Shifting gears

On a sunny autumn morning I was riding my old mountain bike with friends -who are total bike nerds- and noticed their bikes had a single chainring (front gear).

They explained to me that an old-school 3x7 gears configuration adds extra weight for little gain, since many of the chainring/sprocket combinations will yield to equivalent gear ratios. A wider range in the cassette (rear gears) and just one chainring in the crankset is more efficient.

The question is as follows: given the number of teeth on each of the front and rear cogs, we would like to know how many *unique gear ratios* are available.

Input description

The first line will be an integer `N` specifiying the number of cases to solve.

After that, `N` lines containing a case each will follow.

Each case has the following format:

```
C S [c_i...] [s_j...]
```

Where `C` is the number of chainrings, `S` is the number of sprockets, followed by a list of `C` chainring's teeth `c_i` and a list of `S` sproket's teeth `c_j`.

Input constraints

- `1 <= N <= 1000`
- `1 <= C <= 10`
- `1 <= S <= 100`
- `1 <= c_i < 2**63`
- `1 <= s_j < 2**63`

There will be exactly `1 + N` lines.

Each token will be separated by a single whitespace character.

No extraneous characters are to be expected in the input, the number of tokens per line will be `2 + C + S`.

Output description

The output should contain one line per case, containing only the number of unique gear ratios for the given input.

Sample

Input

```
3
1 1 40 15
2 3 53 43 11 17 23
3 6 48 38 28 14 16 19 21 24 28
```

Explanation:

- `3` cases
- First case has `C = 1`, `S = 1`, `c_1 = 40` and `s_1 = 15`
- Second case has `C = 2`, `S = 3`, `c = [53, 43]`, and `s = [11, 17, 23]`
- Third case has `C = 3`, `S = 6`, `c = [48, 38, 28]`, and `s = [14, 16, 19, 21, 24, 28]`

Output

```
1
6
16
```

Explanation:

- Case 1 is a "fixie" bike with just 1 available gear.
- Case 2 has 2 chainrings and 3 sprockets, all of them have a prime number of teeth, so it has 6 unique gear ratios.
- Case 3 has 3 chainrings and 6 sprockets, but selecting the combinations 48/24, 38/19 and 28/14 all yield to the same ratio.

Instructions

In the folder `cases` you will find 4 input files and 3 output files. The first 3 input files are the *easy* ones and you can use them to validate your parsing logic and output formatting against the provided output files. The fourth input is the *hard* one and this one does not have a provided output file (so you are on your own here!).

In this folder you will find a password-protected zip file containing the second challenge. To find out the password that will grant you access to it, you must find the solution of `input_04.txt` and calculate its *sha-1* digest. On Unix the command line tool `sha1sum` will give you the digest, on Windows you can use `certutil -hashfile "output_04.txt" SHA1` or if nothing works you can use an online sha-1 tool.

For reference here are the *sha-1* digests of the provided outputs.

```
a45626ea9011376adf3beb32d15d850327e3936a  output_01.txt
3fb0ad35dceaf5d56665495d4b46be6132781044  output_02.txt
e917e8a920132f405a3b827eefe6aedf450f6b6b  output_03.txt
```

For example, if your solution executable is a Python script named `solve.py`, it reads from *stdin* and writes to *stdout*, in order to access the second problem you could do:

```
unzip -P $(python solve.py < cases/input_04.txt | sha1sum | cut -f1 -d '
') second_challenge.zip
```

If you are one of those who use Windows, you could try this Powershell method instead:

```
(Get-Content cases/input_04.txt | python solve.py | Set-Content output);
(Get-Content output -raw | % {$_ -replace "`r", ""} | Set-Content -
NoNewline clean_output); (Get-FileHash -Algorithm sha1 clean_output)
```

Make sure that the solutions generated by your code have matching hashes (if they look the same but the digest don't match, check for extra carriage return characters —newlines should be Unix style!—, trailing lines, whitespace, etc.).

Once you have found the password that succesfully decrypts the zip file, please send an email to `code-challenge@aircall.io` with the subject `aircall-nova challenge part 1`, the *sha-1* digest in the body and the source code for your solution as an attachment. In case nobody solves the second challenge in the alloted time, the winner will be determined by who was the fastest solving the first part 😉