Analisis Terhadap Algoritma Pembangkit Bilangan Acak "Multiply-with-Carry"

Okaswara Perkasa (13510051)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
okaswara@students.itb.ac.id

***Abstrak* — Makalah ini akan melakukan analisis pada salah satu algoritma pembangkit bilangan acak semu, yakni multiply-with-carry. Penjelasan singkat mengenai algoritma tersebut dibahas pada bagian awal makalah. Sejumlah uji coba dan analisis dilakukan pada multiply-with-carry, terutama keunggulannya dari algoritma pembangkit bilangan acak semu yang lain, yakni linear congruence generator.**

***Kata kunci* — pseudorandom generator, multiply-with-carry**

I. PENDAHULUAN

Pada era informasi dewasa ini, bilangan acak sangat dibutuhkan dalam berbagai macam bidang. Misalnya pada *video games*, pembangkitan bilangan acak sangat diperlukan untuk menambahkan faktor "ketidakpastian" dalam permainan. Bilangan acak juga merupakan salah satu komponen yang esensial pada security, dimana bilangan acak digunakan pada banyak protokol keamanan.

Pada prakteknya, bilangan acak sendiri dibuat dengan menggunakan suatu algoritma tertentu, yang disebut juga sebagai pembangkit bilangan acak (*random number generator*).

Sayangnya, karena sifat komputer yang deterministik, dalam arti hasil keluaran yang diberikan dapat diprediksi dengan pasti jika menggunakan suatu masukan tertentu, maka sebenarnya secara teoritik komputer tidak mungkin membuat urutan bilangan yang benar-benar acak.

Seluruh algoritma pembangkit bilangan acak pada komputer pada umumnya menghasilkan bilangan acak yang semu, atau yang disebut juga sebagai pembangkit bilangan acak semu (*pseudo random number generator).*

Pembangkit bilangan acak yang semu seolah-olah menghasilkan urutan bilangan yang benar-benar acak, namun setelah menghasilkan sejumlah bilangan acak, maka urutan kemunculan bilangan di bagian awal akan terulang kembali.

Oleh karena itu, algoritma pembangkit bilangan acak semu harus dapat menyembunyikan adanya siklus pada skema pembangkitan bilangannya. Algoritma yang sederhana juga dibutuhkan suatu pembangkit bilangan acak, agar bilangan acak dapat dibangkitkan dengan cepat.

Dari sejumlah algoritma pembangkit bilangan acak yang ada, salah satu algoritma bilangan acak yang memenuhi kedua kriteria tersebut adalah multiply-with-carry (MWC).

II. DASAR TEORI

A. Multiply-with-Carry (MWC)

Multiply-with-carry adalah salah satu algoritma pembangkit bilangan acak semu yang sederhana. Algoritma ini dikembangkan oleh George Marsaglia, seorang pakar dalam pembangkitan bilangan acak[1].

Secara matematis, pembangkitan bilangan acak mengikuti persamaan berikut[2]:

$$\boldsymbol{x_n = (a x_{n-1} + c_{n-1}) \bmod b}$$
$$\boldsymbol{c_n = \left\lfloor \frac{a x_{n-1} + c_{n-1}}{b} \right\rfloor}$$

Dengan a adalah multiplier, b adalah base, c adalah carry, dan urutan x_0, x_1, x_2, ..., x_n adalah urutan bilangan acak yang dihasilkan.

Nilai x_0 dan c_0 adalah masukan dari user, dan pada algoritma ini bekerja seperti seed dalam pembangkitan bilangan acak.

Agar algoritma dapat berjalan dengan baik, dalam arti dapat menghasilkan siklus terpanjang yang mungkin, terdapat sejumlah syarat yang harus dipenuhi, yakni[2]:

1. $c_0 < b$, dan
2. ab-1 adalah bilangan prima

Pada implementasinya, untuk menyimpan kedua nilai seed, yakni x_0 dan c_0, digunakan satu buah integer 32 bit, yang dipartisi menjadi dua bagian untuk menampung kedua nilai seed tersebut. Jadi, masing-masing nilai x_0 dan c_0 akan memiliki besar 16 bit.

Selain itu, sejumlah multiply-with-carry dapat dikombinasikan untuk menghasilkan siklus yang jauh lebih panjang dari sebelumnya.

Berikut contoh implementasi algoritma multiply-with-carry dalam bahasa C++, yang menggunakan dua pembangkit multiply-with-carry yang berbeda[1]:

```
unsigned int mwc()   {
    x = 36969 * (x & 65535) + (x >> 16);
    y = 18000 * (y & 65535) + (y >> 16);
    return (x << 16) + y;
}
```

Code 1 – Implementasi multiply-with-carry

Pada source code diatas, x dan y merupakan dua nilai seed untuk dua algoritma multiply-with-carry yang berbeda. Salah satu pembangkit menggunakan nilai a = 36969, dan pembangkit yang lain menggunakan a = 18000.

Ekspresi (x & 65535) dan (y & 65535) akan melakukan operasi and pada kedua seed, yang pada persamaan sebelumnya akan menghasilkan nilai x_n. Selain itu, ekspresi (x >> 16) dan (y >> 16) akan melakukan right shift pada setiap bit, untuk mendapatkan nilai c_n.

Setelah mendapatkan nilai xn dan cn, dilakukan perhitungan untuk mendapatkan nilai xn+1 dan cn+1.

Terakhir, kedua nilai x yang didapatkan dari kedua pembangkit digabungkan, salah satunya menempati 16 LSb dan yang lain menempati 16 MSb. Nilai inilah yang merupakan bilangan acak yang dihasilkan.

B. Linear Congruence Generator (LCG)

Linear congruence generator adalah algoritma pembangkit bilangan acak semu yang banyak digunakan secara praktikal.

Secara matematis, pembangkitan bilangan acak dirumuskan sebagai berikut[3]:

$$x_n = (ax_{n-1} + c) \bmod b$$

Sekilas, hampir tidak ada perbedaan dengan rumus matematis untuk multiply-with-carry, namun perbedaan utama diantara keduanya adalah, pada linear congruence generator, nilai c (carry) selalu konstan.

Seperti multiply-with-carry, terdapat sejumlah syarat yang perlu dipenuhi agar algoritma pembangkit ini dapat bekerja dengan baik, yakni[3]:

1. c dan b haruslah relatif prima, dan
2. seluruh fakor prima b juga merupakan faktor prima dari a-1

Pada implementasinya, terdapat sejumlah varian pada algoritma ini. Setiap varian menggunakan nilai a, b, dan c yang berbeda-beda. Kebanyakan varian yang ada merupakan implementasi yang dilakukan pada sejumlah compiler, seperti versi Borland C++, Borland Delphi, ataupun ANSI C[4].

Berikut implementasi linear congruence generator dalam bahasa C++ (versi ANSI C).

```
unsigned int lcg()   {
   x = (1103515245u * x + 12345) & 2147483647u;
   return x;
}
```

Code 2 – Implementasi linear congruence generator

Pada implementasi ANSI C, nilai a yang digunakan adalah 1103515245, carry bernilai 12345, dengan base sebesar 2^{32}.

C. Diehard Tests

Diehard tests adalah salah satu test yang digunakan untuk memeriksa kualitas dari sejumlah urutan bilangan acak. Seperti halnya multiple-with-carry, George Marsaglia adalah orang yang mengembangkan algoritma pemeriksaan ini pada tahun 1995[5].

Tes ini sendiri terdiri dari lima belas tes urutan bilangan acak yang berbeda-beda[5]:

1. Birthday Spacings
2. Overlapping Permutations
3. Ranks of 31x31 and 32x32 matrices
4. Ranks of 6x8 Matrices
5. Monkey Tests on 20-bit Words
6. Monkey Tests OPSO, OQSO, DNA
7. Count the 1`s in a Stream of Bytes
8. Count the 1`s in Specific Bytes
9. Parking Lot Test
10. Minimum Distance Test
11. Random Spheres Test
12. The Sqeeze Test
13. Overlapping Sums Test
14. Runs Test
15. The Craps Test

George Marsaglia sendiri sebenarnya mengembangkan metode tes ini sudah berbentuk dalam program siap pakai. Program Diehard ini dapat didownload pada [5].

Untuk dapat menjalankan setiap tes yang ada, maka minimal diperlukan sebanyak 2.9 juta bilangan acak, yang berurutan sesuai dengan waktu pembangkitan[5].

III. UJI COBA DAN ANALISIS

Sejumlah uji coba dilakukan untuk menentukan kinerja dari algoritma multiply-with-carry. Algoritma linear congruence generator digunakan sebagai pembanding.

Implementasi untuk uji coba menggunakan C++. Kedua algoritma pembangkit bilangan acak menggunakan source code yang sudah dijabarkan pada dasar teori.

Beberapa uji coba yang dilakukan: Panjang siklus, uji statistik, kecepatan pembangkitan bilangan acak, serta diehard tests.

A. Panjang Siklus

Uji coba ini akan melakukan pembangkitan bilangan acak pada kedua algoritma, sampai algoritma tersebut akan mengulangi siklus pembangkitan.

Cara memeriksa bahwa pengulangan akan terjadi adalah dengan melihat seed yang dihasilkan. Jika seed yang dihasilkan adalah sama dengan seed awal, maka, sesuai sifat komputer yang deterministik, urutan bilangan acak yang dihasilkan akan sama dengan urutan bilangan acak pada bagian awal. Pembangkitan bilangan acak juga akan dihentikan jika bilangan acak yang dihasilkan sudah mencapai sepuluh miliar bilangan.

Setelah diujicobakan, algoritma multiply-with-carry dapat melampaui batas sepuluh miliar bilangan yang dilakukan. Sepuluh miliar sendiri sudah melampaui ukuran suatu word, yang berukuran 4 byte, atau sekitar empat miliar.

Sedangkan pada algoritma linear congruence generator, pengulangan akan terjadi setelah pembangkitan bilangan acak sebanyak dua miliar kali, jauh dibawah siklus yang dimiliki oleh multiply-with-carry.

Hal ini dapat dijelaskan secara teoritik, mengingat algoritma multiply-with-carry yang digunakan berisi dua buah seed yang masing-masing berukuran 2^{32} bit. Sedangkan algoritma linear congruence generator hanya memiliki satu buah seed saja, dengan ukuran yang sama. Oleh karena itu, algoritma multiply-with-carry memiliki variasi seed yang jauh lebih banyak, dan karena itu, memiliki siklus yang lebih panjang.

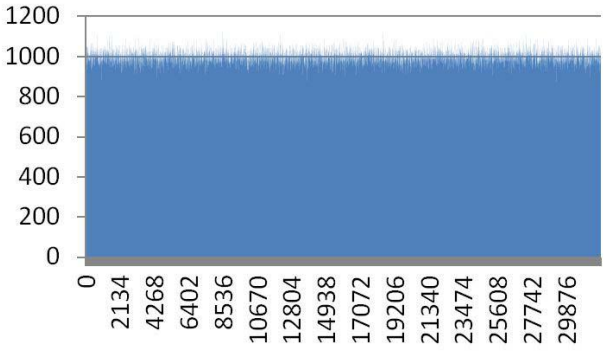
Maka, dapat disimpulkan bahwa multiply-with-carry lebih baik daripada linear congruence generator, dalam hal panjang siklus yang dihasilkan.

B. Uji Statistik

Uji coba ini akan membangkitan sejumlah bilangan acak, yang nantinya bilangan acak yang dihasilkan akan dihitung jumlah kemunculannya.

Kedua algoritma akan membangkitkan bilangan acak dari 0 s.d. 99999 (seratus ribu kemungkinan), sebanyak seratus juta kali. Pemberian batas pembangkitan dilakukan dengan memoduluskan bilangan acak yang dihasilkan dengan 100000 (seratus ribu).

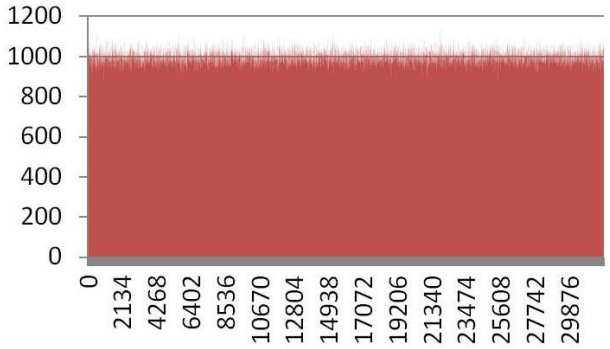
Berikut merupakan distribusi yang dihasilkan oleh algoritma multiply-with-carry:



Gambar 1 – Distribusi bilangan acak dengan multiply-with-carry

Perhitungan statistik menunjukkan bahwa data memiliki variansi sebesar 998.235660, atau standar deviasi sebesar 31.594868.

Berikut merupakan distribusi yang dihasilkan oleh algoritma linear congruence generator:



Gambar 2 – Distribusi bilangan acak dengan linear congruence generator

Perhitungan statistik menunjukkan bahwa data memiliki variansi sebesar 953.295540, atau standar deviasi sebesar 30.875484.

Kedua distribusi menghasilkan grafik yang hampir sama; keduanya memiliki jumlah kemunculan di sekitar 1000 kemunculan, yang memang merupakan rata-rata kemunculan dari setiap angka.

Namun, hasil perhitungan standar deviasi dan variansi menunjukkan nilai yang sedikit berbeda. Algoritma multiply-with-carry memberikan nilai standar deviasi yang sedikit lebih tinggi daripada linear congruence generator. Ini menunjukkan bahwa nilai acak yang dihasilkan multiply-with-carry lebih tidak seragam daripada linear congruence generator.

Sebenarnya, ketidakseragaman ini tidak hanya dapat menjadi kelebihan, namun juga kekurangan. Ketidakseragaman pada pembangkitan dapat membuat algoritma lebih mendekati pembangkitan bilangan acak yang nyata. Namun, di sisi lain, ketidakseragaman dapat menyebabkan kurangnya *fairness* dalam pemilihan bilangan.

Oleh karena itu, secara statistik, tidak ada yang lebih baik diantara kedua algoritma. Algoritma mana yang lebih baik tergantung pada penerapannya, apakah memerlukan pembangkitan bilangan acak yang seragam atau tidak.

C. Kecepatan Pembangkitan Bilangan Acak

Uji coba ini akan menghitung lama waktu yang diperlukan kedua algoritma untuk menghasilkan sejumlah bilangan acak.

Kedua algoritma akan diujikan dengan cara membangkitkan sebanyak satu miliar bilangan acak. Durasi yang diperlukan untuk membangkitkan seluruh bilangan acak tersebut akan dihitung untuk setiap algoritma.

Pada algoritma multiply-with-carry, waktu yang diperlukan untuk membangkitkan satu miliar bilangan acak adalah 18.22 detik.

Sedangkan pada algoritma linear congruence generator, waktu yang diperlukan adalah sebanyak 22.49 detik.

Maka, dapat disimpulkan bahwa multiply-with-carry dapat membuat bilangan acak dengan lebih cepat.

Walaupun operasi yang dilakukan pada multiply-with-carry lebih banyak daripada operasi pada linear congruence generator, namun pada operasi perkalian, orde nilai yang dikalikan pada multiply-with-carry jauh lebih kecil. Selain itu, operator-operator yang lain, seperti logical and dan shift, merupakan operasi dasar pada hardware, sehingga dapat dilakukan dengan sangat cepat.

D. Diehard Tests

Uji coba ini akan melakukan diehard tests pada urutan pembangkitan bilangan acak pada kedua algoritma.

Nantinya kedua algoritma akan membangkitkan sebanyak tiga juta bilangan acak, dan masing-masing bilangan acak tersebut akan dijalankan pada aplikasi Diehard.

Namun sayangnya, uji coba gagal untuk dilakukan pada kedua algoritma. Terdapat sejumlah kendala teknis pada pemasukan data, sehingga program selalu mengalami crash ketika menerima masukan urutan bilangan acak.

KESIMPULAN

Secara umum, algoritma multiply-with-carry memiliki performa yang lebih baik, jika dibandingkan dengan linear congruence generator.

Multiply-with-carry menghasilkan bilangan acak semu dengan siklus yang lebih panjang, selain itu bilangan acak juga dihasilkan dengan kecepatan yang lebih tinggi.

Algoritma ini juga memiliki ketidakseragaman yang lebih tinggi daripada linear congruence generator. Hal ini dapat menguntungkan, dan juga dapat merugikan, tergantung dimana algoritma tersebut diterapkan.

Diehard test juga dapat dilakukan pada kedua algoritma, untuk lebih memperdalam pemahaman mengenai algoritma multiply-with-carry, dan perbandingannya dengan linear congruence generator.

REFERENSI

[1] http://www.codeproject.com/Articles/25172/Simple-Random-Number-Generation, 18 Mei 2013

[2] http://www.stat.fsu.edu/pub/diehard/cdrom/pscript/mwc1.ps, 18 Mei 2013

[3] http://www.cs.indiana.edu/~kapadia/project2/node7.html, 19 Mei 2013

[4] http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3686&rep=rep1&type=pdf, 19 Mei 2013

[5] http://www.stat.fsu.edu/pub/diehard, 20 Mei 2013

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, 20 Mei 2013

Okaswara Perkasa (13510051)