Bilangan Prima, Uji Keprimaan, dan Berbagai Aplikasi Bilangan Prima

Muhammad Furqan Habibi (13511002)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]furqan.habibi@students.itb.ac.id

***Abstrak*—Pada makalah ini akan dibahas mengenai salah satu obyek studi yang sangat penting di bidang teori bilangan, yaitu bilangan prima. Sebagai bilangan yang memiliki sifat keterbagian yang unik, bilangan prima memiliki aplikasi yang luas baik dalam ilmu matematika murni maupun terapan dalam dunia informatika. Oleh karena itu banyak usaha yang dilakukan oleh matematikawan / ilmuan komputer untuk mempelajari sifat-sifat bilangan ini dan pemanfaatan dari sifat-sfat tersebut. Di antara usaha tersebut adalah pengembangan teorema / algoritma baru yang dapat menentukan dengan pasti keprimaan suatu bilangan.**

Di sini akan dibahas mengenai definisi dan sifat bilangn prima, berbagai teori maupun algoritma pengujian keprimaan, serta berbagai aplikasinya di dalam maupun luar informatika.

***Kata Kunci*—Teori bilangan, bilangan prima, uji keprimaan, aplikasi bilangan prima.**

I. PENDAHULUAN

Beberapa dasawarsa terakhir ini, bilangan prima sebagai bilangan bulat yang memiliki sifat keterbagian yang unik mulai menemukan banyak aplikasi di dunia nyata. Hal ini tidak terlepas dari perkembangan komputer sebagai alat hitung yang mulai beralih menjadi perangkat wajib sehari-hari. Dari pemakaian intensif komputer tersebut mulai dirasakan pentingnya kemanan dan kerahasiaan data. Dari sinilah kemudian bilangan prima mulai dilirik sebagai solusi untuk menjaga kerahasiaan data terebut.

Kriptografi, ilmu yang mempelajari dan mempraktikkan teknik-teknik perahasiaan data, merupakan ilmu yang mulai berkembang pesat sejak kebutuhan akan kerahasiaan tersebut mulai dirasa mendesak. Di ilmu kriptografi ini dipelajari berbagai teknik / algoritma untuk meng-*enkripsi* data sehingga tidak dapat di pahami oleh orang lain selain orang yang seharusnya membaca data tersebut. Di ilmu kriptografi inilah keunikan sifat bilangan prima dimanfaatkan untuk membangkitkan "kunci" untuk merahasiakan data tersebut. Di bidang ini bilangan prima memegang peran yang sangat penting untuk menjaga kerahasiaan informasi.

Selain beberapa contoh di atas, masih banyak kasus penerapan bilangan prima lainnya. Di dalam matematika murni sendiri sangat banyak bidang kaji maupun teorema yang memanfaatkan bilangan prima. Bahkan di alam sekalipun, beberapa jenis serangga diamati hanya keluar dari sarangnya setelah 7, 13, atau 17 tahun, sebuah bilangan prima! [1]

Dari penjelasan singkat di atas jelas bahwa bilangan prima memiliki peranan yang cukup penting di berbagai bidang. Maka tidak aneh jika banyak ilmuan yang mendedikasikan waktunya untuk mempelajari bilangan prima ini.

II. TEORI DASAR

A. Definisi Membagi

Salah satu cabang yang cukup penting di Matematika Diskrit adalah Teori Bilangan. Bidang ini mencakup bahasan mengeni sifat-sifat bilangan bulat beserta teorema-teorema yang mendasarinya. Salah satu teorema dasar dalam teori bilangan adalah teorema keterbagian (Teorema Euclidean).

Teorema Keterbagian berkata : "Diberikan bilangan bulat a dan b, b>0, maka terdapat bilangan bulat q dan r yang unik yang memenuhi

$$a = qb + r \qquad 0 \leq r < b$$

dengan q dan r disebut *quotient* dan *remainder* dari pembagian a oleh b" [2].

Secara singkat, teorema di atas mengatakan ketika a dibagi oleh b, maka pasti terdapat bilangan q dan r yang memenuhi a = qb + r . Selanjutnya terdapat hal khusus jika r = 0. Hal tersebut dijelaskan oleh definisi di bawah ini.

Definisi membagi : "Diberikan bilangan bulat a dan b, a≠0. b disebut membagi a, dilambangkan dengan b | a, jika terdapat bilangan bulat q dan r yang memenuhi

$$a = qb + r \qquad r = 0$$

Selanjutnya jika tidak terdapat bilangan bulat q dan r yang memenuhi persamaan di atas, maka disebut b tidak membagi a, dilambangkan dengan b ∤ a."

Definisi di atas cukup menjelaskan sifat keterbagian dari suatu bilangan. Secara singkat, jika terdapat bilangan bulat a dan b, maka b membagi a jika *remainder* yang dihasilkan adalah 0.

Selanjutnya terdapat istilah *divisor* (pembagi) dan faktor. Dua istilah ini memiliki esensi yang sama, dimana bilangan bulat b disebut *divisor* atau faktor dari biangan

bulat a jika b | a .

Lalu, apa hubungannya semua teorema dan definisi di atas dengan bilangan prima ?

B. Definisi Bilangan Prima

Penjelasan dari teorema-teorema dan definisi di atas sudah cukup untuk kita mendefinisikan bilangan prima. Berikut adalah definisi dari bilangan prima :

" Sebuah bilangan bulat p > 1 disebut *bilangan prima*, jika bilangan tersebut hanya memiliki pembagi positif 1 dan p. Bilangan bulat yang lebih dari 1 yang bukan bilangan prima disebut bilangan *komposit* " [3].

Definisi di atas cukup jelas menerangkan sifat unik bilangan prima yang tidak dimiliki oleh bilangan lain (bilangan komposit). Dari definisi tersebut dapat ditarik kesimpulan bahwa sebuah bilangan bulat p disebut prima jika dari semua bilangan 2, 3, 4, …, p-1 tidak ada satupun yang membagi p. Hal ini karena bilangan yang mungkin membagi p hanya bilangan yang lebih kecil atau sama dengan p. Kesimpulan di atas merupakan salah satu cara untuk menguji keprimaan suatu bilangan, yang nanti akan dibahas di subbab lain di makalah ini.

Lalu apakah istimewanya bilangan prima tersebut? Memang ia hanya memilki dua faktor, tapi pasti ada alasan lain mengapa bilangan prima dipelajari secara intensif oleh berbagai ilmuan. Salah satu alasannya ditulis oleh Euclid di bukunya *Elements* mengenai sifat bilangan bulat. Euclid menulis apa yang kemudian dikenal sebagai Torema Dasar Aritmatika (*Fundamental Theorem of Arithmetics*) :

" Semua bilangan bulat n > 1 dapat ditulis sebagai hasil kali dari bilangan-bilangan prima. Penulisan ini unik, dengan mengabaikan urutan penulisan bilangan-bilangan prima tersebut"[4].

Apa akibat dari teorema di atas? Perhatikan bahwa semua bilangan bulat yang lebih dari satu merupakan bilangan prima, atau menurut Teorema Dasar Aritmatika, dapat ditulis dalam perkalian bilangan-bilangan prima. Artinya biangan prima dapat dikatakan sebagai "unsur pembangun" dari semua bilangan bulat.

Dari penjelasan singkat di atas cukup jelas peran besar bilangan prima pada bilangan lain (bilangan komposit). Contohnya untuk mengetahui semua faktor (pembagi) sebuah bilangan komposit, kita cukup menuliskan bilangan tersebut sebagai hasil kali dari bilangan-bilangan prima, lalu menentukan faktornya melalui kombinasi dari bilangan-bilangan prima tersebut.

C.Beberapa Sifat Bilangan Prima

Setelah membahas definsi dan teorema-teorema dasar yang berkaitan dengan bilangan prima, sekarang saatnya kita membahas beberapa sifat-sifat unik dari bilangan prima. Di antaranya adalah :

1) Terdapat tak-hingga banyak bilangan prima

   Bilangan prima sebagai bilangan yang unik selalu menggoda kengintahuan para matematikawan. Di antara hal yang ingin diketahui oleh para matematikawan adalah apakah terdapat tak-hingga bilangan prima atau tidak. Hal ini kembali dibuktikan oleh Euclid di bukunya yang sama *Elements*. Di buku tersebut Euclid memaparkan teoremanya yang singkat, padat, dan jelas. Teorema tersebut berbunyi :

   "Terdapat tak hingga banyaknya bilangan prima".

   Pembuktian teorema tersebut adalah sebagai berikut :

   Misal terdapat hingga banyaknya bilangan prima, sebut saja sebanyak n. Maka misalkan $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 7$, …., sampai p_n blangan prima terakhir. Sekarang perhatikan bilangan berikut :

   $$P = p_1p_2p_3\ldots p_n + 1$$

   Berdasarkan Teorema Dasar Aritmatika, P pasti memilliki minimal satu faktor prima, sebut saja p. Namun bilangan prima hanya terdiri dari $p_1,p_2,\ldots,p_n$, sehingga p haruslah sama dengan salah satu dari $p_1,p_2,\ldots,p_n$. Artinya selain p | P juga p | $p_1p_2p_3\ldots p_n$. Akibatnya p | (P - $p_1p_2p_3\ldots p_n$), atau p | 1. Tetapi tidak mungkin bilangan prima membagi 1. Kontradiksi! Artnya terdapat tak hingga banyaknya bilangan prima.[5]

   Pembuktian Euclid di atas merupakan contoh sebuah pembuktian elegan dengan menggunakan kontradiksi. Hasilnya terbukti terdapat tak hingga banyaknyabilangan prima.

2) Bilangan prima ke-n lebih kecil atau sama dengan $2^{2^{n-1}}$

   Dalam bahasa matematika : Jika p_n adalah bilangan prima ke n, maka $p_n \leq 2^{2^{n-1}}$.

   Pembuktiannya adalah sebagai berikut :

   Akan digunakan induksi matematika. Untuk n=1,

   $$p_1 \leq 2^{2^{1-1}} = 2 \text{ , benar.}$$

   Misal benar untuk semua bilangan prima kurang dari atau sama dengan n. Maka dengan memanfaatkan langkah pembuktian di bagian sebelumnya,

   $$p_{n+1} \leq p_1p_2p_3\ldots p_n + 1$$

   $$p_{n+1} \leq 2\cdot 2^2\cdot\ldots\cdot 2^{2^{n-1}} + 1 = 2^{1+2+2^2+\cdots+2^{n-1}} + 1$$

   Sementara, $1 + 2 + 2^2 + \cdots + 2^{n-1} = 2^n - 1$, sehingga

   $$p_{n+1} \leq 2^{2^n-1} + 1$$

   Padahal $1 \leq 2^{2n-1}$ untuk semua n, seingga

   $$p_{n+1} \leq 2^{2^n-1} + 2^{2^n-1}$$

   $$p_{n+1} \leq 2^{2^n}$$

   Induksi selesai. Terbukti $p_n \leq 2^{2^{n-1}}$.[6]

III. UJI KEPRIMAAN

Dalam dunia informatika, sering ditemukan kasus dimana sebuah bilangan harus ditentukan prima atau tidak. Kasus-kasus seperti ini bahkan sudah merupakan hal biasa bagi mereka yang sering mengikuti atau berpartisipasi dalam berbagai kontes pemrograman, seperti Olimpiade Komputer. Dalam kondisi kontes

seperti itu dibutuhkan efektifitas waktu dan komputasi, di mana peserta kontes diberikan waktu terbatas untuk menyelesaikan soal. Di kondisi ini dibutuhkan algoritma yang mangkus untuk pengujian keprimaan. Di bawah ini akan dibahas beberapa algoritma uji keprimaan, beserta pembuktiannya. Terdapat dua kasus uji keprimaan, yaitu pertama, diberikan input suatu bilangan bulat, harus ditentukan prima atau tidak. Kasus kedua adalah diberikan suatu himpunan bilangan bulat, harus ditentukan bilangan mana yang prima dan mana yang tidak. Kedua kasus tersebut akan di bahas pada subbab ini.

A. Uji Keprimaan Brute-Force

Seperti yang telah saya singgung di subbab sebelumnya, sebuah bilangan p dapat dikatakan prima jika tidak satupun di antara 2, 3, 4, …, p-1 yang membagi p. Fakta ini dapat kita pergunakan sebagai langkah pengujian kepriman suatu bilangan. Langkahnya cukup sederhana. Misal terdapat suatu bilangan n. Tentukan remainder dari pembagian n oleh 2, 3, 4, …, n-1. Jika ditemukan remainder sama dengan 0, maka langsung outputkan tidak prima. Jika tidak, lanjutkan hingga n-1. Jika tidak ditemukan satupun remainder yang 0, maka outputkan prima.

Berikut kode sumbernya dalam bahasa C++ :

```
bool IsPrime(int bil)
{
      if (bil==1)
            return false;
      for(int i = 2; i < bil; i++)
        if (!(bil % i))
            return false;
      return true;
}
```

Cara diatas memang sangat sederhana dan *straightforward*. Namun terdapat kelemahan di cara ini, yaitu tingkat kemangkusannya. Cara tersebut membutuhkan waktu komputasi yang sangat lama, terlebih jika input diperbesar. Karena tingkat kemangkusan yang rendah inilah maka dilakukan usaha untuk mencari cara yang lebih optimal dan mangkus. Beberapa dari cara ini dapat dilihat di bawah.

Cara pertama adalah dengan optimasi batas atas bilangan yang harus dicek remainder pembagiannya terhadap n. Pada cara di atas batas atas pengecekan adalah n-1. Batas ini dapat jauh lebih diperkecil menjadi $\sqrt{n}$. Mengapa $\sqrt{n}$? Berikut penjelasannya :

> Misal suatu bilangan bulat a adalah bilangan komposit. Maka dapat ditulis :
>
> a = bc, di mana (1 < b < a) dan (1 < c < a)
>
> Asumsi b ≤ c, maka $b^2 \leq bc = a$, sehingga $b \leq \sqrt{a}$. Karena b > 1, maka berdasarkan Teorema Dasar Aritmatika, terdapat bilangan prima p sedemikian sehingga p | b, di mana $p \leq b \leq \sqrt{a}$. Lalu karena p | b dan b | a, maka p | a. Dari sini dapat disimpulkan sebuah bilangan komposit a selalu memiliki faktor prima p yang memenuhi $p \leq \sqrt{a}$.[7]

Dari cara di atas dihasilkan algoritma yang cukup mangkus dalam menentukan sifat keprimaan sebuah bilangan bulat. Namun demikian, cara tersebut masih dapat dioptimasi lagi. Pada cara di atas optimasi dilakukan dengan membatasi batas atas bilangan yang harus dicek remaindernya pada pembagian terhadap n. Langkah optimasi selanjutnya adalah dengan memilih bilangan yang harus dicek dengan membuang bilangan-bilangan yang pasti komposit. Mengapa hal ini dilakukan? Berikut penjelasannya :

> Misal terdapat bilangan komposit a dengan salah satu faktor prima p, sehingga p | a. Misalkan juga terdapat bilangan bulat n di mana a | n. Karena p | a dan a | n, maka p | n. Hal ini juga berlaku untuk semua bilangan prima p lain yang merupakan faktor dari a. Dari sini dapat ditarik kesimpulan bahwa jika suatu bilangan bulat komposit membagi n, maka seluruh faktor primanya juga membagi n.

Dari penjelasan di atas dapat dilihat bahwa pengecekan hanya perlu dilakukan dengan bilangan prima saja. Lagi-lagi hal ini mengurangi banyak komputasi dengan sangat mangkus. Hal in karena pengecekan hanya dilakukan pada bilangan prima yang notabene persebarannya tidak terlalu banyak pada bilangan bulat. Bahkan semakin besar bilangan maka biangan prima akan semakin jarang ditemukan. Namun cara ini bukan tanpa cela. Memang komputasi yang harus dilakukan akan jauh diminimalisasi, tapi kita harus menentukan dahulu bilangan prima yang lebih kecil dari $\sqrt{n}$ yang harus dicek. Namun, begitu bilangan prima tersebut sudah diketahui, uji keprimaan terhadap n akan sangat-sangat mangkus.

Berikut salah satu contoh kode sumber yang menggunakan cara brute-force yang sudah dioptimasi. Di sini bilangan yang harus dicek hanya bilangan yang berpotensi prima dalam modulo 30. Semua bilangan yang pasti komposit dalam modulo 30 dibuang, tidak diikutkan dalam pengecekan.

```
bool IsPrime(int bil)
{
      if (bil < 2) return false;
      if (bil == 2) return true;
      if (bil == 3) return true;
      if (bil == 5) return true;
      if (!(bil % 2)) return false;
      if (!(bil % 3)) return false;
      if (!(bil % 5)) return false;
      if (bil < 7*7) return true;
      int cek[] = {7,11,13,17,19,23,29,31};
      int akar = (int)sqrt((double)bil);
      for(int r = 0; r < akar; r += 30)
            for(int i = 0; i < 8; ++i)
                if (!(bil % (r+cek[i])))
                    return false;
      return true;
}
```

B. Uji Keprimaan Fermat

Pierre De Fermat (1601-1665), seorang warga Negara Prancis, bukanlah seorang praktisi matematika, bukan

juga seorang yang mendapat pendidikan matematika yang intensif. Ia hanya seorang pegawai negeri biasa yang berprofesi sebagai pengacara. Kesenangannnya akan matematika hanyalah sebagai hobi pengisi waktu luang, hanya sampingan yang ia lakukan untuk mendapatkan kebahagiaan batin. Namun hobi yang disenanginya tersebut telah membawa kemajuan luar biasa pada matematika, terutama teori bilangan, di mana matematikawan murni pun tidak bisa melakukannya pada saat itu. Berbagi teorema dan penemuan dihasilkannya dari hobi sampingannya tersebut. Sebut saja kalkulus integral dan diferensial, torema peluang, geometri analitik, dan tentu saja sumbangan terbesarnya pada bidang teori bilangan, Fermat's (little) theorem.

Teorema fermat merupakan dasar dari berbagai algoritma uji keprimaan yang dikembangkan akhir-akhir ini. Teorema ini cukup sederhana tetapi sangat *powerful* untuk menentukan keprimaan dari suatu bilangan. Meskipun begitu, teorema fermat tidak 100% menentukan keprimaan sebuah bilangan. Hal ini akan dibahas jauh lebih dalam nanti.

Teorema Fermat mengatakan : misalkan p bilangan prima dan p ł a, maka

$$a^{p-1} \equiv 1 \ (mod\ p)$$ [8]

Sebelum membahas lebih lanjut mengenai teorema fermat, akan dijelaskan secara singkat tentang makna dari lambang kekongruenan (≡). Misal terdapat bilangan bulat a, b, dan c, maka a ≡ b (mod c) berarti c | (a-b). Maka teorema fermat di atas dapat ditulis p | (a^{p-1} - 1).

Selanjutnya berikut adalah pembuktian teorema fermat:

Perhatikan (p-1) buah kelipatan a berikut ini

a, 2a, 3a, ...,(p- l)a

Tidak ada d antara semua bilangan ini yang kongruen modulo p satu sama lain. Jika ada

ra ≡ sa (mod p) 1< r < s < p – 1

Maka a dapat dicoret menghasilkan r ≡ s (mod p) padahal 1< r < s < p – 1, suatu hal yang tidak mungkin. Oleh karena itu himpunan bilangan tadi akan kongruen dalam modulo p dengan himpunan bilangan 1, 2, 3,….,p-1 dengan urutan tertentu. Dengan mengalikan semua bilangan tersebut didapat :

$$a \cdot 2a \cdot 3a \cdots (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \ (mod\ p)$$

$$a^{p-1}(p-1)! \equiv (p-1)! \ (mod\ p)$$

Karena p ł (p-1)! Maka (p-1)! dapat dicoret menghasilkan

$$a^{p-1} \equiv 1 \ (mod\ p)$$

Terbukti! [9]

Teorema fermat di atas lama dianggap sebagai kemajuan matematika murni semata. Hal itu benar hingga perkembangan komputer mencapai titik di mana komputasi bilangan yang besar mulai dirasa *feasible*. Teorema fermat ini kemudian digunakan oleh para ilmuan komputer sebagai dasar pengujian keprimaan suatu bilangan. Bagaimana cara memanfaatkan teorema fermat ini di dalam pengujian keprimaan suatu bilangan?

Untuk menggunakan teorema fermat di atas, kita akan menggunakan kontrapositifnya. Kontrapositif teorema fermat kurang lebih akan seperti di bawah :

Sebuah bilangan ganjil merupakan bilangan komposit jika terdapat sebuah bilangan bulat positif a sedemikian sehingga

gcd(a,n) = 1 dan $a^{n-1} \not\equiv 1$ (mod n)

Lalu bagimana langkah-langkah pengujian keprimaan dengan uji keprimaan fermat di atas? Misalkan bilangan yang akan diuji keprimaannya adalah n. Pertama ambil sebarang bilangan bulat antara 1 sampai n-1, katakanlah biangan itu a. Kemudian hitung $a^{n-1}\ (mod\ n)$. Jika hasilnya bukan 1, maka n bukan bilangan prima. Jika haslnya 1 maka n kemungkinan besar prima. Langkah ini dapat diulangi dengan menggunakan a yang lain.

Cara di atas jika ditulis dalam pseudocode kira-kira akan seperti di bawah

```
/* n adalah biangan yang akan diuji
keprimaannya. iterasi adalah banyak
pengulangan yang ingin dilakukan */
bool Fermat(int n,int iterasi){
    if(n == 1){ // 1 bukan bilangan
prima
        return false;
    }
    for(int i=0;i<iterasi;i++){
        // pilih bilangan bulat
sebarang antara 1 dan n-1
        int a = rand()%(n-1)+1;
        // modulo adalah fungsi untuk
menghitung hasil modulo dari teorema
fermat.
        if(modulo(a,n) != 1){
            return false; /* n pasti
komposit */
        }
    }
    return true; /* n kemungkinan
besar prima */
}
```

Uji kepriman dengan teorema fermat di atas cukup mangkus menentukan keprimaan suatu bilangan, meskipun tidak 100% benar. Bahkan bilangan komposit yang lolos uji keprimaan fermat di atas sangat sedikit hingga bilangan-bilangan tersebut diberi istilah sendiri, yaitu *pseudoprime*. Suatu bilangan bulat komposit n dikatakan pseudoprime jika memenuhi teorema ferrmat di atas, yaitu $a^{n-1} \equiv 1$ (mod n)

Teorema fermat di atas cukup lama digunakan sebagai satu-sautnya cara untuk menentukan keprimaan suatu bilangan. Namun seiring berkembangnya ilmu pengetahuan semakin banyak cara / algoritma yang lebih efektif ditemukan. Meskipun demikian, kebanyakan algoritma baru tersebut juga menggunakan teorema fermat sebagai dasarnya. Hingga sampai saat ini pun teorema fermat masih luas digunakan sebagai algoritma uji keprimaan.

C. Sieve of Eratosthenes (Penyaringan Eratosthenes)

Sejauh ini algoritma uji keprimaan yang dibahas adalah algoritma yang menentukan keprimaan sebuah bilangan

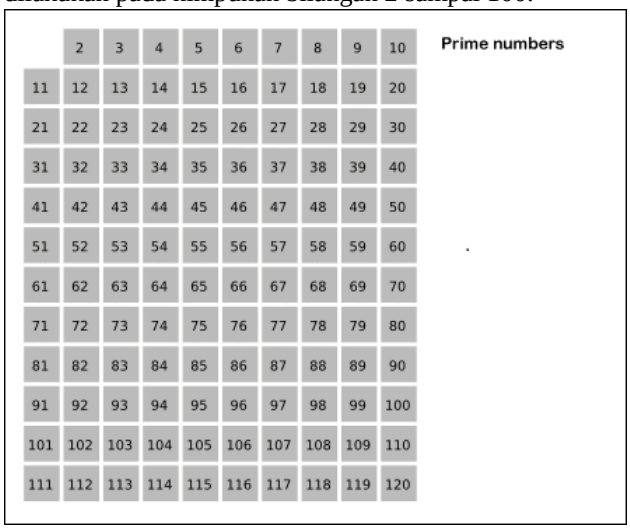
yang diberikan. Selanjutnya di sini akan dibahas algoritma yang menentukan bilangan mana saja yang prima di dalam sebuah himpunan bilangan bulat.

Kasusnya seperti ini, diberikan sebuah himpunan bilangan bulat, akan ditentukan bilangan mana saja yang prima dan mana yang komposit.

Salah satu cara yang umum digunakan, yang paling awal ditemukan, dan paling sederhana adalah Penyaringan Eratosthenes. Eratosthenes adalah kepala perpustakaan Alexandria. Ia hidup dari tahun 276 sampai 194 B.C. Di masa hidupnya ini ia menghasilkan beberapa penemuan yang menakjubkan, salah satu di antaranya adalah pengukuran keliling bumi. Salah satu sumbangsihnya dalam bidang teori bilangan adalah penyaringan Eratosthenes.

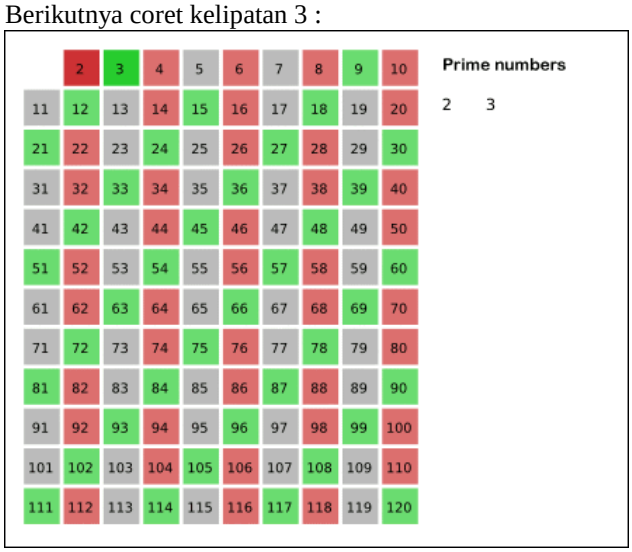
Cara kerja penyaringannya adalah seperti berikut. Pertama tuliskan bilangan dari 1 samapai n, bilangan terakhir yang masuk ke dalam himpunan yang akan disaring. Kemudian coret bilangan 1 dan semua bilangan kelipatan 2 kecuali 2 itu sendiri. Kemudian 3 adalah bilangan terkecil yang belum dicoret setelah 2, 3 ini pasti bilangan prima. Kemudan coret semua kelipatan 3 kecuali 3 itu sendiri. Kemudian 5 adalah bilangan terkecil yang belum dicoret setelah 3. 5 ini pasti bilangan prima. Kemudian coret semua kelipatan 5 kecuali bilangan 5 itu sendiri. Begitu seterusnya hingga bilangan prima yang dihasilkan merupakan bilangan prima terbesar yang masih kurang dari $\sqrt{n}$. Lalu semua bilangan yang belum dicoret patilah bilangan prima.

Untuk memperjelas langkah-langkah penyaringan di atas berikut ditampilkan contoh penyaringan yang dilakukan pada himpunan bilangan 2 sampai 100.

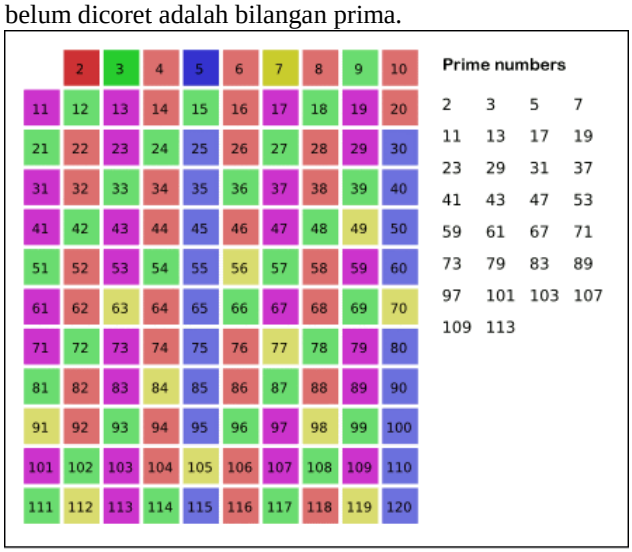


Pertama coret kelipatan 2 :

Berikutnya coret kelipatan 3 :



Lakukan terus menerus hingga semua kelipatan bilangan prima dicoret. Kali ini bilangan prima terbesar yang kurang dari $\sqrt{100}$ adalah 7. Maka semua bilangan yang belum dicoret adalah bilangan prima.



Cara ini cukup mangkus dan menghasilkan bilangan yang pasti 100% bilangan prima. Meskipun begitu untuk himpunan yang anggotanya sangat banyak, penyaringan

ini memakan waktu yang cukup lama. Oleh karena itu terus dikembangkan algoritma penyaringan lainnya yang lebih mangkus. Meskipun demikian, penyaringan Eratosthenes ini tetap menjadi dasar semua algoritma penyaringan, dengan mengganti syarat-syarat pencoretan dengan syarat yang lebih mangkus.

Dalam pseudocode, sieve of Eratosthenes kurang lebih akan seperti di bawah ini. Di sini kita menggunakan array of boolean, dengan indeksnya merupakan bilangan yang diwakilinya.

```
/* Kita akan menggunakan array of
boolean, dengan indeksnya merupakan
bilangan yang diwakilinya.
True berarti bilangan tersebut prima,
dan  false berarti komposit */
bool * SieveOfEratosthenes(int bil)
{
    bool * prima = new bool[bil+1];
    for(int i=0;i<=bil;i++)
        prima[i] = true;
    //0 dan 1 bukan bilangana prima,
maka harus dicoret
    prima[0] = false;
    prima[1] = false;
    int akar = sqrt(bil);
    for(int i=2;i<=akar;i++)
    {
     //Jika prima[i] true, maka kita
harus coret semua kelipatan i
        if(prima[i])
        {
            for(int j=2*i;j<=bil;j+=i)
                prima[j] = false;
        }
    }
    return prima;
}
```

IV. Aplikasi Bilangan Prima

Sejauh ini kita telah membahas definisi dan sifat-sifat bilangan prima, serta berbagai teorema ataupun algooritma pengujian keprimaan. Pada subbab ini akan dibahas pemanfaatan dari sifat-sifat tersebut dalam aplikasi bilangan prima baik dalam dunia informatika maupun di luar informatika.

A. Check-Digit ISBN

International Standard Book Number adalah kode yang dimiliki oleh setiap buku yang diterbitkan. Kode tersebut memiliki arti tertentu yang bukan merupakan bahasan makalah ini. Di sini akan dijelaskan mengapa dalam pengecekan ISBN digunkan modulo 11, sebuah bilangan prima.

Pengecekan ISBN dilakukan dengan cara mengalikan setiap angka dengan posisinya dihitung dari kanan, lalu mnentukan digit ke-10 sebagai check digit, sedemikian sehingga jumlah semua perkalian tadi kongruen 0 modulo 11.

Dalam penulisan kode angka, terdapat beberapa kemungkinan kesalahan yang sering dilakukan oleh manusia. Di antara kesalahan yang paling banyak terjadi adalah kesalahan satu digit dan pertukaran dua digit. Dua kesalahan yang paling umum dilakukan ini dapat terdeteksi 100% dengan check-digit ISBN. Hal ini karena sifat keprimaan bilangan 11 yang digunakan sebagai basis modulo.

Sebelum pendeteksian kesalahan di atas dibuktikan, akan terlebih dahulu dibuktikan sebuah lemma.

Lemma 1 : "Misalkan a dan b bilangan bulat, serta p bilangan prima. Jika $p \nmid a$ dan $p \nmid b$ maka $p \nmid ab$."
Perhatikan bahwa menurut Teorema Dasar Aritmatika, a dan b dapat ditulis dalam perkalian bilangan-bilangan prima. Karena $p \nmid a$ dan $p \nmid b$, maka tidak satupun dari bilangan prima tersebut yang sama dengan p. Jika a dan b dikalikan, maka bilangan-bilangan prima tersebut juga dikalikan, dengan tetap tidak mengandung p. Sehingga tidak satupun bilangan prima penyusun ab yang sama dengan p. Dari sini dapat disimpulkan $p \nmid ab$.

Sekarang misal terjadi kesalahan satu digit. Misal digit ke-i harusnya bernilai x, tetapi ditulis y, di mana $i,x,y<10$. Maka selisih dari pengecekan ISBN sebelum dan sesudah terjadi kesalahan adalah :

$$(11-i)x - (11-i)y = (11-i)(x-y)$$

Perhatikan bahwa jika ekspresi di atas habis dibagi 11, maka hasil pengecekan ISBN tidak akan berubah. Tetapi karena $11 \nmid (11-i)$ dan $11 \nmid (x-y)$, maka menurut lemma 1, $11 \nmid (11-i)(x-y)$. Hal ini mengkibatkan hasil pengecekan ISBN berubah, tidak lagi menghasilkan 0 modulo 11. Dari sini dapat diketahui bahwa telah terjadi kesalahan.

Kemudian misal terjadi pertukaran dua digit. Misal digit a_i dan a_j tertukar, dengan i dan j posisi digit dari kiri. Maka selisih dari pengecekan ISBN sebelum dan sesudah terjadi kesalahan adalah :

$$\begin{aligned}(11-i)a_i + (11-j)a_j - (11-i)a_j - (11-j)a_i \\ = (11-i)(a_i-a_j) + (11-j)(a_j-a_i) \\ = (11-i-11+j)(a_i-a_j) \\ = (j-i)(a_i-a_j)\end{aligned}$$

Perhatikan bahwa jika ekspresi di atas habis dibagi 11, maka hasil pengecekan ISBN tidak akan berubah. Tetapi karena $11 \nmid (j-i)$ dan $11 \nmid (a_i-a_j)$, maka menurut lemma 1, $11 \nmid (j-i)(a_i-a_j)$. Hal ini mengkibatkan hasil pengecekan ISBN berubah, tidak lagi menghasilkan 0 modulo 11. Lagi-lagi kesalahan ini dapat terdeteksi dengan baik. Hal ini tidak lepas dari keunikan sifat bilangan prima yang dimiliki oleh bilangan 11 yang digunakan sebagai basis modulo.

B. Public-Key Cryptography (Algoritma RSA)

Algoritma RSA ditemukan oleh Ron Rivest, Adi Shamir dan Leonard Adleman pada tahun 1977. Algoritma ini digunakan untuk enkripsi-dekripsi data yang ingin dirahasakan. Algoritma RSA termasuk ke dalam *Public-Key Cryptography* karena terdapat dua kunci yang dibutuhkan untuk mengenkripsi dan dekripsi data, yaitu kunci publik dan kunci privat. Kunci public disebarluaskan, digunakan untuk mengenkripsi data. Kunci privat digunakan untuk mendekripsi data yang telah terenkripsi. Algoritma ini sangat bergantung pada

bilangan prima dimana bilangan RSA, biasa dilambangkan dengan n haruslah merupakan bilangan hasil kali dua buah prima yang sangat besar. Pada praktiknya n memiliki lebih dari 200 digit.

Berikut langkah-langkah menentukan kunci public dan kunci privat pada algoritma RSA [10] :

1) Ambil dua bilangan prima yang sangat besar, p dan q
2) Hitung bilangan RSA, n = pq.
3) Hitung totient n, φ(n) = (p – 1)(q – 1)
4) Pilih suatu bilangan e di mana 1 < e < φ(n) dan e relatif prima terhadap φ(n). Bilangan ini akan menjadi kunci public bersama n.
5) Tentukan kunci privat d dari kekongruenan ed ≡ 1 (mod m)

Kemudian misal data yang akan dienkripsi adalah bilangan bulat p. Maka rumus untuk enkripsi dan dekripsi adalah :

- Enkripsi

$$c = p^e \ (mod\ n)$$

- Dekripsi

$$p = c^d \ (mod\ n)$$

Algoritma uji keprimaan sangat berperan besar dalam algoritma RSA ini. Uji keprimaan sangat diperlukan untuk menentukan p dan q yang sangat besar, hingga ratusan digit. Semakin mangkus algoritma uji keprimaaan, maka semakin besar p dan q yang bisa dihasilkan, semakin sulit usaha yang harus dilakukan untuk memfaktorkan n. Hal ini bermuara pada semakin amannya data yang dienkripsi menggunakan algoritma RSA ini.

C. Bilangan Prima di Alam

Bilangan prima dengan segala keunikannya telah banyak dimanfaatkan oleh para ilmuan untuk menghasilkan penemuan yang bermanfaat bagi kehidupan manusia. Tapi keajaiban bilangan prima ternyata tidak cukup sampai di situ. Sejenis serangga yang hidup di amerika bagian utara juga tampak memanfatkan sifat unik dari bilangan prima sebagai strategi untuk bertahan hidup.[11] Serangga tersebut adalah salah satu spesies jangkrik dari genus *magicicada*.

Jangkrik ini lebih banyak menghabiskan waktunya sebagai larva di dalam tanah. Ia hanya keluar dari sarangnya setiap 13 atau 17 tahun untuk berkembang biak kemudian mati. Angka 13 dan 17 ini tampak acak, tetapi sebenarnya tidak. Angka ini adalah bilangan prima yang memunyai manfaat tersendiri bagi jangkrik tersebut. Bilangan prima memiliki sifat tidak dapat dibagi oleh bilangan lain selain 1 dan dirinya sendiri. Sifat inilah yang dimanfaatkan oleh jangkrik untuk menghindari pemangsa yang memiliki siklus hidup yang tetap. Misal salah satu jenis pemangsa muncul setiap 6 tahun. Binatang yang memiliki siklus hidup 9 tahun akan bertemu dengan pemangsa ini setelah 18 tahun. Namun jangkrik yang memiliki siklus hidup 13 tahun tadi tidak akan bertemu dengan pemangsa tersebut.

Secara matematis dapat dijelaskan sebagai berikut. Misal p bilangan prima, siklus hidup jangkrik. Misalkan pula a biangan bulat , a ≠ p, adalah siklus hidup pemangsa. Karena p bilangan prima dan p ≠ a, maka a ∤ p. Misal k suatu bilangan bulat, maka a ∤ kp, kecuali untuk k yang meruapakan kelipatan a. Dari sini dapat disimpulkan siklus hidup jangkrik akan sangat jarang bersamaan dengan siklus hidup pemangsa.

Dengan bilangan prima ini, jangkrik akan memiliki peluang bertahan hidup yang lebih besar.

V. KESIMPULAN

- Bilangan prima sebagai bilangan yang memiliki sifat keterbagian yang unik memiliki banyak manfaat di berbagai bidang.
- Algoritma uji keprimaan yang mangkus sangat dibutuhkan untuk berbagai macam aplikasi.
- Berbagai teori / algoritma berkaitan dngan bilangan prima masih terus dikembangkan.
- Kadangkala bilangan-bilangan acak yang ditemukan di alam merupakan bilangan prima, dengan arti tertentu yang memanfaatkan keunikan sifat bilangan prima.

REFERENSI

[1] "Nature's hidden prime number code" http://www.bbc.co.uk/news/magazine-14305667 Diakses 3.53 PM 18/12/2012

[2] Burton,David M., *Elementary Number Theory Sixth Edition*. New York: McGraw-Hill, 2007, hal 17.

[3] Burton,David M., *Elementary Number Theory Sixth Edition.* New York: McGraw-Hill, 2007, hal 39.

[4] Burton,David M., *Elementary Number Theory Sixth Edition.* New York: McGraw-Hill, 2007, hal 41.

[5] Burton,David M., *Elementary Number Theory Sixth Edition.* New York: McGraw-Hill, 2007, hal 46.

[6] Burton,David M., *Elementary Number Theory Sixth Edition.* New York: McGraw-Hill, 2007, hal 48.

[7] Burton,David M., *Elementary Number Theory Sixth Edition.* New York: McGraw-Hill, 2007, hal 44.

[8] Burton,David M., *Elementary Number Theory Sixth Edition.* New York: McGraw-Hill, 2007, hal 88.

[9] Burton,David M., *Elementary Number Theory Sixth Edition.* New York: McGraw-Hill, 2007, hal 88.

[10] Munir, Rinaldi. *Matematika Diskrit*. Bandung: Informatika.2004.

[11] "Nature's hidden prime number code" http://www.bbc.co.uk/news/magazine-14305667 Diakses 3.53 PM 18/12/2012

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 Desember 2012

ttd

Muhammad Furqan Habibi (13511002)