1. Problem definition

An interesting kind of number in mathematics is vampire number (Links to an external site.). A vampire number is a composite (Links to an external site.) natural number (Links to an external site.) with an even number of digits, that can be factored into two natural numbers each with *half as many digits as the original number* and *not both with trailing zeroes*, where the two factors contain precisely all the digits of the original number, in any order, counting multiplicity. A classic example is: 1260= 21 x 60.

A vampire number can have multiple distinct pairs of fangs. A vampire numbers with 2 pairs of fangs is: 125460 = 204 × 615 = 246 × 510.

The goal of this first project is to use Elixir and the actor model to build a good solution to this problem that runs well on multi-core machines.

2. Requirements

Input: The input provided (as command line to your program, e.g. my_app) will be two numbers: N1 and N2. The overall goal of your program is to find all vampire numbers starting at N1 and up to N2.

Output: Print, on independent lines, first the number then its fangs. **If there are multiple fangs list all of them** next to each other like it's shown in the example below.

Your File name should be proj1.

Example 1:

```
mix run proj1.exs 100000 200000
```

```
125460 204 615 246 510
```

This output indicates that a vampire number between 100000 and 200000 is 125460 and its possible pair of fangs are: 204, 615 and 246, 510.

Actor modeling: In this project, you must use exclusively the actor facility in Elixir (**projects that do not use multiple actors or use any other form of parallelism will receive no credit**). In particular, define worker actors that are given a range of problems to solve and a boss that keeps track of all the problems and perform the job assignment.

README file: In the README file you must include the following material:

1. Please mention group members names and UFID. Also write the steps to run your code.
2. The number of worker actors that you created.
3. Size of the work unit of each worker actor that you determined results in best performance for your implementation and an explanation on how you determined it. Size of the work unit refers to the number of sub-problems that a worker gets in a single request from the boss.

4. The result of running your program for: **mix run proj1.exs 100000 200000**
5. Report the running time for the above problem (4). The ratio of CPU time to REAL TIME tells you how many cores were effectively used in the computation. If you are close to 1 you have almost no parallelism (points will be subtracted).
6. The largest problem you managed to solve (For example You can try finding out bigger vampire numbers than 200000).
7. (Optional)- You could also inspect your code with observer (using- **:observer.start**) and attach a screenshot of CPU utilization chart.