Analisis dan Perbandingan berbagai Algoritma Pembangkit Bilangan Acak

Micky Yudi Utama - 13514011
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514011@std.stei.itb.ac.id

Abstrak—Makalah ini bertujuan untuk membandingkan dan menganalisis berbagai pembangkit bilangan acak yang sudah dibangun. Pembangkit bilangan acak memiliki peranan yang luas dalam berbagai bidang. Oleh karena itu, perlu diketahui kelebihan dan kekurangan dari setiap pembangkit bilangan acak, agar dapat ditentukan algoritma yang akan digunakan pada domain tertentu. Algoritma yang akan dianalisis adalah Mersenne Twister, xoroshiro128+, PCG, SplitMix, dan Lehmer. Untuk masing-masing algoritma, akan dilakukan pengukuran kualitas statistik dengan TestU01 dan kecepatannya. Selain itu, akan dianalisis kompleksitas, penggunaan memori, dan periode dari masing-masing algoritma.

Kata kunci—Pembangkit bilangan acak, kelebihan, kekurangan, TestU01

I. Pendahuluan

Pembangkit bilangan acak merupakan salah satu aplikasi yang sangat banyak digunakan dalam berbagai hal, seperti simulasi dengan komputer, judi, kriptografi, dan bidang-bidang lainnya yang membutuhkan angka acak. Bilangan acak dapat dibangkitkan oleh komputer dengan menggunakan algoritma tertentu dan masukan berupa sebuah nilai penentu yang sering disebut sebagai *seed*. Jika *seed* yang diberikan ke algoritma sama, maka bilangan acak yang dibangkitkan juga akan sama. Pembangkit bilangan acak seperti ini disebut juga sebagai *pseudo-random number generator* (PRNG) atau *deterministic random bit generator* (DRBG).

PRNG yang baik tentunya memiliki beberapa syarat minimal yang harus terpenuhi. Menurut The German Federal Office for Information Security, terdapat dua kualitas utama yang harus dipenuhi oleh pembangkit bilangan acak, yakni dapat membangkitkan sejumlah bilangan acak yang berbeda dengan probabilitias yang tinggi, dan dapat memenuhi beberapa *statistical tests*. Sedangkan, jika PRNG digunakan dalam dunia kriptografi, maka terdapat dua syarat tambahan yang harus dipenuhi yakni dapat bertahan ketika dilakukan penyerangan meskipun penyerang telah mengetahui beberapa *state* atau beberapa nilai acak yang telah dibangkitkan. Hal ini dikarenakan dalam dunia kriptografi, sekuriti merupakan komponen yang sangat penting.

Terdapat banyak algoritma yang telah dibuat untuk membangkitkan bilangan acak, seperti Mersenne Twister dan variannya, HC-256, ChaCha20, ISAAC64, PCG dan variannya, xoroshiro128+, xorshift+, Random123, dll. Setiap algoritma tentu memiliki kelebihan dan kelemahan masing-masing. Oleh karena itu, pada makalah ini, akan dilakukan eksperimen dan analisis terhadap beberapa algoritma PRNG. Algoritma yang dipilih adalah dua varian dari Mersenne Twister yakni MT19937 dan SFMT dikarenakan banyaknya penggunaan Mersenne Twister dalam berbagai aplikasi. Selain itu, dipilih juga xoroshiro128+ dan PCG yang dianggap sebagai dua algoritma terbaik saat ini. Kemudian, dipilih SplitMix sebagai salah satu algoritma yang cukup baru dan terkenal. Yang terakhir, dipilih Lehmer64 yang merupakan pengembangan dari LCG.

Tujuan dari makalah ini adalah untuk mengetahui kelebihan dan kekurangan dari masing-masing algoritma. Dengan begitu, dapat ditentukan algoritma mana yang dapat digunakan pada domain tertentu.

II. *Pseudo-Random Number Generator*

Pseudo-random number generator (PRNG) adalah pembangkit bilangan acak yang menerima masukan berupa nilai yang disebut sebagai *seed*. Nilai tersebut kemudian diproses dengan algoritma tertentu untuk menghasilkan nilai keluaran yang kemudian dianggap sebagai nilai acak. Telah banyak algoritma PRNG yang telah dikembangkan. Beberapa yang terkenal di antaranya adalah Mersenne Twister dan variannya, xoroshiro128+, dan keluarga PCG.

Meskipun banyak algoritma telah dibuat, namun tidak semua algoritma berkualitas. Beberapa masalah yang sering terjadi pada aplikasi pembangkit bilangan acak seperti dapat ditebak dan tidak aman. Banyak pembangkit bilangan acak yang keluarannya dapat ditebak dengan cara menganalisis beberapa keluaran dari algoritma tersebut. Selain itu, beberapa pembangkit bilangan acak memiliki performansi kurang baik Selain memberikan hasil yang baik, pembangkit bilangan acak tentunya juga harus efisien. Efisiensi dari pembangkit bilangan acak dilihat dari kompleksitas dan kebutuhan memorinya.

Selain itu, kecepatan pembangkitan bilangan acak juga dapat digunakan sebagai ukuran efisiensi.

Pada bagian ini, akan dibahas lebih rinci terkait dengan algoritma pembangkit bilangan acak yang akan dianalisis, yakni Mersenne Twister, Xoroshiro128+, PCG, SplitMix, dan Lehmer. Selain itu, akan dijelaskan juga mengenai LCG (Linear Congruential Generator) yang merupakan dasar dari algoritma PCG dan Lehmer.

A. Mersenne Twister

Mersenne Twister merupakan algoritma pembangkit bilangan acak yang dikembangkan pada tahun 1997 oleh Makoto Matsumoto dan Takuji Nishimura. Algoritma ini diciptakan untuk menutup kekurangan-kekurangan yang sering terjadi pada PRNG lama. Mersenne Twister merupakan algoritma PRNG pertama yang dapat membangkitkan *integer* yang berkualitas dalam waktu yang singkat. Selama beberapa tahun, Mersenne Twister menjadi algoritma PRNG yang paling sering digunakan dalam berbagai jenis aplikasi, seperti Microsoft Excel, R, Ruby, PHP, Python, dll. Algoritma Mersenne Twister sering juga disebut MT19937 yang didasarkan oleh Mersenne Prime, $2^{19937-1}$.

Terdapat banyak varian dari Mersenne Twister yakni CryptMT, MTGP, TinyMT, dan SFMT. CryptMT merupakan *stream cipher* dan masuk dalam kategori *cryptographically secure pseudo-random bumber generator* (CSPRNG). MTGP merupakan varian dari Mersenne Twister yang dioptimasi penggunaannya untuk GPU (*Graphics Processing Unit*). TinyMT adalah varian dari Mersenne Twister yang menggunakan *state space* yang kecil yaitu 127-bit. Hal ini ditujukan untuk menghemat penggunaan memori, namun di sisi lain periode dari algoritma berkurang menjadi 2^{127-1}. SFMT (SIMD-*oriented Fast Mersenne Twister*) dikembangkan pada tahun 2006. SFMT merupakan modifikasi dari Mersenne Twister yang dapat membangkitkan bilangan acak lebih cepat jika dijalankan pada 128-bit SIMD. Selain itu, SFMT juga memberikan banyak keuntungan lainnya.

B. Xoroshiro128+

Xoroshiro128+ (XOR, *rotate, shift, rotate*) merupakan algoritma pembangkit bilangan acak yang dikembangkan oleh Sebastian Vigna dan David Blackman. Mereka melakukan perubahan pada algoritma xorshift+ dengan mengubah operasi dasarnya menjadi transformasi linear dengan *shift*/*rotate*. Hasilnya didapatkan perubahan yang signifikan yakni peningkatan kecepatan pembangkitan bilangan acak dan peningkatan kualitas statistik. Berikut merupakan kode program xoroshiro128+ dalam bahasa C.

```
#include <stdint.h>

static inline uint64_t rotl(const uint64_t x, int k) {
        return (x << k) | (x >> (64 - k));
}

static uint64_t s[2];

uint64_t next(void) {
        const uint64_t s0 = s[0];
        uint64_t s1 = s[1];
        const uint64_t result = s0 + s1;
        s1 ^= s0;
        s[0] = rotl(s0, 24) ^ s1 ^ (s1 << 16); // a, b
        s[1] = rotl(s1, 37); // c
        return result;
}
```

C. LCG

Linear Congruential Generator (LCG) merupakan salah satu algoritma pembangkit bilangan acak yang sudah tua. LCG membangkitkan sejumlah bilangan acak dengan melakukan perhitungan berdasarkan persamaan linear diskontinu. LCG tergolong algoritma yang sangat sederhana, mudah dimengerti, dan mudah diimplementasi. Berikut merupakan kode program LCG dalam bahasa Python.

```
def lcg(modulus, a, c, seed):
    while True:
        seed = (a * seed + c) % modulus
        yield seed
```

D. PCG

Permuted Congruential Generator (PCG) merupakan algoritma pembangkit bilangan acak yang dikembangkan oleh Melissa E. O'Neill pada tahun 2014. PCG memiliki sangat banyak varian yang bergantung pada transformasi pada keluaran yang didefinisikan. Detail dari setiap variannya dapat dilihat pada paper yang berjudul "PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generator".

Varian yang paling disarankan untuk digunakan adalah PCG-XSH-RR dengan *state* 64-bit dan *output* 32-bit. Tujuan dari varian ini adalah untuk membuat pembangkit bilangan acak yang baik digunakan dalam berbagai hal, dengan menyeimbangkan kecepatan dan performansi statistik dari pembangkit dengan tetap menjaga keamanan algoritma. Strategi yang digunakan adalah dengan melakukan xorshift untuk meningkatkan *high bits*. Kemudian, dengan menggunakan bits 63-59, dilakukan rotasi pada bits 27-58. Berikut merupakan kode program PCG-XSH-RR dalam bahasa C.

```
#include <stdint.h>
static uint64_t      state      = 0x4d595df4d0f33173;
            // Or something seed-dependent
static uint64_t const multiplier = 6364136223846793005u;
static uint64_t const increment  = 1442695040888963407u;
            // Or an arbitrary odd constant
```

```
static uint32_t rotr32(uint32_t x, unsigned r) {
        return x >> r | x << (-r & 31);
}

uint32_t pcg32(void) {
        uint64_t x = state;
        unsigned count = (unsigned)(x >> 59);   // 59 = 64 -
5
        state = x * multiplier + increment;
        x ^= x >> 18;   // 18 = (64 - 27)/2
        return rotr32((uint32_t)(x >> 27), count);   // 27 =
32 - 5
}

void pcg32_init(uint64_t seed) {
        state = seed + increment;
        (void)pcg32();
}
```

E. SplitMix

SplitMix merupakan algoritma pembangkit bilangan acak yang dikembangkan oleh Guy L. Steele, Jr., Doug Lea, dan Christine H. Flood. pada tahun 2014. SplitMix tergolong sederhana dan dapat membangkitkan bilangan acak dengan sangat cepat. SplitMix64 telah digunakan dalam berbagai aplikasi dan merupakan bagian dari standard JAVA API. Berikut merupakan algoritma SplitMix dalam bahasa C.

```
#include <stdint.h>
static uint64_t x; /* The state can be seeded with any value.
*/

uint64_t next() {
        uint64_t z = (x += 0x9e3779b97f4a7c15);
        z = (z ^ (z >> 30)) * 0xbf58476d1ce4e5b9;
        z = (z ^ (z >> 27)) * 0x94d049bb133111eb;
        return z ^ (z >> 31);
}
```

F. Lehmer

Lehmer *random number generator* atau kadang disebut juga Park-Miller *random number generator* merupakan algoritma pembangkit bilangan acak yang dikembangkan dari LCG (*Linear Congruential Generator*). Lehmer merupakan tipe LCG yang beroperasi pada *multiplicative group of integers modulo n*. Fungsi utamanya adalah $X_{k+1} = g \, . \, X_k \, mod \, n$ dengan n adalah bilangan prima atau bilangan prima kuadrat, g adalah elemen dari *high multiplicative order* modulo n, dan X_0 adalah *coprime* dari n. Berikut merupakan kode program algoritma Lehmer dalam bahasa C.

```
uint32_t lcg_parkmiller(uint32_t *state) {
   return *state = ((uint64_t)*state * 48271u) % 0x7fffffff;
}
```

III. Metode Pengujian

Pada makalah ini, pengujian akan dilakukan pada 4 algoritma pembangkit bilangan acak, yakni MT19937, SFMT, xoroshiro128+, PCG-XSH-RR, SplitMix64, dan Lehmer64. Untuk masing-masing algoritma, akan dilakukan pengujian *statistical test*. Selain itu, masing-masing algoritma juga akan dilakukan pengukuran kecepatan pembangkitan bilangan acak. Kecepatan dihitung berdasarkan nilai *clock cycles* yang dibutuhkan PRNG untuk membangkitkan 1 *byte*.

Pengujian dengan *statistical test* dilakukan dengan menggunakan *library* TestU01 yang diimplementasi dengan bahasa C. TestU01 memberikan berbagai fitur, yakni implementasi berbagai algoritma pembangkit bilangan acak yang sudah terkenal atau sering digunakan, dan implementasi berbagai jenis *statistical tests*. Selain itu, TestU01 juga menyediakan 3 jenis *batteries of tests*, yakni SmallCrush yang terdiri dari 10 jenis tes, Crush yang terdiri dari 96 jenis tes, dan BigCrush yang terdiri dari 106 jenis tes. Detail dari setiap tes pada setiap *battery* dapat dilihat pada panduan pengguna dari TestU01.

Kelemahan dari TestU01 adalah hanya dapat menerima masukan 32-bit dan menginterpretasi masukan tersebuat sebagai nilai di antara [0, 1). Hal ini menyebabkan TestU01 lebih sensitif terhadap kecacatan pada MSB (*most-significant bits*) dibandingkan LSB (*least-significant bits*). Oleh karena itu, untuk algoritma PRNG *multi-purpose* seperti Mersenne Twister, xoroshiro128+, dan PCG, perlu dilakukan pengetesan pada bentuk nilai yang terbalik. Hal ini dilakukan karena beberapa aplikasi menggunakan nilai *low-order bits* yang dihasilkan PRNG.

Pengujian *statistical test* dari masing-masing PRNG akan dilakukan dengan menggunakan masing-masing *batteries of tests*, yakni SmallCrush, Crush, dan BigCrush. Untuk masing-masing pengujian, juga akan dilakukan dalam bentuk normal dan terbalik. Untuk masing-masing kombinasi, akan dilakukan pengujian sebanyak tiga kali. Hal ini disebabkan karena setiap pengujian dapat saja memberikan hasil yang berbeda, tergantung pada nilai acak yang dibangkitkan.

Selain dilakukan pengukuran terhadap kualitas statistik dan kecepatan dari masing-masing algoritma, juga akan dianalisis mengenai kompleksitas, penggunaan memori, dan periode dari algoritma pembangkit bilangan acak. Hal ini diperlukan sebagai pertimbangan tambhan untuk memilih pembangkit bilangan acak yang ingin digunakan.

IV. Eksperimen dan Hasil

Eksperimen akan dilakukan pada algoritma MT19937, SFMT, xoroshiro128+, PCG-XSH-RR, SplitMix64, dan Lehmer64. Pertama, akan dilakukan pengukuran kualitas statistik dengan menggunakan TestU01. Berikut merupakan hasilnya.

Tabel 1. Hasil eksperimen dengan TestU01

PRNG	SmallCrush	Crush	BigCrush
MT19937	Pass Pass Pass	Fail Fail Fail	Fail Fail Fail
SFMT	Pass Pass Pass	Fail Fail Fail	Fail Fail Fail
Xoroshiro128+	Pass Pass Pass	Pass Pass Pass	Pass Fail Pass
PCG-XSH-RR	Pass Pass Pass	Pass Pass Pass	Pass Pass Pass
SplitMix64	Pass Pass Pass	Pass Pass Pass	Fail Fail Pass
Lehmer64	Pass Pass Pass	Pass Pass Pass	Fail Fail Pass

Selanjutnya akan ditunjukkan rincian dari jenis kegagalan yang terjadi. Karena kegagalan pada tes BigCrush telah mencakup kegagalan algoritma secara keseluruhan, maka rincian kegagalan hanya ditunjukkan untuk tes BigCrush. Berikut rinciannya.

Tabel 2. Rincian kegagalan pada tes BigCrush

PRNG	*Fail test / p-value*
MT19937	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
SFMT	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
Xoroshiro128+	RandomWalk1 M (L=1000, r=0) / 4.7e-4
SplitMix64	CouponCollector, r = 10 / 0.9995
	Permutation, t = 3 / 7.3e-4
Lehmer64	SampleProd, t = 8 / 7.3e-4
	RandomWalk1 M (L=50, r=0) / 8.0e-4

Kemudian, eksperimen juga akan dilakukan dengan setiap nilai yang dibangkitkan oleh PRNG dibalik terlebih dahulu sebelum dimasukkan ke TestU01. Berikut merupakan hasilnya.

Tabel 3. Hasil eksperimen dengan TestU01 pada *reversed form*

PRNG	SmallCrush	Crush	BigCrush
MT19937	Pass Pass Pass	Fail Fail Fail	Fail Fail Fail
SFMT	Pass Pass Pass	Fail Fail Fail	Fail Fail Fail
Xoroshiro128+	Pass Pass Pass	Pass Pass Pass	Fail Fail Fail
PCG-XSH-RR	Pass Pass Pass	Pass Pass Pass	Pass Pass Pass
SplitMix64	Pass Pass Pass	Pass Pass Pass	Pass Pass Fail
Lehmer64	Pass Pass Pass	Pass Pass Pass	Fail Pass Fail

Berikut merupakan rincian kegagalan dari berbagai algoritma pada tes BigCrush dengan kondisi nilai yang dibalik.

Tabel 4. Rincian kegagalan pada tes BigCrush pada *reversed form*

PRNG	*Fail test / p-value*
MT19937	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	ClosePairs mNP2S, t = 16 / 1.5e-4 LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1

SFMT	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
	LinearComp, r = 0 / 1 - eps1 LinearComp, r = 29 / 1 - eps1
Xoroshiro128+	MatrixRank, L=5000 / eps
	MatrixRank, L=5000 / eps
	MatrixRank, L=5000 / eps
SplitMix64	MaxOft AD, t = 32 / 4.9e-5
Lehmer64	RandomWalk1 J (L=1000, r=0) / 2.6e-4
	WeightDistrib, r = 0 / 0.9992 WeightDistrib, r = 26 / 3.4e-4

Tabel 1 sampai 4 menunjukkan hasil dari TestU01 terhadap masing-masing algoritma. Dapat dilihat bahwa untuk kedua varian dari Mersenne Twister yakni MT19937 dan SFMT, terjadi kegagalan pada tes Crush dan BigCrush. Kemudian, untuk algoritma xoroshiro128+, SplitMix64, dan Lehmer64, kegagalan mulai terjadi pada tes BigCrush meskipun tidak selalu. Untuk algoritma PCG-XSH-RR, hasilnya menunjukkan bahwa algoritma dapat melewati semua tes tanpa terjadi masalah. Oleh karena itu, dapat disimpulkan bahwa PCG memiliki keunggulan dalam hal kualitas statistik, kemudian diikuti oleh xoroshiro128+, SplitMix64, dan Lehmer64, dan kemudian Mersenne Twister.

Selanjutnya, akan diukur kecepatan dari masing-masing algoritma. Pengukuran dilakukan dengan menghitung waktu yang dibutuhkan algoritma pembangkit bilangan acak untuk membangkitkan satu *byte*. Berikut merupakan hasilnya.

Tabel 5. Kecepatan algoritma PRNG

PRNG	*Cycles/byte*
MT19937	2,18
SFMT	1,66
Xoroshiro128+	0,88
PCG-XSH-RR	1,61
SplitMix64	0,80
Lehmer64	0,80

Pada Tabel 5, dapat dilihat bahwa SplitMix64, xoroshiro128+, dan Lehmer64 unggul dalam hal kecepatan. Kemudian, diikuti dengan PCG-XSH-RR dan yang terakhir Mersenne Twister. Kecepatan merupakan salah satu faktor penting dalam mengukur efisiensi dari algoritma pembangkit bilangan acak. Pada beberapa domain, penggunaan pembangkitan bilangan acak dapat saja dilakukan terus menerus. Jika waktu yang dibutuhkan untuk membangkitkan bilangan acak lama, dan jumlah bilangan acak yang diperlukan banyak, maka penggunaannya akan menjadi tidak efisien. Oleh karena itu, hasil ini dapat dijadikan pertimbangan ketika ingin memilih pembangkit bilangan acak yang dapat membangkitkan bilangan acak dengan cepat.

Kemudian dilakukan analisis terhadap *period* dari masing-masing algoritma. *Period* merupakan nilai maksimum dari *initial seed* yang dapat diterima oleh algoritma. Dengan kata lain, *period* merupakan jumlah maksimal bilangan acak unik yang dapat dibangkitkan. Nilai *period* yang besar merupakan salah satu syarat dari PRNG yang baik. Salah satu *rule of thumb* yang ada adalah jika dibutuhkan *t* nilai bilangan acak, maka dibutuhkan pembangkit bilangan acak yang memiliki *period* minimal t^2. Hal ini diperlukan agar dapat menjamin bahwa bilangan acak yang dibangkitkan tidaklah sama, atau probabilitas bilangan acak yang dibangkitkan sama sangat kecil. Berikut merupakan *period* dari masing-masing algoritma.

Tabel 6. *Period* dari algoritma PRNG

PRNG	*Period*
MT19937	2^{19937}-1
SFMT	2^{19937}-1
Xoroshiro128+	2^{128-1}
PCG-XSH-RR	2^{64}
SplitMix64	2^{64}
Lehmer64	2^{64}

Dapat dilihat bahwa PCG-XSH-RR, SplitMix64, dan Lehmer64 memiliki *period* paling kecil yakni 2^{64}. Namun, hal tersebut bukanlah hal yang buruk dikarenakan nilai tersebut merupakan nilai yang cukup besar untuk digunakan dalam berbagai aplikasi. Di sisi lain, Mersenne Twister memiliki *period* yang sangat besar, yakni 2^{19937}-1. Hal ini mengindikasikan bahwa Mersenne Twister memiliki kualitas yang baik. Namun, *period* yang besar berarti membutuhkan *state* yang besar. Penggunaan *state* yang besar berarti penggunaan memori yang besar. Jika, nilai *period* yang besar tidak dibutuhkan, maka penggunaan memori yang dilakukan oleh Mersenne Twister akan menjadi sia-sia.

Analisis terhadap algoritma juga dilakukan pada kompleksitas dari kode program, ukuran kode, dan memori yang digunakan oleh algoritma. Diketahui bahwa xoroshiro128+, PCG, dan SplitMix64 memiliki kode program

yang sangat pendek dan mudah dimengerti. Hal ini berakibat pada kecilnya ukuran kode dan mudahnya algoritma untuk diimplementasi ulang. Selain itu, kebutuhan memori dari algoritma juga sangat kecil sehingga algoritma tetap dapat digunakan jika memori yang ada terbatas. Di sisi lain, Mersenne Twister memiliki ukuran kode yang besar, kode yang cukup panjang dan kompleks, sehingga sulit dimengerti dan diimplementasi ulang. Selain itu, memori yang dibutuhkan oleh Mersenne Twister juga tergolong besar.

Selanjutnya akan dibahas dan dianalisis lebih lanjut mengenai kelebihan dan kekurangan dari masing-masing algoritma. Analisis kelebihan dan kekurangan algoritma dilakukan berdasarkan penelitian yang telah dilakukan pada bagian atas dan berdasarkan referensi dari internet. Berikut merupakan kelebihan dan kekurangan dari Mersenne Twister.

1. Dapat melewati sebagian besar *statistical tests*, meskipun terdapat beberapa tes yang tidak dapat dilewati.
2. Dapat diprediksi setelah membangkitkan 624 bilangan acak.
3. 623-*dimensionality distributed*
4. Dapat melakukan *jump-ahead* (bergerak maju pada sekuens bilangan acak), namun komputasinya lambat.
5. Memiliki periode yang sangat besar yakni $2^{19937-1}$.
6. Kode program yang kompleks sehingga sulit dimengerti.
7. Penggunaan memori yang besar.
8. Lambat.

Untuk SFMT yang merupakan varian dari Mersenne Twister, terdapat beberapa kelebihan tambahan yakni:

1. Memiliki kecepatan dua kali dari Mersenne Twister.
2. Memiliki *equidistribution property of v-bit accuracy* yang lebih baik dari Mersenne Twister.
3. Memiliki kecepatan *recovery* dari *zero-excess initial state* yang lebih dibandingkan Mersenne Twister.
4. Mendukung berbagai *period* dari 2^{607}-1 sampai 2^{16091}-1.

Berikut merupakan kelebihan dan kekurangan dari algoritma xoroshiro128+.

1. Dapat melewati tes BigCrush dari TestU01. Namun, terkadang terdapat beberapa parameter yang gagal untuk dilewati.
2. Kode program sederhana dan mudah dimengerti.
3. Mudah diimplementasi ulang.
4. Penggunaan memori yang sangat kecil.
5. Memiliki periode 2^{128}-1.
6. Sangat cepat.

Berikut merupakan kelebihan dan kekurangan dari algoritma PCG.

1. Dapat melewati semua *statistical tests* dari BigCrush pada TestU01.
2. Tergolong cepat.
3. Memiliki sangat banyak variasi, tergantung tujuan penggunaannya. Variasi yang paling disaranakan untuk digunakan secara luas adalah PCG.XSH-RR.
4. Penggunaan memori yang kecil, bergantung pada variasi yang dipilih.
5. pcg32 memiliki periode sebesar 2^{64}. Namun jika ingin menggunakan PRNG dengan periode besar, dapat digunakan salah satu variasi dari PCG yakni pcg32_k16383 yang memiliki periode sebesar 2^{524352}.
6. Sulit untuk diprediksi, namun tetap tidak tergolong *cryptographically secure*.
7. *Uniform*.
8. Fitur *jump-ahead* dan *distance-between-states.*
9. Sederhana dan mudah dimengerti.

Berikut merupakan kelebihan dan kekurangan dari algoritma SplitMix..

1. Dapat melewati tes BigCrush dari TestU01. Namun, terkadang mengalami kegagalan pada beberapa tes.
2. Kode program yang sederhana dan mudah dimengerti.
3. Sangat cepat.

Berikut merupakan kelebihan dan kekurangan dari algoritma Lehmer.

1. Dapat melewati tes BigCrush dari TestU01. Namun, untuk beberapa tes, terkadang masih dapat mengalami kegagalan.
2. Kode program yang sangat sederhana dan pendek, sehingga mudah dimengerti dan diimplementasi kembali.
3. Sangat cepat.
4. Memiliki beberapa variasi. Penggunaan tergantung pada keinginan pengguna.

V. Kesimpulan dan Saran

Berdasarkan eksperimen dan analisis yang telah dilakukan, diketahui bahwa PCG memiliki kualitas yang paling baik. Selain algoritma PCG dapat melewati kumpulan *statistical tests*, kode programnya juga sederhana, pendek, dan mudah dimengerti, sehingga mudah untuk diimplementasi kembali. Selain itu, proses pembangkitan bilangan acaknya juga tergolong cepat. Oleh karena itu, dapat disimpulkan bahwa PCG merupakan salah satu algoritma pembangkit bilangan acak yang sangat berkualitas.

Algoritma lainnya juga tidak kalah berkualitas. Xoroshiro128+, SplitMix64, dan Lehmer64 meskipun memiliki kualitas statistik yang kurang dari PCG, namun memiliki kecepatan dua kali dari PCG. Sehingga jika membutuhkan algoritma pembangkit bilangan acak yang dapat menghasilkan nilai dengan cepat, maka ketiga algoritma tersebut dapat dipertimbangkan.

Untuk penelitian lebih lanjut terkait pembangkit bilangan acak, dapat dilakukan pengujian pada pembangkit bilangan acak lainnya untuk mengetahui karakteristik yang ada. Selain itu, dapat dilakukan pengujian lebih lanjut, misalnya pengukuran kualitas statistik dengan PractRand.

VI. Ucapan Terima Kasih

Pertama-tama, saya mengucapkan syukur dan terima kasih kepada Tuhan Yang Maha Esa yang telah melimpahkan berkatnya selama pengerjaan makalah ini hingga selesai. Saya juga ingin mengucapkan terima kasih kepada bapak Rinaldi Munir selaku dosen pengajar IF4020 Kriptografi yang telah memberikan materi mata kuliah kriptografi yang membantu dalam proses penyelesaian makalah ini.

Referensi

[1] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", *ACM Trans. on Modeling and Computer Simulation Vol. 8*, No. 1, January pp.3-30 (1998) DOI:10.1145/272991.272995.

[2] Mutsuo Saito and Makoto Matsumoto, "SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator", *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Springer, 2008, pp. 607 -- 622. DOI:10.1007/978-3-540-74496-2_36.

[3] D. Blackman and S. Vigna, "xoshiro/xoroshiro generators and the PRNG shootout".

[4] M. E. O'Neill, "PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation", Harvey Mudd College, HMC-CS-2014-0905.

[5] M. E. O'Neill, "PCG, A Family of Better Random Number Generators".

[6] M. E. O'Neill, "How to Test with TestU01".

[7] M. E. O'Neill, "Specific Problems with Other RNGs".

[8] G. L. S. Jr., D. Lea, and C. H. Flood, "Fast Splittable Pseudorandom Number Generators", *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, 453-472.

[9] C. Wellons, "Finding the Best 64-bit Simulation PRNG".

[10] P. L'Ecuyer & R. Simard (2007), "TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators", *ACM Transactions on Mathematical Software*, 33: 22.

[11] S.K. Park and K.W. Miller (1988), "Random Number Generators: Good Ones Are Hard To Find", *Communications of the ACM*. 31 (10): 1192–1201, doi:10.1145/63039.63042.

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 plagiasi.

Bandung, 18 Mei 2017

Micky Yudi Utama
13514011