# Circular Logic

**Program Name: logic.java          Input File: logic.in**

Imagine that you have a computer monitor that can only display five lines of text. Also imagine that your only running program is a counting program that begins at 1 and counts upward, one number at a time. It prints the numbers on your monitor, one per line, wrapping back to the top line after writing the bottom one.

The only interaction you have with this program is the ability to hit the space bar to have it wrap early (i.e., before it has written the last line). This is called a 'reset'.

Each column of number below indicates what your computer monitor would show at each step as your program counted from 1 to 21 without any resets:

```
 1  1  1  1  1  6  6  6  6  6 11 11 11 11 11 16 16 16 16 16 21
    2  2  2  2  2  7  7  7  7  7 12 12 12 12 12 17 17 17 17 17
       3  3  3  3  3  8  8  8  8  8 13 13 13 13 13 18 18 18 18
          4  4  4  4  4  9  9  9  9  9 14 14 14 14 14 19 19 19
             5  5  5  5  5 10 10 10 10 10 15 15 15 15 15 20 20
```

However, let's say that you decided to perform a reset after the 7, 12, and 18 were printed.

```
 1  1  1  1  1  6  6  8  8  8  8  8 13 13 13 13 13 18 19 19 19
    2  2  2  2  2  7  7  9  9  9  9  9 14 14 14 14 14 14 20 20
       3  3  3  3  3  3  3 10 10 10 10 10 15 15 15 15 15 15 21
          4  4  4  4  4  4  4 11 11 11 11 11 16 16 16 16 16 16
             5  5  5  5  5  5  5 12 12 12 12 12 17 17 17 17 17
```

As you can see, the first reset caused the 8 to start over in the first row instead of the third. The second reset had no real effect since the 13 was going to be in the first row anyway. And the third reset caused the 19 to be printed on the first row instead of the second. The final result: 19, 20, 21, 16, 17, is considerably different from that in the example without any resets.

Write a program that can determine, given the final output on the monitor, the minimum number of resets needed to generate the final pattern.

## Input

The first line of input will contain a single integer *n*, indicating the number of data sets. The remainder of the input consists of those *n* data sets. Each data set will consist of five lines, containing the final display of the monitor, as illustrated in the above examples.

Notes:

All inputs are valid (i.e., could be generated by performing the correct resets).
None of the final displays in the input will contain numbers larger than 25.
Resets will not occur until after the number 5 is displayed.
At most 10 resets will occur in any one data set.

## Output

For each data set in the input display a single line containing the minimum number of resets needed to produce the final display.

**Example Input File**

```
6
21
17
18
19
20
19
20
21
16
17
6
2
3
4
5
7
2
3
4
5
9
10
3
4
5
11
2
3
4
5
```

**Example Output To Screen**

```
0
2
0
1
2
5
```