Pseudorandom Number Generator Comparison

Bradley Allen

Introduction

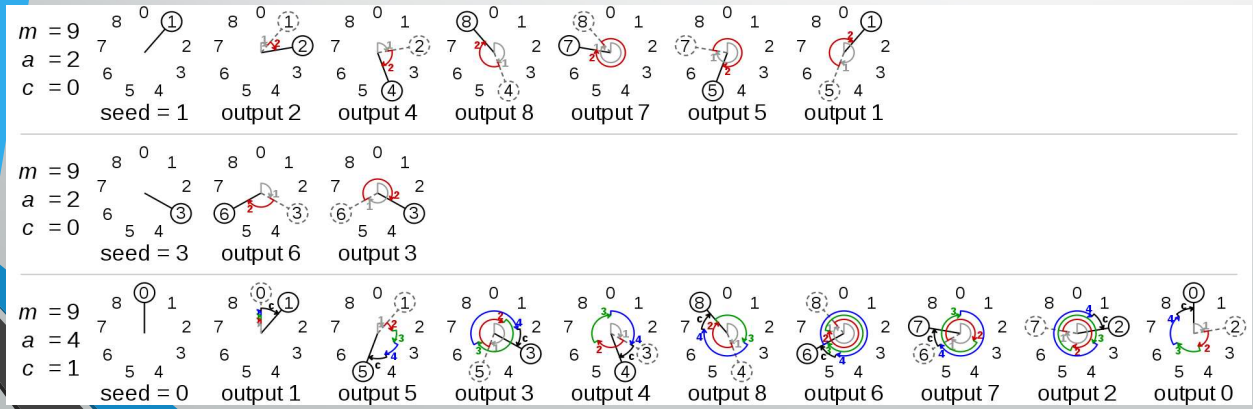
- How to generate a “random” number?
- Linear Congruential Generator
  - $R = \left((S * M) + N\right) \% \, mod$
- XOR Shift

```
public long randomLong() {
  seed ^= (seed << 21);
  seed ^= (seed >>> 35);
  seed ^= (seed << 4);
  return seed;
}
```

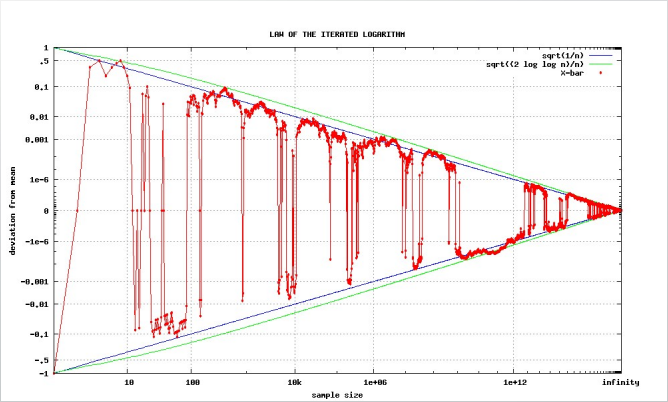
Literature Review

- Study by Ahmad Gaeini, et al.
  - Law of Iterated Logarithm (LIL)
- LCG's low range

TYPE of ALGORITHM	NAME of ALGORITHM	Against LIL	PASS or FAIL
Block Cipher	AES256	Resistant	PASS
Stream Cipher	Salsa20	Resistant	PASS
PRNG	MT19937	Resistant	PASS
PRNG	PHP-MT	Weak	FAIL
PRNG	Standard C LCG	Weak	FAIL



By Cmglee - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=38637545



By Dean P Foster - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6771312

Methodology

- Linear Congruential Algorithm
  - Advantages
    - Simple
    - Fast
  - Disadvantages
    - Low range, repeats quickly
- XOR Shift Algorithm
  - Advantages
    - More random
  - Disadvantages
    - More complex

Implementation

- Java
  - System.nanoTime()
  - Math library
    - Math.abs()
- Eclipse IDE

Experimental Setup

- Standard Variables
  - Seed = System.nanoTime()
  - Modulus = 73
  - Numbers generated = 1,000
- Comparison Metrics
  - Randomness (Repetition)
  - Gap Test
  - Performance (Time elapsed)

Implementation Code #1 (LCG)

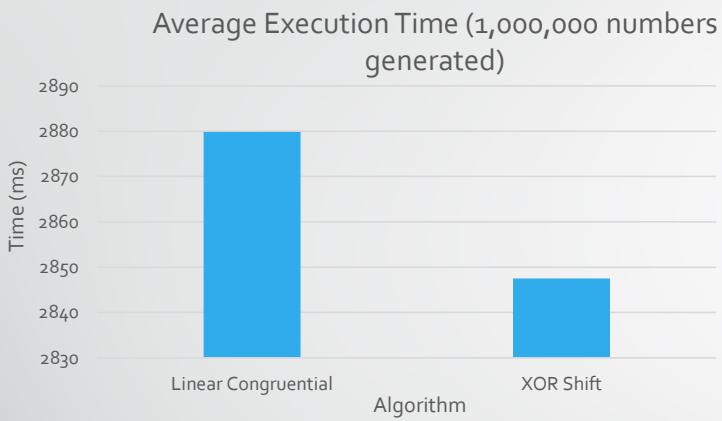
```java
public class LinearCongruential {

    /**
     * Linear congruential algorithm
     * @param seed
     * @param modulus
     * @param multiplier
     * @param increment
     * @param numOfRandomNums
     * @param randomnumbers
     */
    private static void generate(int seed, int modulus, int multiplier, int increment, int numOfRandomNums, int[] randomnumbers) {
        randomnumbers[0] = seed;

        for (int i=1; i<numOfRandomNums; ++i) {
            randomnumbers[i] = ((randomnumbers[i-1] * multiplier) + increment) % modulus;
        }
    }

    /**
     * Driver method
     */
    public static void main(String[] args) {
        int seed = (Math.abs((int) System.currentTimeMillis())) % 73;
        //int seed = 1;
        int modulus = 73;
        int multiplier = 2;
        int increment = 3;
        int numOfRandomNums = 100;
        int[] randomnumbers = new int[numOfRandomNums];
        int[] occurrences = new int[modulus];

        generate(seed, modulus, multiplier, increment, numOfRandomNums, randomnumbers);

        // Read over all randomly generated numbers
        for (int i=0; i<randomnumbers.length; ++i) {
            // Increment counter for that number
            occurrences[randomnumbers[i]] += 1;

            // Print number generated
            System.out.print(randomnumbers[i] + " ");
        }

        System.out.println("");

        // Print number of occurrences
        for (int i=0; i<occurrences.length; ++i) {
            System.out.println(i + ": " + occurrences[i]);
        }

    }
}
```

Implementation Code #2 (XOR Shift)

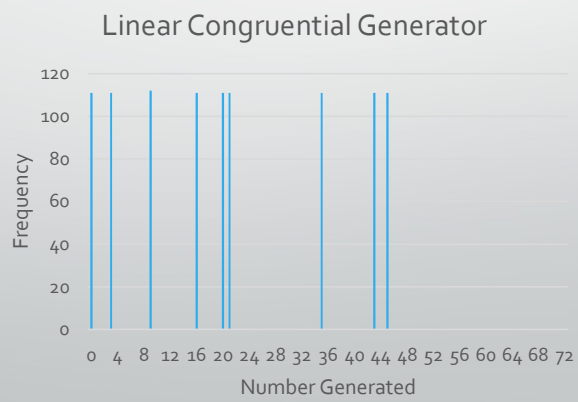
```java
public class XORShift {

    static long seed;

    /**
     * Generates a random long number
     * using XOR shift
     * @return Random Number (seed)
     */
    public static long randomLong() {
      seed ^= (seed << 21);
      seed ^= (seed >>> 35);
      seed ^= (seed << 4);
      return Math.abs(seed % 73);
    }

    public static void main(String[] args) {
        seed = System.nanoTime();
        int numToGenerate = 100;
        int modulus = 73;
        int[] randomnumbers = new int[numToGenerate];
        int[] occurrences = new int[modulus];

        for (int i=0; i<randomnumbers.length; ++i) {
            randomnumbers[i] = (int) randomLong();

            // Increment counter for that number
            occurrences[randomnumbers[i]] += 1;

            // Print number generated
            System.out.print(randomnumbers[i] + " ");
        }

        System.out.println("");

        /** **/
        // Print number of occurrences
        for (int i=0; i<occurrences.length; ++i) {
            System.out.println(i + ": " + occurrences[i]);
        }

    }

}
```

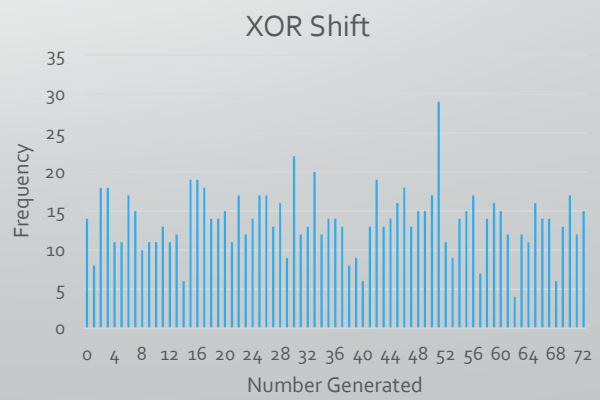
Results Analysis

- Speed/Performance



- Gap Test
- Randomness





Conclusion

- Linear Congruential Algorithm
  - Advantages
    - Simple
    - Fast
  - Disadvantages
    - Low range, repeats quickly
- XOR Shift Algorithm
  - Advantages
    - More random
  - Disadvantages
    - More complex

Conclusion

- Best (of the two):
  - XOR Shift
- Unique Study
  - Comparing XOR Shift and Linear Congruential

References

- Arobelidze, Alexander. "Random Number Generator: How Do Computers Generate Random Numbers?" *FreeCodeCamp.org*, FreeCodeCamp.org, 8 June 2021, https://www.freecodecamp.org/news/random-number-generator/.
- *Chapter 3 Pseudo-Random Numbers Generators - University of Arizona*. https://www.math.arizona.edu/~tgk/mc/book_chap3.pdf.
- "Generating Random Data." *Generating Pseudorandom Numbers - MATLAB & Simulink*, https://www.mathworks.com/help/stats/generating-random-data.html.
- "Java Program to Implement the Linear Congruential Generator for Pseudo Random Number Generation." *GeeksforGeeks*, 17 July 2021, https://www.geeksforgeeks.org/java-program-to-implement-the-linear-congruential-generator-for-pseudo-random-number-generation/.
- "Pseudorandom Number Generators." *Oracle Help Center*, 14 July 2022, https://docs.oracle.com/en/java/javase/17/core/pseudorandom-number-generators.html.
- "XORShift Random Number Generators." *Javamex.* https://www.javamex.com/tutorials/random_numbers/xorshift.shtml

Demo