The number of **P** (Present) characters is at least the number of **A** (Absent) characters.
- The longest contiguous sequence of **L** (Late) characters is at most 2. (That is, the substring `LLL` must not appear.)

The coordinator hands you a batch of attendance reports and asks you to check them quickly.

Your task: For each attendance string, determine whether it is **Balanced** or **Unbalanced**.

Input

The first line contains an integer t — the number of attendance reports. $(1 \leq t \leq 2000)$

Each of the next t lines contains a string s consisting only of the characters `P`, `A`, and `L`. $(1 \leq |s| \leq 2000)$

Output

For each report, print:

- `Balanced`, if both conditions hold.
- `Unbalanced`, otherwise.

Example

Standard Input	Standard Output
3 PPALL APPLL PLLLP	Balanced Balanced Unbalanced

Standard Input	Standard Output
2 PPP AALL	Balanced Unbalanced

Problem B : Divisibility Moves

You are given two positive integers a and b. In one move, you can increase a by 1 (that is, replace a with $a + 1$).

Your task is to determine the minimum number of moves needed to make a divisible by b.

It is possible that you need to make 0 moves if a is already divisible by b.

You have to answer t independent test cases.

Input

The first line contains one integer t ($1 \le t \le 10^4$) — the number of test cases.

Each of the following t lines contains two integers a and b ($1 \le a, b \le 10^9$).

Output

For each test case, print a single integer — the minimum number of moves required to make a divisible by b.

Example

Standard Input	Standard Output
5 10 4 13 9 100 13 123 456 92 46	2 5 4 333 0

Problem C : Sum Pairs

It is late at night in the university computer lab. The annual University Programming Cup is just around the corner, and two teammates — Sara and Nima — are still practicing.

They have been solving hard algorithmic problems for hours, but their brains are tired, so Sara suggests taking a short break.

On the whiteboard in front of them, dozens of random numbers are scribbled — leftovers from an earlier problem. Sara points at the list and says:

"Hey, let's play a small game instead of solving those tree DP nightmares."

Nima raises an eyebrow. *"A game?"*

"Yes," she grins. *"I'll pick a target number K. You'll have to find how many pairs of numbers from this list add up exactly to that target."*

Nima laughs. *"That's too easy! It's just addition!"*

"Maybe," says Sara. *"But you only have one minute — and you can't count the same pair twice. (i, j) and (j, i) are the same. Also, you can't pair a number with itself unless it appears twice."*

Now Nima looks intrigued. He grabs his notebook and starts writing fast, determined to beat Sara's brain teaser with some clever counting trick.

But as the numbers grow larger, the game becomes less trivial. Sara keeps increasing the size of the list — 10, 100, 10 000 numbers — until even Nima realizes that this game is not just simple addition. Now they both need your help.

Your task is to help them find how many perfect pairs exist.

Input

The first line contains two integers N and K — the number of numbers on the whiteboard and the target sum.

The second line contains N space-separated integers $A_1, A_2, \ldots, A_N$, representing the numbers Sara wrote.

Output

Print a single integer — the number of distinct pairs (i, j) such that $A[i] + A[j] = K$ and $i < j$.

Constraints

$$2 \leq N \leq 10^5$$
$$1 \leq A[i] \leq 10^9$$
$$1 \leq K \leq 2 \times 10^9$$

Example

Standard Input	Standard Output
5 7 1 2 3 4 5	2

Standard Input	Standard Output
4 10 5 5 5 5	6

Problem D : Lost Frequencies

Deep in the middle of a vast desert, an isolated research station runs a long–term experiment on a mysterious species of beetles that communicate using ultrasonic frequencies. Each beetle is assigned a unique **non-negative frequency ID**, allowing the monitoring system to track individuals in the wild.

For years, the central computer at the station has been quietly recording these frequencies. Every time a beetle is detected, its frequency ID is stored in a log. But one night, a violent sandstorm hits the observatory. Power flickers, antennas bend under the wind, and the main recording unit is forced to reboot.

When the system finally comes back online, the researchers realize that the frequency log has been corrupted:

- Some recorded values have turned into **negative numbers**, which cannot possibly be valid IDs.
- Some entries have become **enormous integers**, with up to 100 digits, likely the result of overflow or bit flips.
- Only values that form valid **non-negative integers** can still be trusted as real frequency IDs.

Despite the damage, the experiment must continue. Whenever a **new** beetle is detected, the system must assign it the **smallest non-negative frequency ID** that is **not currently used** by any known beetle. This ensures that IDs remain compact and easy to manage.

The old allocator module, which used to handle this task, was destroyed in the reboot. The researchers now need you to re-implement this logic using the corrupted log they managed to recover.

Your job is to read the list of recorded values, filter out the invalid ones, and determine the smallest non-negative integer that does **not** appear among the valid frequency IDs. This is the ID that will be assigned to the next newly detected beetle.

Input

The input consists of the following lines:

- The first line contains an integer N, the number of recorded values.
- Each of the next N lines contains an integer X_i, written in decimal form. Each X_i may be:
  - a negative integer,
  - zero,
  - a positive integer,
  - a very large positive integer (up to 100 digits).

An entry X_i is considered a **valid frequency** if and only if it represents a non-negative integer. All other values (negative numbers or any value outside the allowed range) must

be ignored when determining which frequencies are already used.

Output

Print a single integer — the **smallest non-negative integer** that does not appear among the valid recorded frequencies.

Constraints

$$0 \leq N \leq 10^6$$

Each integer X_i is given as a decimal string of fewer than 100 digits. The input is well-formed: each line represents an integer in standard decimal notation, possibly with a leading minus sign.

Example

Standard Input	Standard Output
5 1 -1 3 0 10	2

In this example, the valid non-negative frequencies are $\{0, 1, 3, 10\}$. The smallest non-negative integer that does not appear in this set is 2.