

## Introduction:

The goal of this project is to use Elixir and the actor model to build a good solution to the Lucas Square Pyramid problem that runs well on multi-core machines. (The bonus part will be in another document)

## Problem definition

An interesting problem in arithmetic with deep implications to *elliptic curve theory* is the problem of finding *perfect squares that are sums of consecutive squares*. A classic example is the Pythagorean identity:

$$3^2 + 4^2 = 5^2 \quad (1)$$

that reveals that the sum of squares of 3, 4 is itself a square. A more interesting example is Lucas' *Square Pyramid* :

$$1^2 + 2^2 + \dots + 24^2 = 70^2 \quad (2)$$

In both of these examples, sums of squares of consecutive integers form the square of another integer.

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

## Requirements:

**Input:** The input provided (as command line to your program, e.g. my app) will be two numbers: N and k. The overall goal of your program is to find all k consecutive numbers starting at 1 and up to N, such that the sum of squares is itself a perfect square (square of an integer).

**Output:** Print, on independent lines, the first number in the sequence for each solution.

**Actor modeling:** This project uses exclusively the actor facility in Elixir.

## Results:

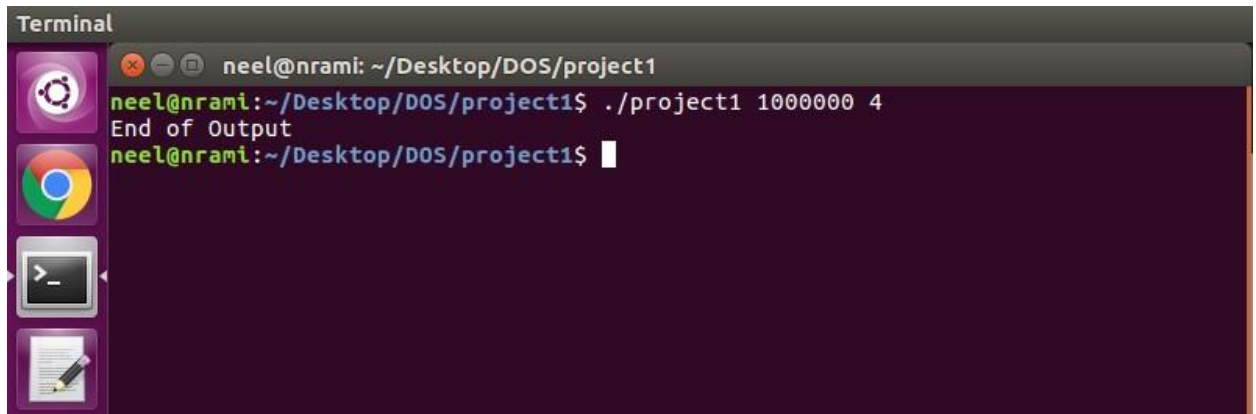
### 1. Instructions for running the code

- a. For Ubuntu based systems
  - i. Go the project directory
  - ii. Type the command in the terminal: `mix escript.build` (Optional)
  - iii. Type the command in the terminal: `./project1 1000000 4`
  - iv. Here the first command line argument is the value of 'n'
  - v. Here the second command line argument is the value of 'k'
  - vi. General command: `./project1 <n> <k>`
- b. For Windows
  - i. Go the project directory
  - ii. Type the command in the cmd: `mix escript.build` (Optional)
  - iii. Type the command in the cmd: `escript .\project1 1000000 4`

- iv. Here the first command line argument is the value of 'n'
- v. Here the second command line argument is the value of 'k'
- vi. General command: `escript .\project1 <n> <k>`

## 2. Result of running the program

Command given: `./project1 1000000 4`



```
Terminal
neel@nrami: ~/Desktop/DOS/project1
neel@nrami:~/Desktop/DOS/project1$ ./project1 1000000 4
End of Output
neel@nrami:~/Desktop/DOS/project1$
```

No number was printed. This means that there are no 4 consecutive numbers whose sum of squares is a perfect square.

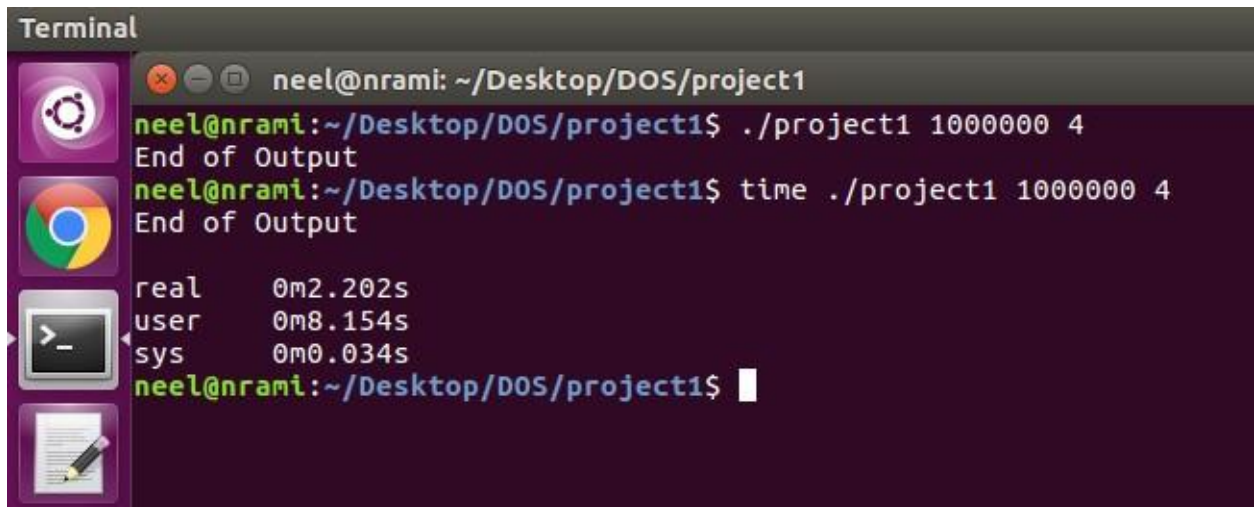
When only 'End of Output' is printed, it means no number was printed.

## 3. Output Format

- a) First all the start indices which follow the Lucas Square Pyramid Principle will be displayed line by line.
- b) Then 'End of Output' is printed.
- c) When only 'End of Output' is printed, it means no number was printed.
- d) The list of numbers is not printed in sorted order.

## 4. Running time of the program

Command given: `time ./project1 1000000 4`

A terminal window titled "Terminal" with a dark background. The prompt is "neel@nrami: ~/Desktop/DOS/project1". The first command is "./project1 1000000 4", which outputs "End of Output". The second command is "time ./project1 1000000 4", which outputs "End of Output" followed by timing statistics: "real 0m2.202s", "user 0m8.154s", and "sys 0m0.034s". The prompt returns to "neel@nrami:~/Desktop/DOS/project1\$". On the left side of the terminal, there are four icons: a gear, a Chrome logo, a terminal icon, and a notepad icon.

```
Terminal
neel@nrami: ~/Desktop/DOS/project1
neel@nrami:~/Desktop/DOS/project1$ ./project1 1000000 4
End of Output
neel@nrami:~/Desktop/DOS/project1$ time ./project1 1000000 4
End of Output
real    0m2.202s
user    0m8.154s
sys     0m0.034s
neel@nrami:~/Desktop/DOS/project1$
```

Real Time = 2.202 seconds

CPU Time = 8.188 seconds (User Time + System Time)

CPU to Real Time Ratio = 3.72

This means 4 cores were used. Also the program was run on a computer with 4 cores.

5. Size of the work unit and reason:

- a. The number actors in our program is 10. Thus, **the size of the work unit is  $n/10$ .**
- b. **Reason:** We have tested our program with several actors along with combinations of  $N$  and the expected value of CPU time to real time ratio for actors with value 10 and above was more or less the same. So we just decided to keep 10 actors so work unit is  $n/10$ .

## 6. Output for other values of n and k

```
Terminal File Edit View Search Terminal Help
neel@nrami:~/Desktop/DOS/project1$ time ./project1 1000000 24
1
9
20
25
44
76
121
197
304
202289
353
540
856
1301
2053
3112
3597
5448
306060
8576
12981
20425
841476
128601
30908
353585
534964
35709
54032
84996
End of Output

real    0m2.562s
user    0m9.278s
sys     0m0.026s
neel@nrami:~/Desktop/DOS/project1$
```

```
neel@nrami:~/Desktop/DOS/project1
neel@nrami:~/Desktop/DOS/project1$ time ./project1 1000000 49
25
End of Output

real    0m3.030s
user    0m11.016s
sys     0m0.040s
neel@nrami:~/Desktop/DOS/project1$ time ./project1 1000000 409
71752
End of Output

real    0m10.979s
user    0m42.412s
sys     0m0.045s
neel@nrami:~/Desktop/DOS/project1$ time ./project1 1000000 289
20
140
199
287
433
724
1595
End of Output

real    0m7.798s
user    0m29.819s
sys     0m0.028s
neel@nrami:~/Desktop/DOS/project1$
```