Sieve of Eratosthenes vs Naive Prime Searching

Juro Sutantra 13517113
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517113@std.stei.itb.ac.id

***Abstract*—Makalah ini menjelaskan tentang bagaimana sifat bilangan prima dan bagaimana identifikasinya. Disamping algoritma manusia, cara identifikasi oleh komputer juga disajikan dalam dua algoritma yaitu sieve of Eratosthenes dan Naïve Prime Searching. Di dalam pembahasan akan dielaborasikan mengenai waktu yang digunakan dan ditunjukkan mengenai penerapan algoritma ini untuk RSA. Dituliskan juga bagaimana efek dari kompleksitas kedalam komputasi RSA.**

Kata Kunci—Prima, Faktor, RSA

I. PENDAHULUAN

Bilangan Prima sudah menjadi pengetahuan bagi semua orang di dunia ini. Sejak kecil kita sudah diajarkan apa itu bilangan prima dan bagaimana sifat-sifatnya. Ketika kita diminta untuk menuliskan bilangan-bilangan prima antara 1-100, dengan cepat dan lincah kita bisa menuliskannya. Mungkin karena hafal, mungkin juga karena kemampuan berhitung dan memfaktorkan yang sangat cepat. Setelah mampu mengidentifikasi bilangan prima, biasanya kita akan langsung diajarkan untuk menggunakan bilangan prima pada perhitungan yang lebih kompleks lagi. Sebagai contoh Faktor Persekutuan Terbesar dan Kelipatan Persekutuan terkecil. Dua komputasi tersebut membutuhkan konsep bilangan prima untuk menentukan hasilnya. Namun biasanya kita hanya menghitung bilangan prima yang relatif kecil, sekitar 1-50 saja. Kemudian pertanyaan muncul, bagaimana jika dibutuhkan perhitungan bilangan prima yang sangat besar. Sebagai contoh, kita harus menyederhanakan akar dari 17534. Oleh karena kebutuhan tersebut, mencari faktor dari sebuah angka menjadi kebutuhan yang cukup penting bagi keberjalanan hidup manusia.

Sumber :

Kebutuhan akan teknologi akhir-akhir ini menjadi sangat tinggi dan dibutuhkan bagi semua unsur penyusun hidup manusia. Semua hal bisa diakses melalui alat elektronik, komputer, Smartphone, laptop, dan lain-lain. Internet juga mendukung perkembangan kebutuhan teknologi. Industri 4.0 merupakan sebutan orang-orang mengenai fenomena tersebut. Oleh sebab itu, kebutuhan akan keamanan meningkat drastis, karena semua orang butuh melindungi data yang diunggah ke internet, ataupun menjaga kerahasiaan dari suatu data yang sangat penting. Untuk itu, diciptakanlah sebuah teknologi bernama enkripsi. Enkripsi ini berguna untuk menjaga data dengan cara mengubah format data melalui sebuah metode atau teknik yang digunakan. Sebagai contoh, cipher mengubah teks dengan "menggeser" huruf menjadi berapa di sebelahnya. Misalkan dari huruf a diubah menjadi b dan b menjadi c dan seterusnya. Sehingga sebuah kata aku akan dienkripsi menjadi blv. Pada akhirnya ketika orang membaca hasil enkripsi tersebut, tidak akan ada yang bisa mengerti maksud dari teks terkait. Berangkat dari fenomena tersebut dibutuhkan sebuah teknik untuk melakukan dekripsi. Ketika seseorang memiliki kunci dari enkripsi, maka melakukan dekripsi akan menjadi mudah. Tujuan dari enkripsi ini biasanya adalah mencari suatu metode yang sulit untuk ditebak kunci dekripsinya. Sehingga kerahasiaan pesan menjadi meningkat.

Gambar ilustrasi enkripsi

sumber https://www.dictio.id/t/bagaimanakah-algoritma-enkripsi-yang-aman/13133

Bilangan Prima biasanya banyak digunakan untuk melakukan dekripsi dan enkripsi, karena biasanya ketika bilangan prima diterapkan dalam enkripsi, terbentuk sebuah pesan baru tidak bermakna yang tidak memiliki pola. Sehingga untuk menentukan kunci tidak bisa semata-mata hanya di

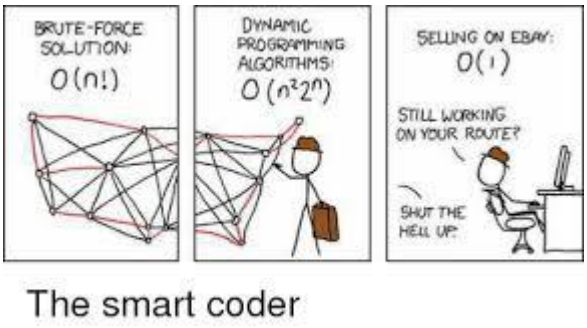
analisis dari pesan yang terbentuk. Salah satu Algoritma terkenal untuk kriptografi ini adalah RSA. Algoritma ini ternyata menggunakan bilangan prima untuk mendapatkan kuncinya. Tentu saja tidak semata-mata bilangan prima biasa tetapi ada perhitungan-perhitungan yang harus dilakukan juga untuk bisa mendapatkannya.

Komputasi dan algoritma mencari faktor dari prima atau menentukan sebuah bilangan prima atau bukan menjadi sangat dibutuhkan untuk membantu melakukan komputasi RSA ini. sehingga banyak juga dikembangkan algoritma-algoritma efektif dan paling efisien dalam menyelesaikan task tersebut. karena domain angka yang digunakan dalam bidang kriptografi itu sangat besar dan akan memakan waktu yang cukup lama ketika dikerjakan oleh manusia secara manual. Pada makalah ini, akan dibahas sedikit mengenai algoritma yang cukup tua dan cukup terkenal dalam menyelesaikan pencarian bilangan prima. Untuk memberi pembanding akan dibahas pula mengenai pendekatan yang naïf yaitu menggunakan brute force biasa.

II. TEORI DASAR

A. *Dynamic Programming*

Dynamic Programming merupakan suatu strategi algoritma untuk mendekati berbagai masalah komputasi jaman sekarang. Dynamic Programming atau yang biasa dikenal sebagai DP, biasanya menggunakan rekursif dalam implementasinya. Namun tidak menutup kemungkinan DP diimplementasikan tanpa menggunakan rekursif. Tetapi pada intinya karakteristik utama dari DP adalah menggunakan solusi optimum pada tahap sebelumnya untuk mendapatkan solusi pada tahap berikutnya. Biasanya karakter tersebut muncul ketika terdapat caching/penyimpanan hasil tiap tahap.



Sumber : https://me.me/i/brute-force-solution-dynamic-programming-algorithms-selung-on-ebay-o-n-17661475

Secara formal berikut tujuan dari DP yaitu menemukan solusi untuk masalah yang diberikan, dengan mengoptimalkan kuantitas Q yang diinginkan (fungsi objektif); misalnya, ingin memaksimalkan laba atau meminimalkan biaya. Algoritma bekerja dengan menggeneralisasi masalah asli. Lebih khusus, DP bekerja dengan menciptakan berbagai masalah terkait tetapi lebih sederhana, kemudian menemukan nilai Q yang optimal untuk masing-masing masalah ini. lalu menghitung nilai untuk masalah yang lebih rumit dengan menggunakan nilai yang sudah dihitung pada masalah yang lebih mudah. Ketika kita selesai, nilai optimal Q untuk masalah asli dapat dengan mudah dihitung dari satu atau lebih nilai dalam array. Setelah itu nilai-nilai pada array akan digunakan untuk menghitung solusi bagi masalah asli dimana hasilnya akan menjadi Optimal untuk Q yang diinginkan. DP akan selalu menyajikan algoritma pemrograman dinamis dalam 4 langkah berikut.

Step 1: Jelaskan larik (atau larik) nilai yang ingin Anda hitung. (Jangan katakan bagaimana cara menghitungnya, tetapi jelaskan apa yang ingin Anda hitung.) Katakan bagaimana menggunakan elemen-elemen tertentu dari array ini untuk menghitung nilai optimal untuk masalah awal.

Step 2: Berikan pengulangan yang menghubungkan beberapa nilai dalam array ke nilai lain dalam array; untuk entri yang paling sederhana, perulangan harus mengatakan bagaimana cara menghitung nilai-nilai mereka dari awal. Kemudian (kecuali perulangan itu benar benar) membenarkan atau membuktikan bahwa perulangan itu benar.

Step 3: Berikan program tingkat tinggi untuk menghitung nilai array, menggunakan pengulangan di atas. Perhatikan bahwa seseorang menghitung nilai-nilai ini dengan cara bottom-up, menggunakan nilai-nilai yang telah dihitung untuk menghitung nilai-nilai baru. (Seseorang tidak menghitung nilai-nilai secara rekursif, karena ini biasanya akan menyebabkan banyak nilai dihitung berulang-ulang, menghasilkan algoritma yang sangat tidak efisien.) Biasanya langkah ini sangat mudah dilakukan, menggunakan perulangan dari Langkah 2. Kadang-kadang seseorang akan juga menghitung nilai-nilai untuk array bantu, untuk membuat perhitungan solusi pada Langkah 4 lebih efisien.

Step 4: Tunjukkan bagaimana menggunakan nilai-nilai dalam array (dihitung pada Langkah 3) untuk menghitung solusi optimal untuk masalah aslinya. Biasanya seseorang akan menggunakan pengulangan dari Langkah 2 untuk melakukan ini.

B. *Brute Force*

Sumber : https://www.alamy.com/3d-illustration-of-brute-force-solution-script-with-pointing-hand-image155660217.html

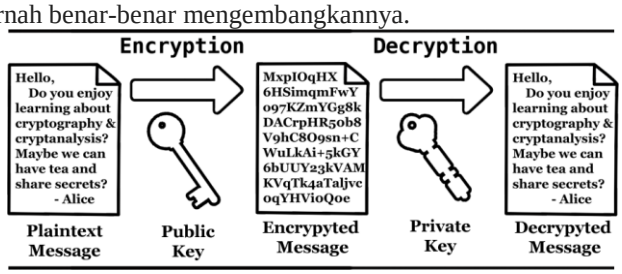
Brute Force biasa disebut sebagai algoritma pemrograman yang naïf karena mendekati sebuah permasalahan sesuai dengan definisi masalahnya atau spesifikasi. Biasanya brute

force ini memberikan solusi yang kurang optimal, karena mendekati berdasarkan apa yang didefinisikan. Bisa saja dia boros memori atau boros waktu. Namun algoritma ini menjamin suatu masalah yang memiliki solusi pasti ditemukan solusinya jika di dekati dengan metode ini. oleh sebab itu, algoritma ini selalu dipertahankan, karena untuk menyelesaikan suatu masalah biasanya didekati dengan brute force terlebih dahulu, baru dikembangkan dengan strategi strategi yang lainnya.

Brute force biasanya bisa diklasifikasikan kedalam dua permasalahan besar, yaitu permasalahan subset dan permasalahan permutasi. Permasalahan subset sebagai contoh seperti masalah penjadwalan, knapsack, dan TSP. untuk permasalahan ini dibagi menjadi 3 langkah yaitu generate subset evaluasi dan optimasi. Kompleksitas algoritma untuk brute force permasalahan subset ini adalah $O(n.2^n)$. untuk permasalahan permutasi kompleksitasnya O(n.n!) biasanya untuk mengatur kombinasi dari suatu tempat duduk dan banyak masalah lainnya. Disamping pendekatan itu, juga bisa disusun pendekatan yang cukup bebas sesuai dengan definisi bergantung pada masalah. Maka kompleksitasnya juga bergantung pada algoritma yang disusun.

C. Rivest-Shamir-Adleman Algorithm

Pada tahun 1978, Ron Rivest, Adi Shamir, dan Leonard Adleman memperkenalkan algoritma kriptografi, yang pada dasarnya untuk menggantikan algoritma National Bureau of Standards (NBS) yang kurang aman. Yang paling penting, RSA mengimplementasikan cryptosystem kunci publik, serta tanda tangan digital. RSA dimotivasi oleh menerbitkan karya Diffie dan Hellman dari beberapa tahun sebelumnya, yang menggambarkan gagasan semacam itu algoritma, tetapi tidak pernah benar-benar mengembangkannya.



Sumber : http://aircampgames.com/cryptological-systems-from-history/

Diperkenalkan pada saat era email diharapkan segera muncul, RSA diimplementasikan dua ide penting pertama enkripsi kunci publik, Gagasan ini menghilangkan perlunya "kurir" untuk mengirimkan kunci kepada penerima saluran aman lain sebelum mengirim pesan yang dimaksudkan semula. Di RSA, kunci enkripsi adalah publik, sedangkan kunci dekripsi tidak, jadi hanya kunci dekripsi yang benar yang dapat menguraikan pesan terenkripsi. Setiap orang memiliki kunci enkripsi dan dekripsi sendiri. Kunci harus dibuat cara kunci dekripsi tidak mudah disimpulkan dari kunci enkripsi publik. 2. Tanda tangan digital. Penerima mungkin perlu memverifikasi bahwa itu berasal dari pengirim (tanda tangan), dan bukan hanya berasal dari sana (otentikasi). Ini dilakukan dengan menggunakan kunci dekripsi pengirim, dan diverifikasi oleh siapa pun, menggunakan yang sesuai kunci enkripsi publik. Oleh karena itu tanda tangan tidak dapat dipalsukan. Juga, tidak ada penanda tangan yang kemudian dapat menyangkal telah menandatangani pesan.

Ini tidak hanya berguna untuk email, tetapi untuk transaksi elektronik dan transmisi lainnya sebagai transfer dana. Keamanan algoritma RSA telah divalidasi jauh, karena tidak ada upaya yang diketahui telah berhasil, sebagian besar karena kesulitan dengan angka besar n = pq, di mana p dan q adalah bilangan prima yang besar.

RSA diaplikasikan untuk transmisi dana elektronik juga. Informasi keuangan harus aman, dan cek dapat ditandatangani secara elektronik dengan RSA. Langkah-langkah selanjutnya harus diambil, seperti menerapkan nomor cek unik yang memungkinkan cek dengan nomor tertentu ini dapat ditransmisikan / diuangkan 3 hanya sekali.

Bahkan, sistem seperti itu dapat diterapkan pada sistem elektronik apa pun yang perlu memiliki kriptosistem diimplementasikan. Dalam makalah RSA 1978 mereka, penulis RSA meramalkan dunia email aman untuk berkembang dan agar RSA digunakan untuk mengenkripsi percakapan telepon langsung. Sekarang, hal-hal ini memang merupakan bagian dari lebih dari sekedar kehidupan sehari-hari karena RSA.

Perangkat enkripsi tidak boleh menjadi penyangga langsung antara terminal dan komunikasi saluran. Sebagai gantinya, itu harus menjadi subrutin perangkat keras yang dapat dieksekusi sesuai kebutuhan, karena mungkin perlu untuk dienkripsi / didekripsi dengan beberapa urutan kunci yang berbeda, untuk memastikan lebih banyak privasi dan / atau lebih banyak tanda tangan.

D. Bilangan Prima

Jika kita menulis a b maka kita katakan bahwa a adalah pembagi b. Salah satu metode yang biasa digunakan di sekolah dasar untuk menentukan pembagi suatu bilangan adalah menggunakan kertas berpetak dan menampilkan bilangan itu sebagai suatu persegipersegi panjang. Sebagai contoh, 12 dapat disajikan dengan menampilkan persegi-persegi panjang dengan susunan 1 baris 12 kolom, atau 2 baris 6 kolom, atau 3 baris 4 kolom. Dengan demikian 12 mempunyai 6 buah pembagi yang berbeda, yaitu 1, 2, 3, 4, 6, dan 12. Bagaimana dengan 7? Untuk 7, kita hanya dapat menampilkan persegi-persegi panjang dengan susunan 1 baris 7 kolom atau 7 baris 1 kolom. Dengan demikian 7 hanya mempunyai 2 pembagi yang berbeda, yaitu 1 dan 7.

1	2	3	4	5	6	7	8
1	2	4	6	16	12		24
	3	9	8		18		30
	5	25	10		20		
	7		14		28		
	11		15		32		
	13		21				
	17		22				
	19		26				
	23		27				
	29		33				
	31		34				
	37		35				

Tabel 1. Banyak Faktor Suatu Bilangan

Dari tabel 1 tampak bahwa 12 berada pada kolom ke enam karena 12 mempunyai 6 pembagi, dan 7 berada pada kolom ke dua karena 7 mempunyai 2 pembagi. Apakah anda melihat suatu pola yang terbentuk pada tabel di atas? Bilangan apa berikutnya pada kolom ke tiga? Bilangan itu adalah 49. Sekarang, bilangan-bilangan pada kolom ke dua akan menjadi pusat perhatian kita saat ini. Perlu diingat bahwa bilangan-bilangan itu mempunyai tepat dua pembagi, yaitu bilangan itu sendiri dan 1. Sebarang bilangan bulat positif yang mempunyai tepat dua pembagi positif berbeda disebut bilangan prima. Sebarang bilangan bulat lebih besar dari 1 yang mempunyai suatu faktor positif selain 1 dan dirinya sendiri disebut bilangan komposit.

III. PEMBAHASAN

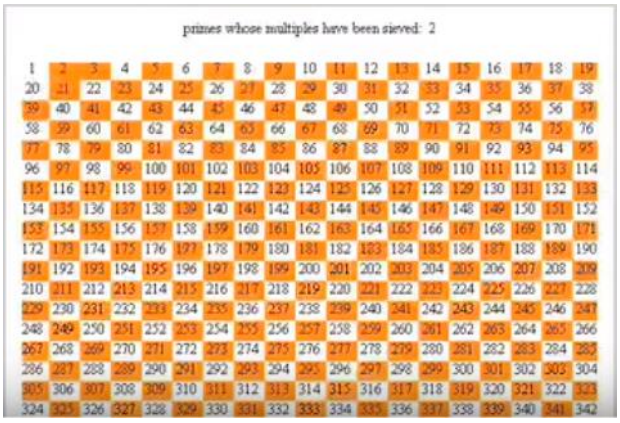
A. *Penjelasan Algoritma*

1) *Sieve of Eratosthenes*

Pada sieve of Erathosthenes pendekatan yang dilakukan adalah dengan strategi Dynamic Programming. Karena pada setiap langkahnya, akan diberikan solusi untuk mengupdate data-data lainnya. Dari teori dasar bilangan prima, kita tahu bahwa ada bilangan komposit dan bilangan prima. Dimana bilangan komposit adalah bilangan yang bukan prima. Dan sifat bilangan komposit adalah memiliki 1 atau lebih faktor yang merupakan bilangan prima selain 1. Sehingga bisa kita simpulkan bahwa kelipatan dari bilangan prima sudah pasti bilangan komposit. Algoritma yang digunakan pada Sieve of Eratosthenes itu terbagi menjadi 4 tahap besar. Yang pertama menyiapkan array untuk melakukan penyimpanan data sambil jalan ke tahap-tahap berikutnya. array yang paling tepat digunakan untuk algoritma ini adalah array dengan index dimulai dari 1 sampai ke n dan bertipe Boolean. Sebagai penanda apakah bilangan dengan index tertentu merupakan prima atau bukan. Pada awalnya array tersebut diinisialisasi dengan true yang menyatakan bilangan prima. Inisialisasi ini digunakan karena lebih mudah untuk mencari bilangan komposit dibandingkan bilangan prima. Langkah yang kedua adalah melakukan iterasi dengan mengupdate array yang sudah disiapkan menjadi baru. karena kita tahu untuk mencari suatu bilangan adalah prima atau tidak, kita hanya butuh mencarinya sampai ke lebih kecil akar bilangan tersebut. karena ketika tidak ada bilangan prima yang menyusun dia dan lebih kecil dari akarnya maka untuk mendapatkan angka tersebut haruslah angkanya sendiri dikalikan dengan 1. Sehingga iterasi dilakukan dari 2 hingga ke akar n. dimana pada setiap iterasi kita akan melakukan pemeriksaan apakah angka pada tahap ini prima atau tidak. Jika seandainya bilangan tersebut prima maka akan dilakukan iterasi mengupdate array dari kuadrat angka itu ke n dengan increment sebesar angka itu sendiri. Pada langkah ketiga lanjutkan iterasi pada tahap 2 ke angka berikutnya tapi dengan array yang sudah terupdate. Sehingga ada kemungkinan bilangan pada saat itu tidak prima, sehingga tidak dilakukan iterasi pengupdatean array. Pada langkah terakhir sudah ditemukan list bilangan prima untuk 1 sampai n bisa digunakan untuk berbagai macam hal, seperti menuliskan bilangan prima 1 sampai n. untuk mencari faktor bisa di modifikasi sedikit dari template algoritma yang sudah ada.

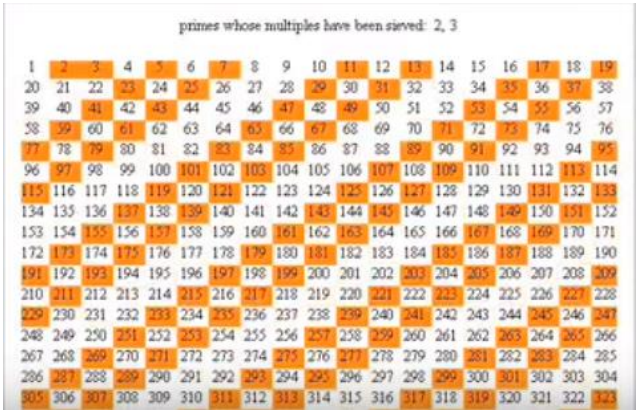
Apabila dilihat dan dikaji lebih dalam akan terdapat perbandingan array pada awal akan terjadi sebanyak akar n kali. Dan untuk beberapa kasus akan terjadi operasi mengubah array, yang memakan sebanyak n-i*i/i kali. Dimana i adalah angka pada saat itu. misalkan n = 50 dan angka pada langkah ini adalah 2, maka akan terdapat operasi penggantian isi array sebanyak kurang lebih 22 kali. Oleh sebab itu karena penggantian array itu bergantung pada situasi maka menurut teori big Oh akan didapat kompleksitas algoritma dari program ini adalah O(akar n)

Berikut adalah visualisasi program melalui gambar. Ketika berada pada angka 2 maka dimulai dari angka 4 akan mengganti kelipatan 2 menjadi bilangan bukan prima. Seperti gambar berikut



Sumber: https://www.youtube.com/watch?v=ysrIKCtP4fA

Ketika pada angka 3 maka update akan dimulai dari angka 9 yang merupakan kuadrat dari 3 sampai ke batas input n. serperti gambar berikut

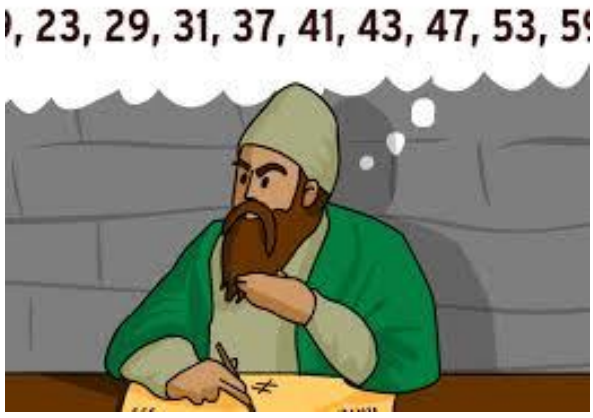


Sumber :https://www.youtube.com/watch?v=ysrIKCtP4fA

Pada algoritma ini akan dihasilkan secara langsung deret angka dari satu ke n yang merupakan bilangan prima dengan kompleksitas hanya sebatas O(akar n) sehingga ketika digunakan untuk program lain seperti menuliskan bilangan prima dari 1 ke n maka akan mendapatkan kompleksitas sebatas O(akar n) saja. Karena data telah siap dan hanya tinggal digunakan. Dan mengakses n kali relatif kecil jika dibandingkan O(akar n) sehingga bisa diabaikan

2) Naïve Prime Searching

Naive Prime Searching menggunakan pendekatan brute force untuk menyelesaikan permasalahan identifikasi bilangan prima. Dari pengertian dan definisi bilangan prima, bilangan diatas 1 yang hanya memiliki tepat 2 faktor, yaitu 1 dan dirinya sendiri beserta 1 itu sendiri. maka didapati pendekatan brute force yang didapat adalah melakukan iterasi dari angka 1 sampai ke n. dimana ketika modulo antara n dengan iterator = 0 akan dicatat jumlahnya. Barulah pada akhir iterasi kita cek apakah jumlah yang dicatat sama persis dengan dua atau tidak. Jika sama persis maka dia termasuk bilangan prima. Apabila ditelaah lebih dalam, maka bisa kita lihat bahwa jumlah perbandingan modulo yang dilakukan akan sejumlah n kali, karena dihitung mulai dari 1 sampai n. sehingga bisa kita simpulkan kompleksitas dari algoritma Naïve Searching ini adalah O(n).

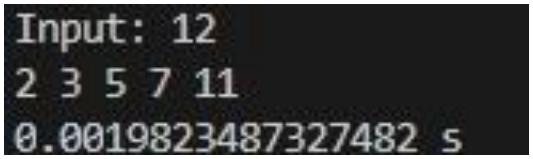


Sumber: https://www.brainpop.com/math/numbersandoperations/primenumbers/

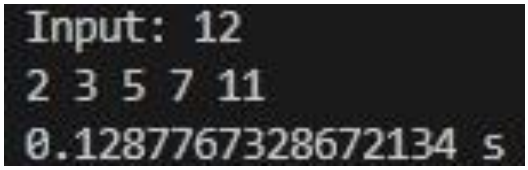
Dari naïve prime searching tersebut, biasanya algoritma dikembangkan menjadi beberapa program turunan seperti menuliskan bilangan prima antara 1 sampai n, untuk mencari suatu solusi rumpang dari deret prima, atau untuk melakukan komputasi FPB dan KPK yang bisa berguna bagi banyak komputasi lainnya. Kita ambil salah satu contoh yaitu menuliskan bilangan prima antara 1 sampai n. untuk mencapai tujuan itu jelas kita harus melakukan pengecekan apakah dia bilangan prima atau bukan. Jika dia bilangan prima maka dia akan disimpan atau diprint. Jika bukan akan dilewati. Secara naïf kita bisa menuliskan algoritma untuk mencari solusi masalah tersebut dengan menambahkan satu iterasi lagi di luar iterasi naïve prime searching. Sehingga pada setiap angka 1 sampai n akan dilakukan pengulangan untuk mengecek apakah dia prima atau bukan. Apabila dikaji dari segi perbandingan modulo, maka ketika n yang diinput adalah 10 maka akan ada 10*1 + 10*2 + 10*3 ... + 10*10. Bila didekati dengan metode Big oh maka didapat kompleksitas algoritma untuk kasus ini adalah $O(n^2)$

B. Hasil uji dan Analisis

Setelah dibuat algoritma, kedalam algoritma adapun salah satu kasus hasil uji yang didapat adalah sebagai berikut :



Gambar Hasil uji menggunakan algoritma Sieve of Eratosthenes waktu eksekusi dalam sekon



Gambar Hasil uji menggunakan algoritma Naïve Prime Search waktu eksekusi dalam sekon.

Bisa terlihat pada hasil uji terdapat selisih pada waktu eksekusi antara kedua algoritma yang kita bahas. Kasus yang saya ambil untuk eksperimen ini adalah menuliskan deret bilangan prima denan batas input sebagai n. selisih ini bisa muncul dikarenakan perbedaan kompleksitas pada kedua algoritma. Bisa kita lihat pada algoritma sieve of Eratosthenes kompleksitas algoritma nya adalah O(akar n) sedangkan pada naïve prime search kompleksitasnya adaah $O(n^2)$ sehingga terlihat perbedaan yang cukup jauh. Antara akar n dengan n kuadrat. Dari input yang dimasukkan n adalah 12 sehingga bila dihitung secara kasar kuadrat dua belas adalah 144 dan akar 12 adalah 3,4. Rasio dari 144 dan 12 adalah sekitar 42,35 sedangkan rasio dari waktu eksekusi algoritma sieve of Eratosthenes dengan naïve prime search adalah sekitar 64,96196. Sehingga terlihat bahwa kompleksitas tersebut cukup terbukti dari eksperimen tersebut. margin yang tinggi bisa disebabkan Karena data uji yang terlalu kecil. Karena pendekatan Big oh mengaproksimasi dengan keakuratan yang semakin tinggi untukdata yang semakin tinggi. Tetapi karena

kesulitan penyajian data pada makalah jadi saya memilih untuk tidak menampilkan dan mengelaborasi hasil uji tersebut.

C. Pemanfaatan untuk pencarian bilangan prima bernilai besar

Melalui kompleksitas yang didapatkan tidak mungkin untuk menghasilkan bilangan-bilangan prima dengan data yang sangat besar seperti angka dengan 6 digit bisa dilakukan menggunakan naïve prime search. Bayangkan saja kompleksitasnya akan menjadi 6 digit kuadrat yang berarti sekitar 12 digit mencapai trilliun. Bila diasumsikan satu kali proses membutuhkan sekitar 0.0001 detik maka dibutuhkan 1 miliar detik untuk eksekusi. Dibandingkan dengan sieve of Eratosthenes waktu yang dibutuhkan hanya sekitar 1 detik. Karena kompleksitas yang dibutuhkan hanya 3 digit saja. Sehingga sangat membantu untuk menghasilkan angka-angka yang prima berdigit besar.

Untuk kriptografi RSA yang sudah dibahas dalam teori dasar, dibutuhkan sebuah key yang merupakan bilangan prima dengan digit yang cukup besar. Bisa mencapai 6 digit atau lebih, untuk menjaga kerahasiaan dan agar tidak ada komputasi yang bisa mencari faktor dari private key yang merupakan hasil kali dari dua public key tersebut. untuk mencari public key itu kita sudah pasti membutuhkan beberapa data bilangan prima dengan digit tertentu. Oleh sebab itu bila dilakukan Naïve Prime search akan memakan banyak waktu karena kompleksitas $O(n^2)$ sangat mahal. Sehingga solusi satu-satunya adalah menerapkan algoritma sieve of Eratosthenes pada pencarian key untuk Algoritma RSA. Ini merupakan salah satu penerapan yang cukup penting dari algoritma pencarian bilangan prima karena hampir semua data sekarang membutuhkan kriptografi, dan RSA tidak akan pernah mati, karena bilangan tidak akan ada batasnya. Ketika sudah ditemukan komputer untuk memecahkan 12 digit. Masih bisa kita generate angka dengan 13 digit menggunakan komposisi bilangan prima yang tepat.

IV. KESIMPULAN

sieve of Eratosthenes menjadi suatu algoritma yang akan selalu dibutuhkan karena kebutuhan akan bilangan prima akan terus dibutuhkan oleh algoritma RSA. Sehingga bidang ini cukup menarik dan layak untuk diteliti lebih dalam. Karena semakin lama kemampuan komputer terus meningkat. Sehingga RSA juga dituntut untuk terus bisa mencari bilangan prima dengan digit yang cukup besar untuk bisa mencari private key yang tidak bisa dipecahkan.

V. REFERENSI

[1] http://file.upi.edu/Direktori/FPMIPA/JUR._PEND._MATEMATIKA/196008301986031-SUFYANI_PRABAWANTO/Bilangan_Prima.pdf

[2] http://www.cs.mun.ca/~kol/courses/2711-w08/dynprog-2711.pdf

[3] http://web.mit.edu/15.053/www/AMP-Chapter-11.pdf

[4] https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf

[5] https://www.geeksforgeeks.org/segmented-sieve/

[6] https://math.stackexchange.com/questions/2500494/prime-factorization-using-the-sieve-of-eratosthenes

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 2019

Juro Sutantra
13517113