Pembangkit Bilangan Acak Semu dan Kaitannya dengan P vs. NP

Mohammad Dwinta Harits C./13519041
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519041@std.stei.itb.ac.id

***Abstract*—Seberapa cepat kita dapat mengurutkan sebuah larik bilangan? Atau mecari angka di dalam larik tersebut? Cukup cepat bukan? Bagaimana jika kita diharuskan menentukan rute tercepat pada sebuah graf berbobot sehingga setiap simpul tepat dikunjungi sekali dan rute membentuk siklus? atau menentukan barang jualan apa saja yang harus dimasukkan ke dalam tas dengan kapasitas tertentu hingga mendapatkan nilai jual maskimal? Apakah sama cepatnya dengan persoalan terkait larik?**

Bagaimana dengan bilangan acak? Apakah sebenarnya bilangan acak itu, dan bagaimana komputer dapat membangkitkannya? dan yang lebih penting adalah, sulit atau mudah kah membangkitkan bilangan acak ini?

Keywords—P, NP, deterministik, bilangan acak semu

I. PENDAHULUAN

Pada *video game* yang kita mainkan hampir selalu ada momen dimana kita membutuhkan "keacakan". Seperti pada saat melakukan *gacha*, atau demi menentukan *item* apa yang dijatuhkan lawan yang tumbang. Namun bagaimana sebenarnya komputer mensimulasikan keacakan tersebut? Bukankah komputer kita bersifat deterministik, sehingga tidak ada langkah acak yang ditimbulkan?

Makalah ini akan membahas sedikit tentang bagaimana komputer membangkitkan bilangan acak, yang disebut bilangan acak semu. Adapun tujuan dari makalah ini adalah mengkaji keterkaitan pembangkit bilangan acak semu dan kaitannya dengan permasalahan yang begitu populer di ilmu komputer: P vs. NP.

Struktur makalah ini didominasi oleh landasan teori, dengan pembahasan dan kesimpulan yang dijadikan satu bab tersendiri.

II. LANDASAN TEORI

2.1 Mesin Turing

Automata adalah sebuah mesin abstrak yang menerima suatu himpunan masukan dan menghasilkan sebuah keluaran dari suatu himpunan keluaran, dimana setiap satuan masukan akan mengubah keadaan terkini dari automata terkait. Automata disebut mesin abstrak karena pada dasarnya ia adalah sebuah konsep. Namun terdapat perbandingan dari automata di dunia nyata, seperti mesin penjual otomatis dan komputer yang kita gunakan.

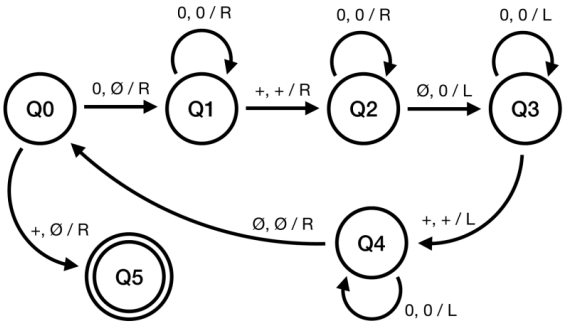
Penjelasan lebih mendalam mengenai automata dan jenis-jenisnya telah dibahas pada perkuliahan Teori Bahasa Formal dan Automata (*Formal Languages and Autoamata Theory*). Diantara jenis-jenis automata ialah *Finite-state machine*(FSM) dan Mesin Turing.

Finite State Automata adalah kelas automata yang tidak dapat "mengingat" masukan yang telah diberikan ataupun mengubah karakteristik dari keaadan yang dimiliki. Sebagai contoh, sebuah elevator di pusat perbelanjaan tidak perlu mengingat lantai berapa saja yang telah dikunjungi dan tidak perlu juga "mengubah" caranya beroperasi di setiap lantai(lantai dalam kasus ini sebanding dengan keadaan pada FSM). Pada FSM terdapat DFA atau *Deterministic Finite Automata*, yaitu FSM yang jika diberi masukan pada setiap keadaan maka keadaan berikutnya dapat diprediksi secara pasti. Terdapat pula NFA, atau *Non-Deterministic Finite Automata* yang tidak selalu bisa diprediksi apa keadaan berikutnya diketahui keadaan sekarang dan masukannya.

Adapun jenis automata yang terkait topik makalah ini adalah Mesin Turing. Tidak seperti FSM, Mesin Turing dapat "mengingat" masukan-masukan apa saja yang telah dijalankan dan mengubah karakteristik dari keadaan-keadaan yang ada di dalamnya.

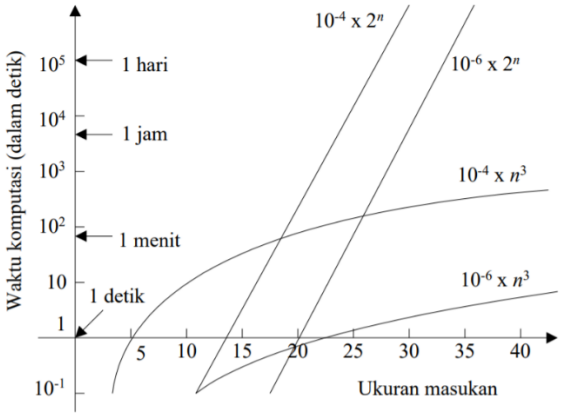
Mesin Turing pertama kali digagaskan oleh Alan Turing pada tahun 1936 untuk memberi solusi pada *Entscheidungsproblem*, sebuah masalah matematika yang digagaskan oleh matematikawan Hilbert di awal tahun 1900an. Sama seperti FSM, terdapat pula mesin turing *Deterministic* dan *Non-Deterministic*.

Walaupun terkesan simpel, mesin turing dapat mensimulasikan seluruh perhitungan yang dapat dilakukan oleh komputer modern. Oleh karena itu segala teori dalam ilmu komputer dapat dimodelkan dalam mesin turing. Atas hal tersebut pengetahuan soal esin turing akan menjadi sangat berguna dalam membahas persoalan dalam ilmu komputer.



Gambar 2.1 Contoh mesin turing deterministik

(https://medium.com/background-thread/whats-a-turing-machine-and-why-does-it-matter-1cd1b4606c6a)



Gambar 2.2 ilustrasi kompleksitas algoritma (http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf)

2.2 Kompleksitas Algoritma

Pembuatan sebuah algoritma tidak hanya harus memenuhi spesifikasi(kebutuhan) akan tetapi juga harus memiliki efektivitas tinggi. Algoritma yang mangkus(efektif) berarti lebih serba guna pada berbagai mesin dan prosesor, sebuah sifat yang penting dalam dunia informatika.

Efektivitas algoritma dinilai dari memori dan waktu yang digunakan, semakin sedikit memori dan waktu terpakai maka algoritma dinilai lebih efektif.

Kebutuhan memori dan waktu hanya bergantung pada masukan algoritma, yang dikenal dengan n, semakin besar nilai masukan maka memori dan waktu yang dibutuhkan untuk menjalankan proses lebih besar pula. Efektivitas algoritma tidak bergantung pada prosesor maupun *compiler*.

Kompleksitas algoritma adalah besaran abstrak yang digunakan untuk mengukur waktu dan ruang terpakai dalam sebuah program. Terdapat dua jenis kompleksitas algoritma, yakni kompleksitas ruang(memori) dan konpleksitas waktu. Dengan menggunakan besaran kompleksitas waktu dan ruang algoritma, dapat ditentukan laju peningkatan waktu yang diperlukan algoritma dengan meningkatnya ukuran masukan(n).

a. Kompleksitas Ruang

Kompleksitas ruang, S(n), adalah besaran yang digunakan untuk mengukur jumlah memori yang dipakai algoritma. Hal tersebut berarti jumlah memori terbanyak yang dibutuhkan suatu algoritma pada tiap-tiap waktu algoritma dijalankan. Sebagaimana dengan kompleksitas waktu, perhatian utama kompleksitas ruang adalah bagaimana kebutuhan memori bertambah seiring bertambahnya input.

b. Kompleksitas Waktu

Kompleksitas waktu, T(n), berguna untuk mengukur waktu yang terpakai selama algoritma bekerja tanpa menghiraukan penggunaan memori. Besaran ini dihitung berdasarkan jumlah operasi yang dilakukan satu algoritma, operasi yang dimaksud ialah: baca, tulis, aritmetika, *assignment*, perbandingan, dan pengaksesan/pemanggilan.

Sama halnya dengan kompleksitas ruang, terdapat tiga jenis kompleksitas waktu, yaitu

- Tmax(n): kebutuhan waktu maksimum pada *worst-case scenario*. Contoh: tidak ada elemen yang dicari pada *sequential search*
- Tmin(n): waktu tercepat algoritma dapat terselesaikan. Contoh: elemen yang dicari berada pada awal array pada *sequential search*
- Tavg(n): waktu yang dibutuhkan pada umunya untuk suatu algoritma menyelesaikan proses.

Misalkan terdapat algoritma *Bubble Sort* dalam *pseudocode* sebagai berikut:

```
procedure BubbleSort(input/output a1, a2, ..., an : integer)
{ Mengurut larik A yang berisi n elemen integer sehingga terrut menaik }
Deklarasi
   i, j, temp : integer
Algoritma
   for i ← n – 1 downto 1 do
      for j ← 1 to i do
         if aj + 1 < aj then
            { pertukarkan aj dengan aj + 1 }
            temp ← aj
            aj ← aj + 1
            aj + 1 ←temp
         endif
      endfor
   endfor
```

Gambar 2.3 *Bubble Sort*
(http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf)

Untuk mencari jumlah operasi yang dilakukan perhitungan jumlah perbandingan pada *loop* terdalam:

for j ← 1 **to** i **do**
    {proses}

berarti dilakukan i kali proses di *loop* terdalam.
Sementara pada *loop* terluar:

for i←(n-1) **downto** 1 **do**
    {pengulangan dalam}

Dengan secara keseluruhan jumlah proses yang dilakukan oleh algoritma *Bubble Sort* adalah:

(n-1)+(n-2)+(n-3)+..+2+1 = n(n-1)/2 (deret aritmatik)

Jika diasumsikan proses pada pengulangan terdalam sebagai proses tunggal, maka kompleksitas waktu di atas adalah kasus terbaik, terburuk, dan rata-rata secara bersamaan. Hal ini disebabkan algoritma *Bubble Sort* tidak membedakan larik yang sudah terurut atau belum dalam menjalankan proses.

c. Kompleksitas Waktu Asimptotik

Sering kali waktu presisi dari sebuah proses kompleks tidak dipedulikan. Pada n yang kecil, waktu yang terpakai oleh tiap-tiap algoritma cenderung sama karena. Hal yang lebih penting adalah bagaimana *trend* penggunaan waktu algoritma berdasarkan perubahan masukan.

Sebagai contoh, sebuah algoritma memiliki kompleksitas waktu rata-rata sebagai berikut:

$$T(n) = n^2 + 2n\log(n) + 3$$

Waktu yang diperlukan oleh algoritma tersebut untuk melakukan proses pada masukan tertentu tidak terlalu penting. Yang lebih penting ialah bagaimana waktu yang dibutuhkan algoritma tersebut berubah seiring masukan bertambah besar. Notasi waktu algortima untuk jumlah n yang besar disebut kompleksitas waktu asimptotik.

Dalam kasus T(n) di atas, suku yang dominan terhadap perubahan n adalah n^2(pada n yang cukup besar n^2 saja tidak terlalu berbeda dengan T(n)). Fakta tersebut dinyatakan dalam notasi Big-O yakni:

$$T(n) = O(n^2)$$

2.3 P vs. NP

Kompleksitas waktu yang dibutuhkan untuk mencari sebuah objek dari kumpulannya ialah O(n) dengan algoritma biasa atau O(log(n)) jika menggunakan *Binary Search*. Operasi pengurutan larik dengan algoritma *Divide and Conquer* memiliki kompleksitas waktu O(n log(n)). Kedua kelas kompleksitas tersebut masuk ke dalam kelas "*Polynomial Time*", yang berarti permasalahan terkait dapat diselesaikan dalam waktu yang relatif "cepat" bergantung ukuran masukan. Kompleksitas yang termasuk lelas "*Polynomial Time*" adalah yang memiliki notasi Big-O dalam bentuk polinomial atau logaritma.

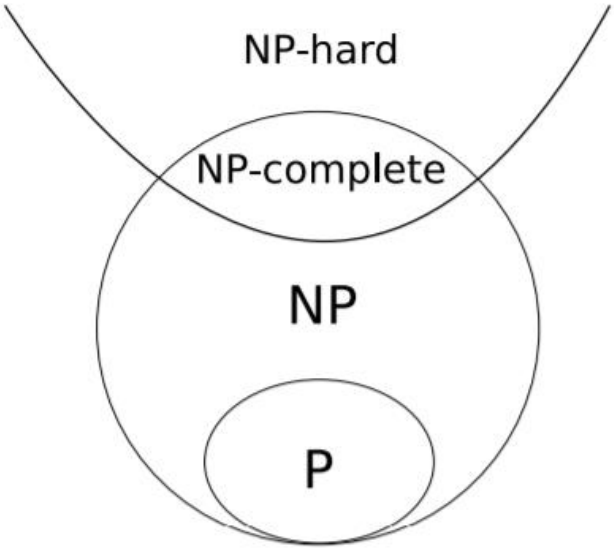
Adapun beberapa permasalahan yang lebih kompleks seperti mewarnai graf planar dengan 3 warna dengan syarat tidak ada simpul bertetangga dengan warna yang sama(*graph colouring problem*), memilih beberapa bilangan dari sebuah himpunan sehingga jumlahnya adalah sebuah **T**(*sum of subset problem*), atau menentukan ada tidaknya siklus hamiltonian pada sebuah graf(*Hamiltonian Path Problem*) adalah permasalahan "sulit" yang solusi-solusinya memiliki kompleksitas bukan termasuk kelas "*Polynomial Time"*.

Kelas masalah yang tidak dapat diselesaikan dalam waktu cepat tersebut termasuk kelas "*Nondeterministic Polynomial Time*", masalah-masalah yang dianggap "sulit". Nama *Nondeterministic* mengacu kepada *Nondeterministic Turing Machine*, seperti yang diketahui merupakan mesin turing yang perpindahan antar *state*-nya tidak dapat diprediksi. Fakta ini menyatakan bahwa mesin turing nondeterministic dapat menyelesaikan masalah-masalah "sulit" dalam waktu yang singkat, hanya saja tidak ada ekivalen dari mesin turing nondeterministik di dunia nyata.

Atas dasar kelas "*Polynomial Time*" dan "*Nondeterministic Polynomial Time*" inilah bangkit persoalan P vs. NP. P vs. NP mempertanyakan apakah setiap permasalahan yang saat ini tidak dapat diselesaikan dalam waktu polinomial dengan mesin turing memiliki algoritma yang dapat menyelesaikannya dalam waktu polinomial(mengindikasikan bahwa P = NP). Tidak semua permasalahan yang tidak dapat diselesaikan dalam waktu polinomial berarti termasuk ke dalam kategori NP, hanya saja mungkin belum ditemukan algoritma yang mangkus.

a. NP-Complete

Suatu persoalan **x** masuk ke dalam kelas NP-Complete jika **x** termasuk ke dalam NP dan seluruh persoalan yang termasuk ke dalam NP dapat ditransformasi ke **x** dalam waktu polinomial. Dengan kata lain, seluruh persoalan yang termasuk ke dalam NP memiliki letak kesamaan yang menjadikan persoalan tersebut "sulit". Hal yang menjadikan NP-Complete beigtu penting adalah jika satu saja persoalan NP-Complete dapat diselesaikan dalam waktu polinomial, maka seluruh persoalan NP dapat diselesaikan dalam waktu polinomial pula(mengindikasikan bahwa P = NP).



Gambar 2.4 Diagram yang menggambarkan P ≠ NP
(https://www.scottaaronson.com/papers/pnp.pdf)

b. Mengapa P vs. NP begitu penting

Ranah pertanyaan "Apakah P = NP atau sebaliknya?" tidak hanya terbatas ke perkembangan ilmu komputer dan pengkategorian masalah. Ambil contoh kriptografi, seluruh kata sandi yang ada sebenarnya dapat dipecahkan, hanya saja waktu yang dibutuhkan untuk memcahkan sandi dengan panjang 8 huruf saja sudah mencapai 92 tahun. Karena pemecahan sandi masuk ke kategori NP, jika saja P = NP maka seluruh kata sandi yang digunakan untuk mengamankan akun sosial media kita akan menjadi tidak berguna.

Pada kenyataannya, banyak ilmuwan komputer dan matematikawan yang beranggapan P pasti merupakan *proper subset* dari NP(P ≠ NP). Akan tetapi selama tidak ada bukti tertulis dan langkah konkret yang menunjukkannya, klaim tersebut tidak dapat dianggap valid. Membuktikan P ≠ NP mirip dengan membuktikan bahwa perjalanan waktu itu tidak mungkin, keduanya paling masuk akal secara nalar, namun tidak dapat dibuktikan.

Status hubungan P dan NP begitu unik. Permasalahan ini termasuk ke dalam "Millennium Problems" dimana penyelesainya akan membawa hadiah 1.000.000 dollar amerika yang disponsori oleh Institut Matematika Clay. Yang paling menarik adalah, karena ketujuh masalah ini tergolong ke dalam NP(termasuk P vs. NP sendiri), maka jika P bisa ditunjukkan sama dengan NP semua persoalan millennium akan bisa dipecahkan dengan komputer.

2.4 Pembangkit Bilangan Acak Semu

Bilangan acak adalah bilangan yang dihasilkan secara acak. Suatu Bilangan dinyatakan sebagai billangan acak apabila kemunculannya hanya dapat ditebak dengan kemungkinan. Pada pemilihan bilangan acak dari suatu himpunan, seluruh elemen memiliki kemungkinan yang sama untuk terpilih. Contoh pengambilan bilangan acak adalah dengan menggunakan dadu.

Komputer sebagai mesin *deterministic*, yakni dapat ditebak seluruh keluaran berdasarkan masukan yang diberikan, tidak dapat menghasilkan bilangan acak sejati. Hal ini disebabkan seluruh masukan yang sama akan menghasilan keluaran yang sama pula.

Adapun bilangan acak semu adalah bilangan acak yang tidak memenuhi seluruh kriteria bilangan acak asli. Bilangan acak semu umumnya dihitung dengan komputer. Akibat komputer yang tidak dapat mensimulasikan ketidakpastian, pembangkitan bilangan acak semu akan menghasilkan hasil yang sama pada periode tertenu.

Pembangkit Bilangan Acak Semu(PBAS) adalah algoritma yang memberikan keluaran berupa bilangan yang memiliki sifat kemunculan mirip bilangan acak asli. Bilangan yang dihasilkan oleh PBAS tentu bukan merupakan bilangan acak asli, karena kemunculan keluaran bergantung pada bilangan pancingan awal(*seed*). Walaupun komputer mampu menghasilkan bilangan acak asli dengan bantuan perangkat keras(contoh: bangkitnya transistor dalam prosesor yang tidak dapat diduga), PBAS tetap umum digunakan dalam program komputer karena kecepatan proses dan nilai yang dihasilkan mumpuni.

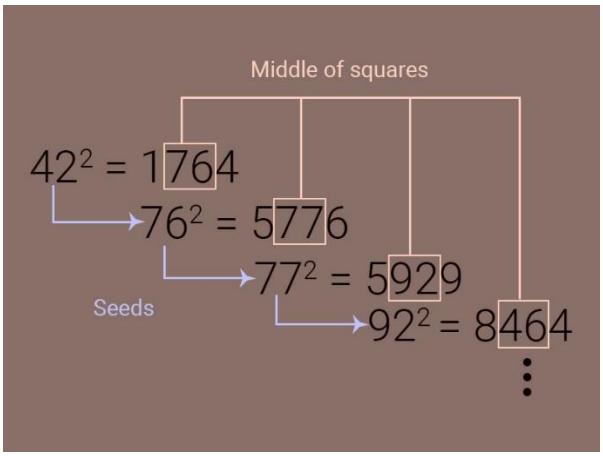
Badan Federal Kemanan Informasi Jerman memberikan 4 kriteria kualitas dari sebuah PBAS, antara lain:

1. Kemungkinan yang tinggi barisan bilangan keluaran PBAS berbeda tiap-tiap iterasi.
2. Tidak dapat dibedakan dengan bilangan acak asli berdasarkan uji statistik tertentu.
3. Tidak mungkin bagi manusia untuk menebak dengan pasti bilangan yang dihasilkan PBAS beberapa keluaran ke depan secara terurut.
4. Tidak mungkin bagi manusia untuk menebak dengan pasti bilangan yang dihasilkan PBAS sebelumnya secara terutut.

Standar yang memenuhi penggunaan program adalah standar 3 atau 4 saja.

PBAS awalnya diajukan oleh Matematikawan John von Neumann pada tahun 1946. Algoritma yang dia ciptakan dikenal sebagai "*middle-squared method*" atau "metode tengah-kuadrat". Prosesnya adalah sebagai berikut; ambil pancingan sebanyak x digit, lalu dikuadratkan, tulis hasil perhitungan sebagai 2x digit dan ambil x digit yang paling tengah dari hasil, bilangan tengah tersebut adalah hasil dan digunakan sebagai masukan iterasi berikutnya. Algoritma ini tidak cukup dipakai pada masa ini karena bilangan akan mengulang terlalu cepat, sebagai contoh bilangan 0000.

Neumann merasa tidak perlu melakukan perbaikan terlalu mendalam terhadap algoritma yang ia ciptakan karena saat itu metode ini mencukupi pekerjaannya.



Gambar 2.5 Ilustrasi algoritma Neumann pada 2 digit angka (https://svijaykoushik.github.io/blog/2019/10/04/three-awesome-ways-to-generate-random-number-in-javascript/)

Berikut beberapa contoh algoritma PBAS:

a. Metode Linear Kongruen

Metode linear kongruen adalah algoritma PBAS, termasuk PBAS baik yang pertama ditemukan. LCM memanfaatkan model linier untuk membangkitkan bilangan acak yang didefinisikan dengan:

$$X_{n+1} = (aX_n + c) \bmod m$$

Dengan ketentuan sebagai berikut:

- m > 0
- 0 < a < m
- 0 ≤ c < m
- 0 ≤ X_0 < m
- PBB(c,m) = 1
- a-1 dapat dibagi faktor prima dari m
- a-1 kelipatan 4 jika m kelipatan 4
- a yang sangat besar meningkatkan efektivitas algoritma.

b. Mersenne Twister

Mersenne Twister adalah PBAS yang paling banyak diimplementasikan dalam dunia komputer. Nama PBAS ini diambil karena periode bilangan hasil adalah bilangan prima Mersenne pilihan. Algoritma ini dibuat tahun 1997 untuk menutupi kelemahan-kelemahan dari PBAS sebelumnya.

Mersenne Twister digunakan sebagai PBAS standar dalam berbagai bahasa pemrograman, seperti Python, R, Matlab, dan PHP.

```
// Create a length n array to store the state of the generator
int[0..n-1] MT
int index := n+1
const int lower_mask = (1 << r) - 1 // That is, the binary number of r 1's
const int upper_mask = lowest w bits of (not lower_mask)

// Initialize the generator from a seed
function seed_mt(int seed) {
    index := n
    MT[0] := seed
    for i from 1 to (n - 1) { // loop over each element
        MT[i] := lowest w bits of (f * (MT[i-1] xor (MT[i-1] >> (w-2))) + i)
    }
}

// Extract a tempered value based on MT[index]
// calling twist() every n numbers
function extract_number() {
    if index >= n {
        if index > n {
          error "Generator was never seeded"
          // Alternatively, seed with constant value; 5489 is used in reference C code[53]
        }
        twist()
    }

    int y := MT[index]
    y := y xor ((y >> u) and d)
    y := y xor ((y << s) and b)
    y := y xor ((y << t) and c)
    y := y xor (y >> l)

    index := index + 1
    return lowest w bits of (y)
}

// Generate the next n values from the series x_i
function twist() {
    for i from 0 to (n-1) {
        int x := (MT[i] and upper_mask)
                + (MT[(i+1) mod n] and lower_mask)
        int xA := x >> 1
        if (x mod 2) != 0 { // lowest bit of x is 1
            xA := xA xor a
        }
        MT[i] := MT[(i + m) mod n] xor xA
    }
    index := 0
}
```

Gambar 2.6 *Pseudocode* dari Mersenne Twister (https://en.wikipedia.org/wiki/Mersenne_Twister)

III. Pembahasan dan Kesimpulan

Pada titik ini kita sudah mengetahui apa itu Mesin Turing, P dan NP, serta bilangan acak dan algoritma yang membangkitkannya. Karena komputer sebagai mesin deterministik tidak dapat menghasilkan bilangan acak asli, maka timbul pertanyaan: "Adakah keterkaitan antara pembangkit bilangan acak semu dengan masalah P vs. NP?". Pertanyaan itulah yang menjadi tema pada makalah ini.

Walaupun algoritma pembangkit bilangan acak semu pada awalnya merupakan algoritma yang lemah, dimana pola pembangkitan dapat langsung tertebak, saat ini terdapat banyak sekali algoritma yang menghasilkan deretan bilangan acak semu yang sangat mirip dengan aslinya. Seperti yang kita ketahui, pembangkitan bilangan acak semu adalah proses yang termasuk ke dalam kelas P, sementara proses menebak atau *derandomize* masuk ke dalam kelas NP. Dengan demikian, jika dapat dibuktikan bahwa *derandomization* tidak dapat menjadi kelas P maka P ≠ NP. Sebaliknya, jika terdapat bukti bahwa seluruh proses *derandomization* dapat menggunakan waktu polinomial maka P = NP.

Video Link Youtube

intip.in/videoStimaDwinta

UCAPAN TERIMA KASIH

Puji syukur kehadirat Allah SWT. penulis dapat menyelasaikan makalah ini sebagai pelengkap penelaian mata kuliah IF2211 Strategi Algoritma. Sholawat dan salam penulis sampaikan kepada Rasulullah SAW. beserta para Rasul-Rasul terdahulu, keluarga, dan sahabatnya.

Terima kasih saya hanturkan kepada:

1. Pak Rila Mandala, dosen yang mengajar mata kuliah IF2211 kelas saya.
2. Para Asisten Dosen, semoga kita dapat segera menjalin silaturahmi di Labtek V.
3. Teman-teman Async, teman seperjuangan perkuliahan tingkat 2 IF ini.
4. Teman-teman SMA, para sahabat yang menjadi tempat melepas jenuh perkuliahan.
5. Kedua Orang Tua saya, yang jasanya lebih dari membiayai kuliah, namun segala ilmu, kasih sayang, candaan, emosi, dan kebersamaan. Semoga Ayah dan Ibu menjadi orang yang diridhoi Allah dan saya menjadi anak yang sholeh.
6. Orang-orang yang spesial yang belum sempat saya sebut

DAFTAR PUSTAKA

[1] Daniel G.G. (2013) Deterministic and Nondeterministic Turing Machine. In: Runehov A.L.C., Oviedo L. (eds) Encyclopedia of Sciences and Religions. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-8265-8_200983

[2] Cambridge, *Intrudoction: What is a Turing Machine?*, https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/one.html#:~:text=A%20Turing%20machine%20is%20a,mathematician%20Alan%20Turing%20in%201936.&text=Above%20is%20a%20very%20simple,other%20form%20of%20data%20storage.

[3] https://cstheory.stackexchange.com/questions/2839/p-vs-np-and-pseudorandom-bit-generators

[4] https://cs.stackexchange.com/questions/16315/difference-between-a-turing-machine-and-a-finite-state-machine

[5] Scott, A. *P ? = NP*, https://www.scottaaronson.com/papers/pnp.pdf

[6] Stanford, Basics of Automata Theory, https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html#:~:text=Therefore%2C%20the%20major%20difference%20between,simulating%20computer%20execution%20and%20storage

[7] Widgerson, A. *Randomness, Pseuodorandomness and Derandomization*. *http://www.cs.toronto.edu/~ledt/papers/Avi/L1.pdf*

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, 26 April 2021

Ttd,

Nama dan NIM

Mohammad Dwinta Harits Cahyana/13519041