Analisis Berbagai Macam Pseudo Random Number Generator

Muhammad Naufal (13514073)
Teknik Informatika / Sekolah Tinggi Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
mnaufal75@gmail.com

***Abstract*—Saat ini keamanan data merupakan salah satu hal yang sangat penting. Untuk itu dibutuhkan algoritma kriptografi yang baik, yang dapat digunakan untuk mengamankan pesan dengan baik. Beberapa elemen yang dibutuhkan dalam kriptografi adalah generator random number yang baik dan memenuhi standar.**

Keyword—PRNG; Multiple-with-carry; Blum Blum Shub; Mersenne Twister

I. PENDAHULUAN

Pada masa sekarang ini, teknologi sudah berkembang dengan sangat pesat. Dengan perkembangan teknologi yang pesat ini, maka dibutuhkan beberapa cara untuk mengamankan teknologi. Salat satu cara yang digunakan sebagai pengamanan adalah kriptografi. Kriptografi adalah seni mengamankan pesan sehingga tidak dapat dimengerti selain pengirim dan penerima.

Salah satu aspek yang penting dalam ilmu kriptografi adalah pembangkit bilangan acak yang baik. Bilangan acak yang ideal adalah bilangan acak yang tidak dapat ditebak kemunculan angkanya.

Terdapat dua pendekatan untuk membangkitkan nilai acak, yaitu pembangkit bilangan acak semu dan pembangkit bilangan acak sejati. Pembangkit bilangan acak semu menggunakan rumus dan teori matematika untuk membangkitkannya.

II. DASAR TEORI

A. Pseudo Random Number Generator

PRNG (Pseudo Random Number Generator) digunakan untuk membangkitkan bilangan acak. Angka yang dihasilkan oleh PRNG sebenarnya tidak random, karena ditentukan oleh masukan awal (seed) dan rumus matematika yang digunakan.

PRNG dimanfaatkan dalam berbagai hal yaitu games, kriptografi, dan simulasi.

B. Multiple-with-carry

Multiple-with-carry adalah sebuah algoritmaa pembangkit bilangan acak yang dikembangkan oleh George Marsalia. Kelebihan utama dari algoritma ini adalah kecepatannya, karena algoritma ini hanya memanfaatkan operasi aritmatika sederhana.

Berikut ini adalah implementasi dalam bahasa Python:

```
import random

def Seed():
   multiplier_a = 4164903690
   base_b = 2**32

   c = random.randint(1, multiplier_a)
   x = random.randint(1, base_b)

   for _ in range(100000):
       _x = (multiplier_a * x + c) % base_b
       _c = (multiplier_a * x + c) % base_b
       x = _x
       c = _c
       print(x)

if __name__ == "__main__":
  Seed()
```

C. Blum Blum Shub

Blum Blum Shub adalah sebuah algoritma bilangan acak yang dikembangkan oleh Lenore Blum, Manuel Blum, dan Michael Shub.

Berikut ini adalah implementasi algoritma Blum Blum Shub dalam bahasa Python:

```
import random
import prime

def gcd(p, q):
   while (p != q):
```

```
        if (p > q):
            p = p - q
        else:
            q = q - p
    return p

def phi(p):
    result = 1
    for i in range(2, p):
        if gcd(i, p) == 1:
            result = result + 1
    return result

def generateM():
    p = random.randrange(2, 10000)
    while (p % 4 != 3 or prime.millerRabin(p) ==
False):
        p = random.randrange(2, 10000)
    q = random.randrange(2, 10000)
    while (q % 4 != 3 or prime.millerRabin(q) ==
False):
        q = random.randrange(2, 10000)
    print("{} {}".format(p, q))
    print("{} {}".format(p % 4, q % 4))
    print("Phi {}".format(gcd(phi(p - 1), phi(q -
1))))

    M = p * q
    return M

def generateSeed(M):
    seed = random.randrange(2, 10000)
    while (gcd(seed, M) != 1 or
prime.millerRabin(seed) == False):
        seed = random.randrange(2, 10000)
    print("Seed {}".format(seed))
    return seed

if __name__ == "__main__":
    M = generateM()
    seed = generateSeed(M)

    for i in range(100000):
        xi = (seed * seed) % M
        print(xi)
        seed = xi
```

D. Mersenne Twister

Algoritma ini dikembangkan oleh Makoto Matsumoto dan Takuji Nishimura pada tahun 1997. Merupakan salah satu algoritma yang paling sering digunakan untuk membangkitkan bilangan acak.

Terdapat beberapa varian dari Mersenne Twister, yang saya gunakan pada penelitian ini adalah MT19937, dengan implementasi MT19937 dalam Python di bawah ini:

```
hiMask = 0xffffffff80000000
loMask = 0x000000007fffffff

matrixA = 0xB5026F5AA96619E9

class MT19937:
    state = []
    index = notSeeded

    def Seed(self, seed):
        x = MT19937.state
        x.append(seed)
        for i in range(1, n):
            x.append(6364136223846793005 * (x[i -
1] ^ (x[i - 1] >> 62)) + i)
        MT19937.index = n

    def Uint64(self):
        x = MT19937.state

        if MT19937.index >= n:
            if MT19937.index == notSeeded:
                MT19937.Seed(self, 5489)

            for i in range(n - m):
                y = (x[i] & hiMask) | (x[i + 1] &
loMask)
                x[i] = x[i + m] ^ (y >> 1) ^ ((y
& 1) * matrixA)

            for i in range(n - m, n - 1):
                y = (x[i] & hiMask) | (x[i + 1] &
loMask)
                x[i] = x[i + m - n] ^ (y >> 1) ^
((y & 1) * matrixA)

            y = (x[n - 1] & hiMask) | (x[0] &
loMask)
            x[n - 1] = x[m - 1] ^ (y >> 1) ^ ((y
& 1) * matrixA)
            MT19937.index = 0

        y = x[MT19937.index]
        y ^= (y >> 29) & 0x5555555555555555
        y ^= (y << 17) & 0x71D67FFFEDA60000
        y ^= (y << 37) & 0xFFF7EEE000000000
        y ^= (y >> 43)
        MT19937.index += 1

        return y
```

III. METODE ANALISIS

Untuk pengujian algoritma-algoritma di atas, akan digunakan metode Kendall dan Smith. Metode ini berguna untuk mengukur *randomness* dari algoritma-algoritma tersebut secara statistik.

Pada metode Kendall dan Smith ini terdapat 4 tes yang dilakukan:

1. Frequency Test, untuk melihat apakah setiap digit mempunyai jumlah kemunculan yang sama
2. Serial Test, sama seperti Frequency Test, akan tetapi untuk melihat frekuensi kemunculan pasangan 2 digit
3. Poker Test, adalah pengujian frekuensi bit dengan pengelompokan lima bit.
4. Gap Test, untuk melihat jarak antara dua 0 (00 berarti *gap* berukuran 0, 0120 berarti *gap* berukuran 2)

IV. PENGUJIAN

Pada penelitian yang dilakukan ini, akan dilakukan. Implementasi perangkat lunak tersebut dilakukan pada lingkungan dengan spesifikasi sebagai berikut:

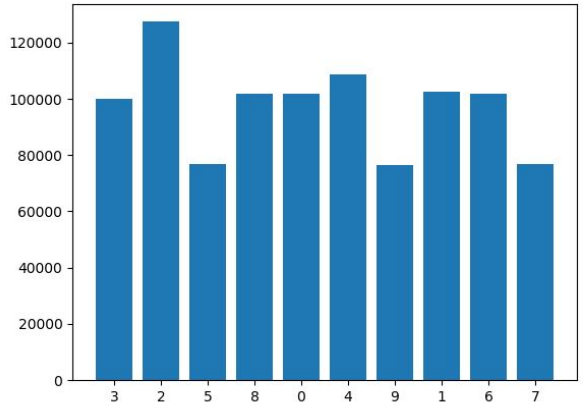
1. Processor: Intel(R) Core(™) i3-3217U CPU @ 1.80 GHz (4CPUs)
2. RAM: 4096 MB
3. System Type: 64-bit Operating System
4. Hard Disk: 500 GB
5. VGA: Intel(R) HD Graphics 4000
6. Operating System: Microsoft Windows 8.1 Ultimate

Pada percobaan ini akan diambil tiga tes dari metode Kendall dan Smith, yaitu frequency test, serial test, dan poker test.

A. Frequency Test

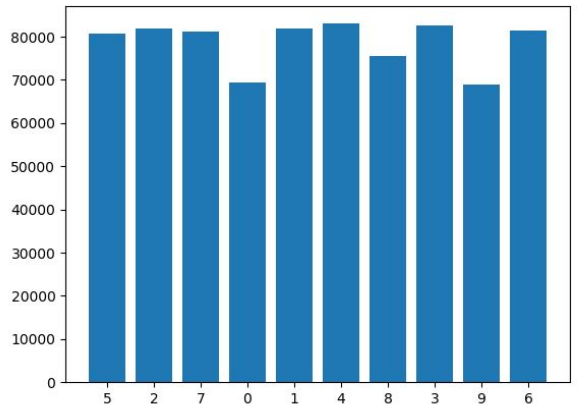
Semakin rata grafik yang dihasilkan, maka semakin baik pula algoritma tersebut.

1. Multiple-with-carry



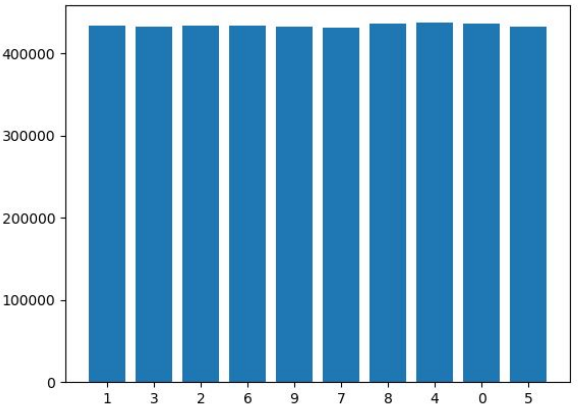
Gambar 1

2. Blum Shub



Gambar 2

3. Mersenne Twister

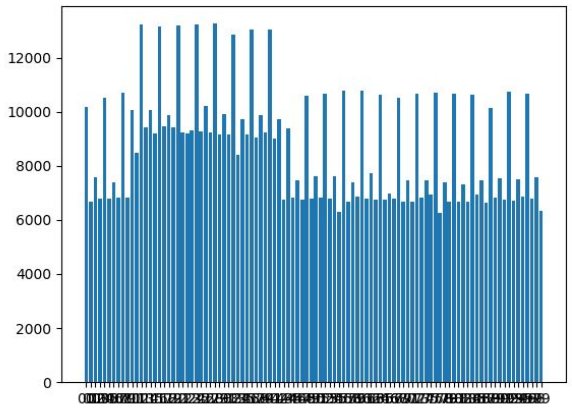


Gambar

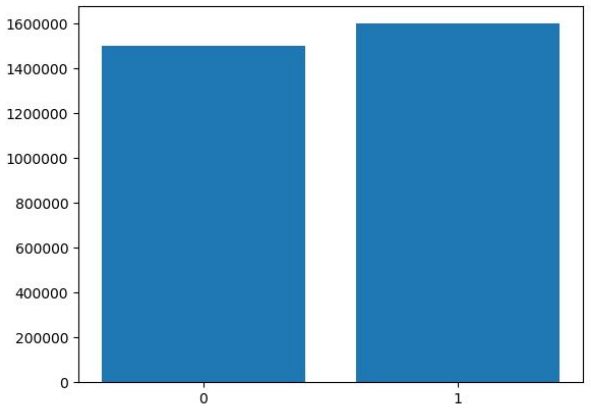
B. Serial Test

Semakin rata grafik yang dihasilkan, maka semakin baik pula algoritma tersebut.

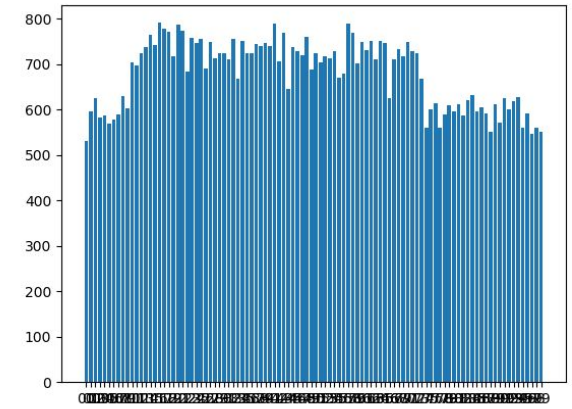
1. Multiple-with-carry



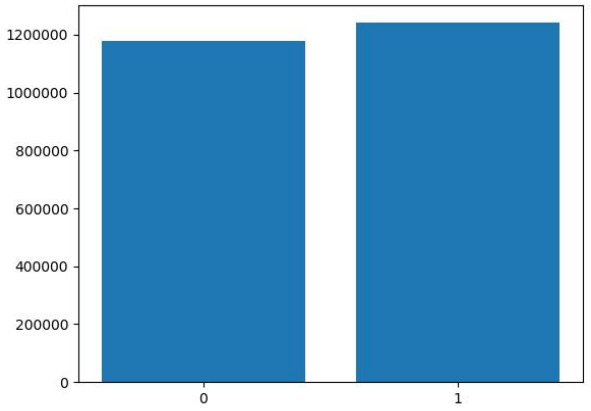
2. Blum Blum Shub



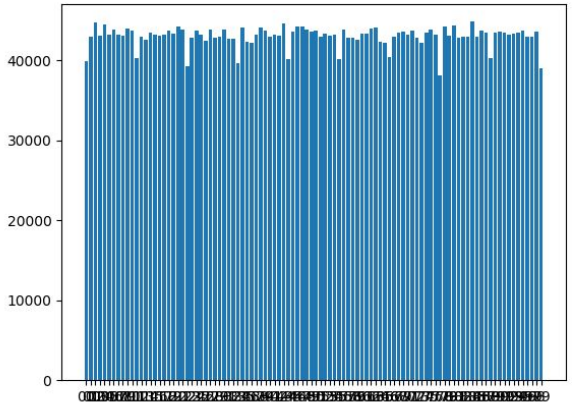
2. Blum Blum Shub

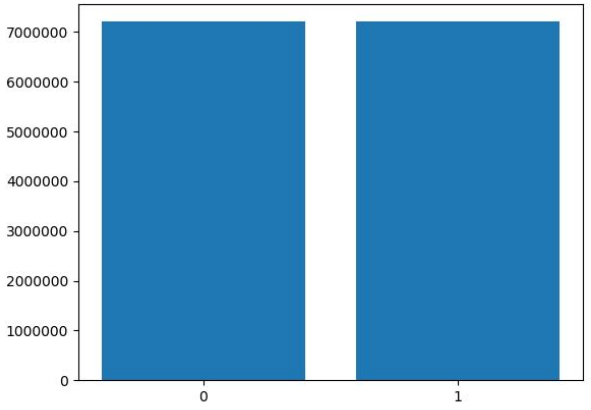


3. Mersenne Twister



3. Mersenne Twister





C. Poker Test

Semakin rata grafik yang dihasilkan, maka semakin baik pula algoritma tersebut.

1. Multiply-with-carry

V. KESIMPULAN DAN SARAN

Dari hasil ketiga tes yang telah dilakukan pada ketiga algoritma di atas, dapat disimpulkan bahwa algortima Mersenne Twister adalah yang paling baik, karena grafik

yang dihasilkan paling merata dibandingkan dengan dua algoritma lain.

KATA PENUTUP

Saya ingin mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T., sebagai dosen mata kuliah IF4020 Kriptografi yang telah membagikan ilmu ini selama satu semester.

DAFTAR PUSTAKA

[1] http://www.quadibloc.com/crypto/co4814.htm
[2] http://wiki.fib.upc.es/sim/index.php/Blum_Blum_Shub
[3] https://www.garykessler.net/library/crypto.html

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiarisasi

Bandung, 18 Mei 2018

Muhammad Naufal