```
foobar:~/doomsday-fuel cihan.goksu.88$ cat readme.txt
```

Doomsday Fuel
==========

Making fuel for the LAMBCHOP's reactor core is a tricky process because of the exotic matter involved. It starts as raw ore, then during processing, begins randomly changing between forms, eventually reaching a stable form. There may be multiple stable forms that a sample could ultimately reach, not all of which are useful as fuel.

Commander Lambda has tasked you to help the scientists increase fuel creation efficiency by predicting the end state of a given ore sample. You have carefully studied the different structures that the ore can take and which transitions it undergoes. It appears that, while random, the probability of each structure transforming is fixed. That is, each time the ore is in 1 state, it has the same probabilities of entering the next state (which might be the same state). You have recorded the observed transitions in a matrix. The others in the lab have hypothesized more exotic forms that the ore can become, but you haven't seen all of them.

Write a function solution(m) that takes an array of array of nonnegative ints representing how many times that state has gone to the next state and return an array of ints for each terminal state giving the exact probabilities of each terminal state, represented as the numerator for each state, then the denominator for all of them at the end and in simplest form. The matrix is at most 10 by 10. It is guaranteed that no matter which state the ore is in, there is a path from that state to a terminal state. That is, the processing will always eventually end in a stable state. The ore starts in state 0. The denominator will fit within a signed 32-bit integer during the calculation, as long as the fraction is simplified regularly.

For example, consider the matrix m:

```
[
  [0,1,0,0,0,1],  # s0, the initial state, goes to s1 and s5 with equal probability
  [4,0,0,3,2,0],  # s1 can become s0, s3, or s4, but with different probabilities
  [0,0,0,0,0,0],  # s2 is terminal, and unreachable (never observed in practice)
  [0,0,0,0,0,0],  # s3 is terminal
  [0,0,0,0,0,0],  # s4 is terminal
  [0,0,0,0,0,0],  # s5 is terminal
]
```

So, we can consider different paths to terminal states, such as:

s0 -> s1 -> s3
s0 -> s1 -> s0 -> s1 -> s0 -> s1 -> s4
s0 -> s1 -> s0 -> s5
Tracing the probabilities of each, we find that
s2 has probability 0
s3 has probability 3/14
s4 has probability 1/7
s5 has probability 9/14
So, putting that together, and making a common denominator, gives an answer in the form of
[s2.numerator, s3.numerator, s4.numerator, s5.numerator, denominator]
which is
[0, 3, 2, 9, 14].

Languages
=========

To provide a Java solution, edit Solution.java
To provide a Python solution, edit solution.py

Test cases
==========
Your code should pass the following test cases.
Note that it may also be run against hidden test cases not shown here.

-- Java cases --
Input:
Solution.solution({{0, 2, 1, 0, 0}, {0, 0, 0, 3, 4}, {0, 0, 0, 0, 0}, {0, 0, 0, 0,0}, {0, 0, 0, 0, 0}})
Output:
    [7, 6, 8, 21]

Input:
Solution.solution({{0, 1, 0, 0, 0, 1}, {4, 0, 0, 3, 2, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}})
Output:
    [0, 3, 2, 9, 14]

-- Python cases --
Input:
solution.solution([[0, 2, 1, 0, 0], [0, 0, 0, 3, 4], [0, 0, 0, 0, 0], [0, 0, 0, 0,0], [0, 0, 0, 0, 0]])
Output:
    [7, 6, 8, 21]

```
Input:
solution.solution([[0, 1, 0, 0, 0, 1], [4, 0, 0, 3, 2, 0], [0, 0, 0,
0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0]])
Output:
    [0, 3, 2, 9, 14]
```

Use verify [file] to test your solution and see how it does. When you are finished editing your code, use submit [file] to submit your answer. If your solution passes the test cases, it will be removed from your home folder.