ICS440/SEC540

Cryptography and Blockchain Applications

TERM 201 – Sultan Almuhammadi

Programming Assignment – PA2

Due: Monday, Nov 16

The goal of this assignment is to enhance your understanding of some concepts in number theory that we have studied earlier, like: Fermat little theorem, and fast exponentiation. You will also learn how to generate large primes of some special form called *Mersenne primes*.

In this assignment, you will do some experiments with prime numbers, and use your primality test function *is_prime* (in PA1) to generate large Mersenne primes. You need to report all your findings in one file saved as *PA2.report.pdf* for submission.

Problem Set: [100 points]

Q1. [30 pts] *Mersenne prime*: is a prime number of the form $M_p = 2^p - 1$, where p is a prime number. For example, $M_5 = 2^5 - 1 = 31$ is a Mersenne prime. Notice that if p is not prime, then $2^p - 1$ must be composite. To generate a Mersenne prime of a large size, take a small prime p, then compute $n = 2^p - 1$, use primality test to check if n is prime. If so, then n is Mersenne prime of length p bits. Otherwise, n is called a Mersenne composite. Implement a Boolean function *mersenne* (p) that returns true if and only if M_p is a Mersenne prime. The function should call your primality test function *is_prime* (with k bases shown below).

a) In the main program, use your function to print all the Mersenne primes M_p for p between 2 and 29, (with k = 3 in the *is_prime* function).
b) Repeat part (a) using k =1 (for base 2 only), and compare the results
c) Report the outcomes of this experiment (in the *PA2.report.pdf* file)
d) Test *mersenne* (31). How long does it take to test if M_{31} is a Mersenne prime? Report your observations with justification if any.

Q2. [40 pts] Implement the modular exponentiation (a.k.a. fast exponentiation) function *mod_exp* (b, n, m) to compute $b^n \pmod m$ more efficiently. (Hint: to read n bit-by-bit, use / and % operations repeatedly)

a) Test your function for b = 3, $n = 2^{31} - 2$, $m = 2^{31} - 1$.
b) Report the result and the time (in seconds) it takes to find the result.

Q3. [30 pts] Modify your *is_prime* function to use the *mod_exp* (b, n, m) instead of the standard power operation (b**n % m). Rename it as *is_prime2*. Modify the *mersenne* (p) function to use *is_prime2*, and call it *mersenne2*.

a) Use the modified function *mersenne2* to print all the Mersenne primes M_p for p between 2 and 31 if possible, (with k = 3 in the *is_prime* function). Compare the results with the ones found in Q1.
b) Gradually increase the range of p to find more Mersenne primes (say up to p = 101 if possible). What is the largest Mersenne prime you can achieve here?

c) Extend the work in part (b) and find the maximum Mersenne prime you can get from this experiment in a reasonable amount of time (Say within two minutes).
d) Report your findings in this experiment, and comment on the speed up achieved by implementing the fast exponentiation algorithm.

Submission: (Deadline: see Blackboard)

What to submit? Submit a zipped file containing the codes of all the functions (*mersenne, mod_exp, is_prime2, mersenne2*), the main programs, and the report *PA2.report.pdf*. You can either keep the codes in one file or separated files saved as: *q1, q2, q3* and *main_program* (with appropriate extension for the programming language), then zip them all with the report in one zip file.

File Name: Save the zip file as **PA2.csn##.zip**, where ## is your CSN. (For Python users, you may submit a Jupyter-notebook including all the codes, saved as **PA2.csn##.ipynb**)