**Slot Machine 2.0**

A slot machine has multiple wheels that are spun n times. In each spin, each wheel may have multiple stops from 1 to 9 and shows one random number on the machine's dashboard.

Given the number of spins n, determine the minimum number of stops on each wheel to produce the numbers displayed on the dashboard for each spin. Then, calculate the total stops.

****Example****

*n = 4*

*spins[] = [ '712', '246', '365', '312' ]*

the spins on a slot machine with *3* wheels are recorded as an array, *history*:

7 1 2

2 4 6

3 6 5

3 1 2

One wheel needs to have at least *7* stops to produce the numbers displayed on the dashboard for *1st* spin. Since *7* is the highest value in any row, remove the highest value from each of the rows:

1 2

2 4

3 5

1 2

Now the highest value is *5*, so another wheel must have *5* stops to produce the numbers displayed on the dashboard for 3*rd* spin. Using the same logic, the final wheel needs *3* stops. Total stops are *7 + 5 + 3 = 15*.

****Function Description****

Complete the function *slotSpins* in the editor below.

slotWheels has the following parameter(s):

*string spins [n]:*  an array of equal length strings of digits spun

Returns:

*int:* an integer that represents the sum of the minimum number of stops on all of the wheels.

****Constraints:****

* *1 ≤ |spins| ≤ 50*
* *1 ≤ spins[i] ≤ 50*
* All *spins[i]* in a test case will be of equal length.
* All characters in each *spins[i] ∈ [0-9]*

Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *history*.

Each of the next *n* lines contains a string *spins[i]* where *0 ≤ i < n*.

Sample Case 0

****Sample Input 0****

STDIN    Function Parameters

-----    -------------------

4 →  *spins*[] Size n = 4

137 →  *spins*[] = ['137', '364', '115', '724']

364

115

724

****Sample Output 0****

14

****Explanation 0****

the spins on a slot machine with *4* wheels are recorded as an array, *spins*:

1 3 7

3 6 4

1 1 5

7 2 4

One wheel needs to have at least *7* stops to produce the numbers displayed on the dashboard for *1st* spin and *4th*  spin. Since *7* is the highest value in any row, remove the highest value from each of the rows:

1 3

3 4

1 1

2 4

One wheel needs to have at least *4* stops to produce the numbers displayed on the dashboard for 2*nd* spin and *4th*  spin. Since *4* is the highest value in any row, remove the highest value from each of the rows. Using the same logic, the final wheel needs *3* stops. Total stops are *7 + 4 + 3 = 14*.

**Sample Case 1**

****Sample Input 1****

STDIN    Function Parameters

-----    -------------------

4     →  *spins*[] Size = 4

1112 →  *spins*[] = ['1112', '1111', '1211', '1111']

1111

1211

1111

****Sample Output 1****

5

****Explanation 1****

the spins on a slot machine with *4* wheels are recorded as an array, *spins*:

1 1 1 2

1 1 1 1

1 2 1 1

1 1 1 1

One wheel needs to have at least *2* stops to produce the numbers displayed on the dashboard for *1st* spin and 3*th*  spin. Since *2* is the highest value in any row, remove the highest value from each of the rows:

1 1 1

1 1 1

1 1 1

1 1 1

Using the same logic, 3 wheels need to have 1 stop each. Total stops are 2*+ 1 + 1 + 1 = 5*.

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'slotWheels' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts STRING\_ARRAY history as parameter.

\*/

public static int slotWheels(List<String> history) {

// Write your code here

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

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

List<String> history = IntStream.range(0, historyCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.collect(toList());

int result = Result.slotWheels(history);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*