Assignment – 2

(Topics covered: Basic Functions, Lists, Recursion)

Your tasks

The Möbius Function is a simple classifier of positive integers, devised by August Möbius. Given a positive integer n, the Möbius function μ(n) returns:

- •0 if the number is a multiple of a square
- •-1 if the number has an odd number of distinct prime factors
- •1 if the number has an even number of distinct prime factors

The n = 1 case is a special case; μ(1) is defined to return 1.

You can probably see right away that this function is all about generating and then analyzing the list of prime factors for the input value. A number is a multiple of a square if it has any duplicate prime factors; for example, 12 = 2 × 2 × 3, so it is a multiple of a square, and therefore μ(12) = 0.

For this section we will focus on one interesting detail about the Möbius Function; runs of square-multiples. We want to **write a program that finds the first run of square-multiples of length N**. For example, the first run of three square-multiples (N=3) starts with 48; 48, 49 and 50 are all multiples of squares.

Note: I will be testing each sub function with all random test cases ;)

All of your recursive functions for this section should be tail-recursive!

Your Tasks

1. **Create a module named "<your first name>", in the file "<your first name>.erl".**
   Example: module named "akshay", in the file akshay.erl

Functions to code:

2.**Create a function is_prime that takes a single argument N, and returns true if the number is prime, or false if the number is not prime.**

Your implementation doesn't need to be particularly fast. For example, you can create a helper function that iterates over integers from 2 to sqrt(N), checking whether N is evenly divisible by each value. If N is not evenly divisible by any value then it is prime; otherwise, it is not prime.

Some Tips from my side to decrease computation time:

- **Tip 1:** You can use the remainder operator rem to check if N is evenly divisible by a particular value. A rem B returns the remainder from dividing A by B.
- **Tip 2:** Computing square-roots is computationaly costly. A better approach is to take the value, which we will call Val, and if Val × Val > N then you are done. (You could use guard statements [when etc.])

Add is_prime to your module's export-list, and test it to make sure it works properly.

3.**Create a function prime_factors that takes a single argument N, and returns a list of all prime factors of N.** The result doesn't have to be in any particular order. Add this function to your module's export-list.

4.**Create a function is_square_multiple that takes a single argument N, and returns true if the argument is a multiple of some square, or false if it is not.**

- **Tip 1:** A number is a multiple of a square if it has any prime-factors that appear more than once.
- **Tip 2:** You might find some of the functions in the lists and/or sets modules to be of use! The online erlang documentation is available http://erlang.org/erldoc. For example, you might sort the list of prime-factors, then search the sorted list for the same prime-factor appearing twice in sequence.

5.**Finally, create a function find_square_multiples(Count, MaxN).** This function takes a count of how many square-multiples in a row there should be, and also a maximum value of where to stop searching.

- If the function finds a series of Count square-multiples in the range [2, MaxN], it should return the first number in the run.
- If the function doesn't find any series of Count square-multiples in this range, it should return the atom fail.

So, for example, you might have this interaction:

```
1> c(akshay.erl).
{ok,akshay}
2> akshay:find_square_multiples(3, 49).
48
3> akshay:find_square_multiples(3, 20).
fail
```

Note that the start of the run must be no larger than MaxN; the entire run itself may extend beyond MaxN. Example 1 above shows that MaxN=49, so the start of run should be less than or equal to 49 but the run can exceed MaxN=49 ie. The run is 48, 49, 50.

This function should also be exported by your mobius module.

6.**Final task: Once you have finished this function, find the first square-multiple runs of length 4, length 5, and length 6.** You will need to choose a MaxN of 30000. Your program should definitely complete in under 1 minute; if it takes longer then you may have non-tail-recursive code in your implementation, and you need to fix this. (In fact, a good implementation shouldn't take very long to find the answers, but you will have up to a minute to compute the answer.)

Put your results at the end of a <your first name>.txt file.

Example: akshay.txt contains (for example, these are not the answers :P)

```
akshay:find_square_multiples(4,300).
65
akshay:find_square_multiples(5,6000).
1425
```

Best of luck to all the coders!!

Mail your .erl and .txt results to ieeedpwer@googlegroups.com