Azer Hojlas
2021-04-08

Pseudo-Random Number Generator

Overview:

This PRNG was designed according to the lecture slides, i.e the Lehner generator. This generator has been designed in accordance with the following requirements from the lecture slides:

- Desired properties for a PRNG Uniform distribution Independent numbers
- Very long period
- A simple PRNG is a linear congruential generator
- nextSeed = coefficient * seed + constant (mod modulus)
- *coefficient* ε [1*, m* – 1] , *constant* ε [0*, m* – 1]

In order to be able to safely encrypt/decrypt as many bytes as possible, I had to choose a high value “modulus” and a primitive root that would result in a high period, more specifically, a prime number (prime numbers produce the highest period according to Euler's function). The period is equivalent to how many bytes that can be safely processed, as one number in this sequence corresponds to one byte that can be xored. As the requirements have stated that one must be able to encrypt/decrypt files with several megabytes, I have chosen the variable modulus to be roughly 10 000 000, i.e 10 MB. This variable and it’s primitive root has been found with the help of the WolframAlpha website, which hosts online mathematical tools that are free to use. In order to avoid sequences with 1 period, I have restricted the seed to any non-zero integer number, as well as setting the constant to 0.

Overwritten methods:

- int next(int bits):
  I have replaced the java method with the one described in the picture slides. I have used the “nextSeed” variable to receive a value r = nextSeed/modulus that is between 0 and 1, *r* ε (0*,* 1). This value is then multiplied with 2 to the power of argument “bits”, in order to ensure that the sought after result is in the desired binary range. We add this number with 1 and take the floor of this new value to get our desired sequence range. The argument can be modified and called through the nextInt method, which I have left unmodified. Finally, the nextSeed is updated for the next number in the sequence, by calling the method “setSeed”.

- void setSeed(long seed)
  This method is self-explanatory. The seed gets updated with the value calculated in the formula above.

- This class that I have created inherits the Random class, which has been practical as I have only needed to overwrite two methods. Otherwise this would have proven to be a much more difficult task.