Computatio: Journal of Computer Science and Information Systems, 9/1 (2025), 19 – 29

PEMERIKSA PRIMALITAS SUATU BILANGAN MENGGUNAKAN PEMROGRAMAN DINAMIS

Teady Matius Surya Mulyana[1], Agustinus Frits Wijaya[2], Jusia Amanda Ginting[3], Francka Sakti Lee[4], Giovanni Alvaro[5]

[1,2,3,5] Program Studi Informatika, Fakultas Teknologi Dan Desain, Universitas Bunda Mulia, Jln. Lodan Raya nomor 2, Jakarta Utara, 11440, Indonesia

[4] Program Studi Sistem Informasi, Fakultas Teknologi Dan Desain, Universitas Bunda Mulia, Jln. Lodan Raya nomor 2, Jakarta Utara, 11440, Indonesia

E-mail: [1]tmulyana@bundamulia.ac.id, [2]agustinus.wijaya@bundamulia.ac.id, [3]jginting@bundamulia.ac.id, [4]flee@bundamulia.ac.id ,[5]s32210056 @student.ubm.ac.id

Abstrak

Bilangan prima merupakan bilangan yang sangat penting dalam berbagai bidang komputasi. Sebagai bilangan yang tidak habis terbagi bilangan manapun selain 1 dan bilangan itu sendiri, sehingga sering dimanfaatkan pada pengkodean yang memerlukan bilangan unik. Salah satu masalah yang timbul sehubungna dengan keperluan bilangan prima ini adalah untuk mendapatkan suatu bilangan unik yang prima memerlukan pengujian primalitas bilangan. Pemeriksaan primalitas dapat mengintegrasikan prinsip Dynamic Programming (DP) melalui teknik memoization dan penyimpanan data persisten. Proses ini akan memanfaatkan tabel bilangan prima yang tersimpan sehingga dapat digunakan Kembali pada pengujian bilangan primalitas bilangan berikutnya. Pendekatan trial division, memungkinkan sistem untuk secara menyimpan dan menggunakan kembali hasil komputasi sebelumnya, sehingga mempercepat proses pemeriksaan bilangan prima di masa mendatang. Uji coba pembagian dioptimalkan dengan batas akar kuadrat. Melalui serangkaian pengujian kinerja, hasil menunjukkan peningkatan efisiensi yang signifikan. Waktu eksekusi berkurang drastis ketika bilangan yang diperiksa sudah ada dalam tabel memoized atau berada dalam jangkauan bilangan prima yang telah dibangun sebelumnya. Namun, penelitian ini juga mengidentifikasi keterbatasan kinerja yang substansial. Ketika program dihadapkan pada kebutuhan untuk memperluas tabel ke rentang bilangan yang sangat besar, algoritma uji coba pembagian sekuensial yang digunakan untuk perluasan tabel menjadi tidak optimal dan memakan waktu komputasi yang sangat lama.

Kata kunci*—Pemeriksaan Primalitas, Dynamic Programming, Memoization, trial division, Bilangan Prima*

Abstract

Prime numbers are crucial in various computing fields. As numbers that are not divisible by any number other than 1 and themselves, they are often used in coding that requires unique numbers. One of the challenges associated with this need is that obtaining a unique prime number requires a primality test. Primality testing can integrate Dynamic Programming (DP) principles through memoization and persistent data storage techniques. This process utilizes a stored prime number table for reuse in subsequent primality tests. The trial division approach allows the system to automatically store and reuse previous computational results, thereby speeding up the process of checking for prime numbers in the future. Trial division is optimized with square root limits. Through a series of performance tests, the results show significant efficiency improvements. Execution time is drastically reduced when the number being checked is already in the memoized table or is within the range of a previously constructed prime number. However, this study also identified substantial performance limitations. When the program is faced with the need to expand

the table to a very large range of numbers, the sequential trial division algorithm used for table expansion becomes suboptimal and takes a very long computational time.

Keywords— *Primality Checking, Dynamic Programming, Memoization, Trial Division, Prime Numbers*

1. PENDAHULUAN

Pemeriksaan primalitas suatu bilangan adalah salah satu masalah yang sering ditemui pada berbagai penerapan teori bilangan dalam ilmu komputer[1]. Sebagai bilangan yang unik, maka bilangan prima sering banyak digunakan pada proses komputasi, salah satu yang sering ditemui adalah pada kriptografi[2], teori kode[3], *steganography*[4] dan komputasi ilmiah. Dalam kriptografi, misalnya, bilangan prima adalah fondasi dari algoritma kunci publik seperti RSA (Rivest–Shamir–Adleman)[5]. Keamanan RSA sangat bergantung pada kesulitan faktorisasi bilangan komposit besar menjadi faktor-faktor primanya[6]. Oleh karena itu, kemampuan untuk secara efisien menentukan apakah sebuah bilangan itu prima (atau untuk menemukan bilangan prima besar) sangat penting untuk generasi kunci yang aman[7]. Di bidang keamanan jaringan, bilangan prima juga digunakan dalam protokol pertukaran kunci Diffie-Hellman untuk memastikan komunikasi yang aman[8][9]. Keamanan system merupakan salah satu kebutuhan sistem yang sangat diperlukan[10][11][12] . Selain itu, dalam teori kode, bilangan prima berperan dalam desain kode koreksi kesalahan yang efisien, .deteksi kesalahan pengkodean sangat diperlukan pada transmisi data[13][14]. Sementara dalam komputasi ilmiah dan simulasi, bilangan prima dapat digunakan dalam pembangkitan bilangan acak berkualitas tinggi dan optimasi kinerja algoritma tertentu. Dengan demikian, pengembangan metode pengujian primalitas yang efisien dan andal terus menjadi area penelitian yang relevan dan krusial.

Sebuah bilangan prima adalah bilangan asli yang lebih besar dari 1 dan hanya memiliki dua pembagi positif: 1 dan bilangan itu sendiri.[15]. Pertumbuhan kebutuhan akan komputasi yang efisien, menjadikan pengembangan algoritma pemeriksaan primalitas yang optimal menjadi sangat relevan[16][17]. Secara matematis, pemeriksaan bilangan prima N dapat didefinisikan sebagai fungsi is_prime(n,*divisor*) seperti pada persamaan (1).

$$is\_prime(n, divisor) = \begin{cases} FALSE & \rightarrow Jika\ n < 2 \\ TRUE & \rightarrow Jika\ divisor \times divisor > n \\ FALSE & \rightarrow\ jika\ n\ mod\ divisor = 0 \\ is\_prime(n, divisor + 1) & \rightarrow selain\ ketiga\ kondisi\ di\ atas \end{cases} \quad (1)$$

Dimana suatu bilangan bulat bukanlah bilangan prima jika n<2 dan jika n habis dibagi suatu bilangan[18]. Jika pembagi dikalikan pembagi jumlahnya lebih dari n, selama n belum ditemukan sebagai bukan bilangan prima, maka n ada bilangan prima, proses akan melakukan rekursif dengan menambahkan pembagi sampai dicapai nilai pembagi yang dikuardaratkan memiliki nilai yang lebih besar dari n[19][20].

Pemrograman dinamis (atau *Dynamic Programming*) adalah sebuah teknik optimasi dalam ilmu komputer yang digunakan untuk menyelesaikan masalah kompleks[21]. *Dynamic programming* memungkin suatu proses melakukan perubahan proses rekursi berdasarkan kondisi rekursi, dimana *dynamic programming* melakukan pemecahan masalah menjadi sub masalah yang kecil,menghitung ulang soluis untuk tiap masalah yang sama dan meyimpannya dalam memori atau yang disebut *memoization*[22]. Memoization merupakan tabel penyimpanan Solusi yang dapat dipilih secara langsung ketika suatu proses pada dynamic programming mengalami

kasus yang sama[23]. Kemudian dengan menggunakan solusi dari sub masalah, maka dibangun solusi untuk masalah yang lebih besar secara efisien.[24]

Penelitian ini bertujuan untuk mengembangkan dan menganalisis sebuah program pemeriksaan primalitas suatu bilangan yang menerapkan prinsip *Dynamic Programming* (DP) dengan memanfaatkan tabel bilangan prima yang di-*memoized*[25]. Tabel ini disimpan secara persisten dalam file biner, memungkinkan program untuk mengingat bilangan prima yang telah ditemukan sebelumnya dan menggunakannya untuk mempercepat proses pemeriksaan primalitas suatu bilangan. Pendekatan pemeriksaan yang digunakan pada penelitian ini adalah Uji Coba Pembagian atau *Trial Division*[15].

1.1. Penelitian Terdahulu

Uji Coba Pembagian (*Trial Division*) adalah metode paling sederhana dan intuitif untuk menentukan apakah sebuah bilangan bulat N adalah prima[26]. Prinsip dasarnya adalah memeriksa apakah N habis dibagi oleh bilangan bulat apa pun dari 2 hingga N[17]. Jika tidak ada pembagi yang ditemukan dalam rentang ini, maka N adalah prima. Sekalipun *Trial Division* mudah dipahami dan diimplementasikan, kompleksitas waktu uji coba pembagian adalah O(N), menjadikannya tidak praktis untuk memeriksa primalitas bilangan yang sangat besar[19]. Trial Division merupakan pengujian primalitas yang pandir, karena akan menguji semua bilangan sebagai pembagi N mulai dari 2 sampai $\sqrt{N}$ [19]. meskipun pada prakteknya trial division dapat dipersingkat dengan membaginya hanya dengan bilangan ganjil, sehingga kompleksitas proses dapat dipersingkat separuhnya[19].

Square Root Rule menyatakan bahwa sebuah bilangan bulat N pasti prima jika tidak habis dibagi bilangan prima lain mulai dari 3 sampai $\sqrt{N}$. Tetapi tentu saja dengan jika bilangan tersebut adalah bilangan ganjil. Tentu saja dengan *Trial Division* murni hal ini tidak dapat dilakukan karena tidak adanya informasi bilangan prima di bawah $\sqrt{N}$.[15].

Fokus dari Penelitian ini adalah pendekatan yang menggabungkan *trial division* dengan *square root rule* yang merupakan uji coba sederhana untuk primalitas bilangan yang diterapkan pada algoritma *Dynamic Programming* dengan *memoizing table* nya yang menyimpan deret bilangan prima yang digunakan sebagai pembagi untuk menguji primalitas dari input bilangan N. Dimana skenario Pembangunan tabel bilangan prima untuk digunakan kembali dilakukan secara bertahap, berbeda dengan algoritma yang dirancang untuk satu kali pengujian bilangan sangat besar atau pembuatan saringan masif. Dengan digunakannya *Dynamic programming*, maka bilangan prima yang pernah digunakna untuk menguji primalitas bilangan sebelunya dapat digunakan untuk menguji primalitas bilangan lainnya[27]. Dengan demikian maka *square root rule* dapat diterapkan pada *trial division*. Sebagaimana yang sedang digemari belakangan ini, pengujian dan analisis pada model matematis yang merupakan acuan penelitian diterapkan menggunakan pemrograman Python[28].

2. METODE PENELITIAN

Metode pemeriksaan primalitas bilangan bulat N disusun dengan menggabungkan *trial division* dan *square root rule*. Suatu bilangan bulat N akan diperiksa primalitas nya dengan menggunakan sekumpulan bilangan prima 2 sampai $\sqrt{N}$. Jika pada pemeriksaan trsebut ada satu saja bilangan prima tersimpan yang berhasil membagi bilangan N tersebut sampai habis, maka bilangan bulat N tersebut bukanlah bilangan prima.

2.1. Skenario *dynamic programming*, *trial division* dan *square root rule*

Untuk mewujudkan *square root rule* dan *trial division* pada program ini sehingga dapat saling melengkapi, maka skenario yang disusun adalah sbb:

1. Jika *memorizing table* masih kosong, maka *trial division* secara *naïve* atau pandir akan dilakukan dengan asumsi 2 digunakan untuk membagi pertama kali agar memastikan bilangan N tersebut adalah bilangan ganjil. Selanjutnya bilangan dibagi 3, 5, 7 dst yang merupakan bilangan ganjil sampai $\sqrt{N}$.
2. Jika *memorizing table* berisi daftar bilangan prima, maka *square root rule* dilakukan dengan membagi bilangan N tersebut dengan bilangan-bilangan prima tersimpan mulai dari 2 sampai $\sqrt{N}$.
3. Jika $\sqrt{N}$ melebihi bilangan tersimpan, maka mekanisme *trial division* berdasarkan bilangan prima tersimpan terakhir dengan menambahkan 2, karena bilangan prima selain 2 pasti bilangan ganjil, maka bilangan yang diberikan *trial division* ini sudah pasti bilangan ganjil,
4. Pada saat melakukan *trial division*, maka proses rekursi *dynamic programming* dari fungsi is_prime(n, *divisor*) akan dilakukan untuk menguji primalitas bilangan yang dihasilkan oleh trial division. Jika bilangan tersebut adalah bilangan prima maka bilangan tersebut akan disimpan pada *memozing table*, dan proses *trial division* dilanjutkan sampai $\sqrt{N}$.

2.2. Algoritma Pemeriksaan Primalitas *(is_prime_dp)*

Program pemeriksaan primalitas dikembangkan menggunakan bahasa pemrograman Python. Metode yang diterapkan adalah uji coba pembagian (*trial division*) yang dioptimalkan dengan *memoization* dan penyimpanan persisten berdasarkan *square root rule*.

Fungsi is_prime_dp (n, prime_table) adalah inti algoritma yang melakukan pemeriksaan primalitas. Fungsi ini menerima bilangan N dan tabel prima saat ini. Ia mengimplementasikan logika uji coba pembagian yang dioptimalkan dan secara kondisional memicu proses perluasan tabel dengan menemukan dan menambahkan bilangan prima baru, memanfaatkan tabel yang sudah ada untuk mempercepat pemeriksaan.

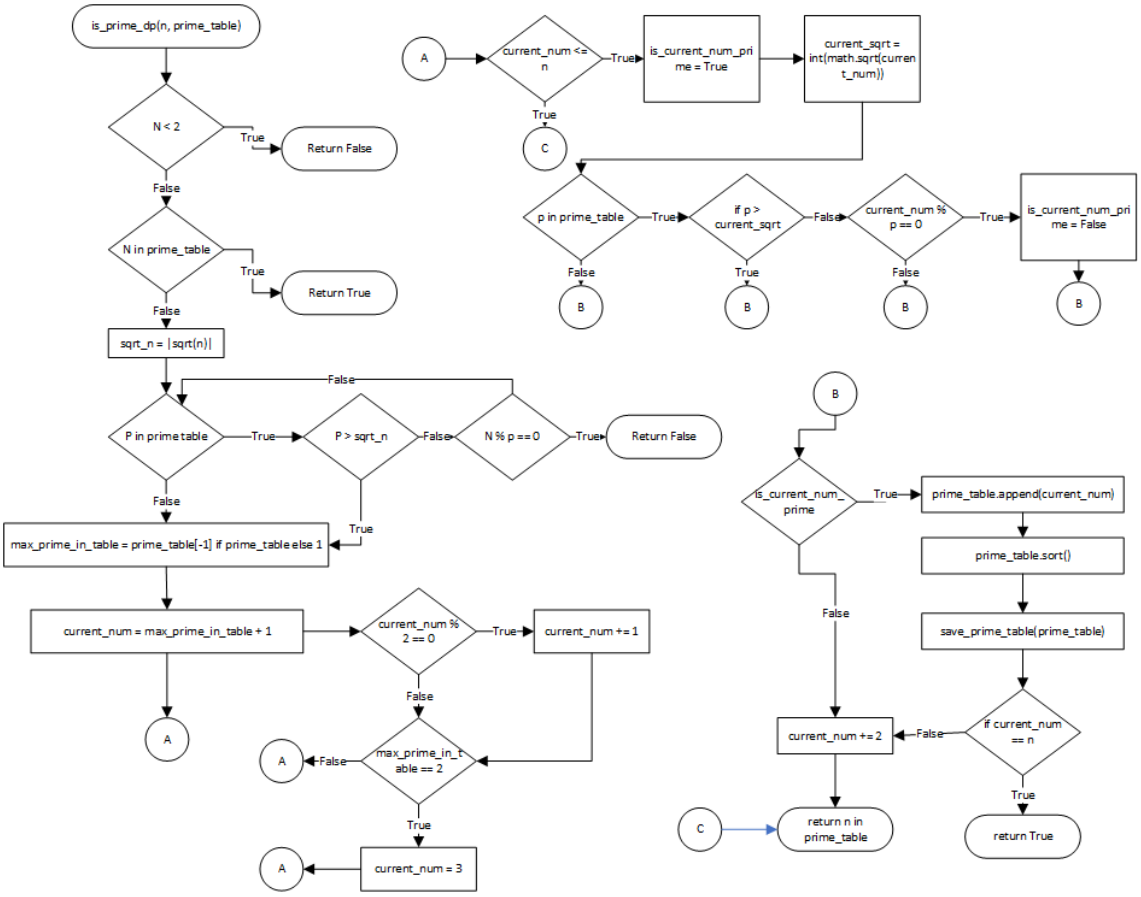
Proses pada fungsi is_prime dapat dilihat pada Gambar 1. Fungsi inti is_prime_dp(n, prime_table) mengimplementasikan logika pemeriksaan primalitas. Kasus Dasar adalah Bilangan N<2 langsung dikategorikan bukan prima. Jika N sudah ada dalam prime_table, ia langsung diidentifikasi sebagai prima.

Optimasi Akar Kuadrat dilakukan dengan memeriksa primalitas N, program hanya mencoba membagi N dengan bilangan prima yang sudah ada di prime_table hingga batas N. Jika N habis dibagi oleh salah satu bilangan prima dalam rentang ini, maka N bukan prima.

Perluasan Tabel merupakan proses yang mewujudkan *Dynamic Programming*: Jika N tidak habis dibagi oleh bilangan prima yang sudah ada di tabel hingga N (dan N mungkin lebih besar dari bilangan prima terbesar di tabel), program akan mulai mencari bilangan prima baru untuk memenuhi persyaratan dari *square root rule*. Pencarian dimulai dari bilangan ganjil berikutnya setelah bilangan prima terbesar yang diketahui (max_prime_in_table + 2).

Untuk setiap current_num dalam rentang ini (hingga N), program secara rekursif menerapkan logika pemeriksaan primalitas yang sama: membagi current_num dengan bilangan prima yang sudah ada di prime_table hingga current_num.

Jika current_num ditemukan sebagai bilangan prima, ia ditambahkan ke prime_table, tabel diurutkan, dan disimpan kembali ke file biner. Proses ini memastikan tabel terus bertambah dengan bilangan prima baru yang ditemukan.



Gambar 1 Proses Pengujian Primalitas

2.3 Analisis Kompleksitas Algoritma

Analisis kompleksitas algoritma `is_prime_dp` perlu mempertimbangkan dua skenario utama: ketika bilangan yang diuji telah ada di dalam tabel prima (*memoized*) dan ketika tabel prima perlu diperluas. Dan sebagai pembanding ditambahkan skenario *trial division* tanpa *dynamic programming*, dimana seknario ini hanya melakukan *trial division* saja, pengujian dilakukan mulai dari bilangan prima terkecil sampai bilangan prima terakhir sebelum $\sqrt{N}$.

2.3.1 Analisis Kompleksitas Waktu

1. Kasus N sudah ada pada `prime_table`:
   Jika bilangan N sudah ada dalam tabel `prime_table` yang dimuat ke memori, pemeriksaan primalitas menjadi sangat cepat. Proses ini hanya melibatkan pencarian dalam daftar yang sudah ada. Jika daftar diimplementasikan sebagai *hash set* atau diurutkan dan menggunakan pencarian biner, kompleksitas waktu adalah O (1) untuk hash set secara rata-rata atau O (log K) untuk pencarian biner pada daftar berukuran K, dimana K adalah jumlah bilangan prima yang tersimpan.
2. Kasus Tabel Perlu Diperluas:
   Sebagai skenario kasus terburuk untuk algoritma ini, terutama saat pertama kali menjalankan program atau ketika menguji bilangan N yang jauh lebih besar dari bilangan

prima terbesar yang sudah tersimpan. Proses perluasan tabel melibatkan *sequential trial division* untuk menemukan bilangan prima baru hingga $\sqrt{N}$.

2.3.2 Analisis Kompleksitas Memori

Kompleksitas memori terkait langsung dengan ukuran tabel `prime_table` yang disimpan.

1. Penyimpanan 'prime_table'

   Ukuran memori yang dibutuhkan untuk menyimpan tabel bilangan prima adalah O(K), di mana K adalah jumlah bilangan prima yang telah ditemukan dan disimpan. Menurut Teorema Bilangan Prima, jumlah bilangan prima hingga

   $$\frac{x}{\ln x}$$

   Jadi, untuk menyimpan semua bilangan prima hingga Nmax (bilangan terbesar yang dicakup tabel), kompleksitas memori adalah

   $$O\left(\frac{Nmax}{\ln Nmax}\right)$$

   Semakin besar Nmax, semakin besar pula kebutuhan memori untuk tabel tersebut. Penyimpanan persisten (file biner) akan mencerminkan ukuran ini.

2. Panggilan Rekursif (*Stack Space*)

   Fungsi `is_prime_dp` menggunakan rekursi. Kedalaman rekursi dalam skenario perluasan tabel dapat mencapai $\sqrt{N}$ dalam kasus terburuk (misalnya, jika setiap `current_num` adalah prima dan memicu panggilan rekursif untuk pengujiannya sendiri hingga $\sqrt{current\_num}$. Ini dapat menyebabkan *stack overflow* untuk bilangan sangat besar jika tidak diatur dengan baik, meskipun Python memiliki batas rekursi *default*. Pengujian current_num dengan prime_table yang sudah ada, kedalamannya lebih ke $O(\log\sqrt{current\_num})$ jika pencarian biner digunakan pada tabel, atau $O(\sqrt{current)num})$ jika dilakukan *trial division* dengan pembagi baru yang belum ada di tabel.

3. HASIL DAN PEMBAHASAN

Hasil dari proses ini ditujukan untuk melakukan efisiensi pemeriksaan primalitas bilangan kecil dan berulang. Salah satu keuntungan dari proses ini adalah jika bilangan N yang dicari merupakan bilangan yand dapat diuji dengan data bilangan prima sampai $\sqrt{N}$. Keuntungan lain yang didapat adalah jika bilangan yang diuji primalitasnya adalah bilangan yang tersimpan pada memorizing table atau sudah pernah digunakan untuk menguji primalitas bilangan, maka proses trial division tidak perlu dilakukan karena berarti bilangan tersebut adalah bilangan prima. Dengan demikian makan keuntungan yang didapatkan dari proses ini adalah Efisiensi untuk Bilangan Kecil dan Berulang, dimana untuk bilangan N yang relatif kecil atau yang berada dalam jangkauan tabel bilangan prima yang sudah ada. Program akan menunjukkan efisiensi yang baik. Pemeriksaan menjadi sangat cepat karena hanya perlu melakukan pencarian di tabel yang sudah ada atau sedikit perluasan.

Tabel bilangan prima berhasil bertumbuh dan disimpan ke file biner setiap kali bilangan prima baru ditemukan selama pemeriksaan. Ini adalah implementasi kunci dari pendekatan *Dynamic Programming*, di mana pengetahuan yang diperoleh dari satu komputasi digunakan untuk mempercepat komputasi di masa depan.

Dengan penggunaan pendekatan *trial division* maka ada keterbatasan kinerja untuk bilangan sangat besar: Meskipun konsep *Dynamic programming* diterapkan, program menunjukkan keterbatasan kinerja yang signifikan ketika dihadapkan pada bilangan N yang

sangat besar (misalnya, 108 atau 109), terutama jika tabel bilangan prima yang ada masih jauh di bawah N.

Fenomena ini terlihat jelas ketika menginput N=100.000.013 atau N=1.000.000.003 setelah tabel terakhir hanya mencapai 9001. Program akan mencoba mencari dan mengidentifikasi semua bilangan prima di antara 9001 dan N. Proses ini melibatkan iterasi melalui jutaan bahkan miliaran bilangan (current_num) dan untuk setiap current_num melakukan uji coba pembagian. Meskipun uji coba pembagian dioptimalkan hingga current_num, jumlah total operasi komputasi menjadi sangat besar, menyebabkan program berjalan sangat lambat (berjam-jam atau bahkan berhari-hari) atau terlihat "macet" tanpa memperbarui tabel. Hal ini disebabkan oleh sifat algoritma perluasan tabel yang masih berbasis uji coba pembagian sekuensial, yang tidak optimal untuk menghasilkan daftar bilangan prima dalam rentang yang sangat luas secara sekaligus.

3.1. Metodologi Pengujian Kinerja

Untuk menguji dan membandingkan kecepatan pencarian bilangan prima dengan dan tanpa tabel yang sudah terisi, pengujian dilakukan dalam dua fase. Pemilihan bilangan 1201, 7879, dan 9001 sebagai data uji didasarkan pada beberapa pertimbangan. Bilangan-bilangan ini dipilih karena merupakan bilangan prima dalam rentang yang relatif kecil hingga menengah, yang memungkinkan demonstrasi yang jelas mengenai proses perluasan tabel secara bertahap dan manfaat memoization persisten. Bilangan-bilangan ini cukup terpisah satu sama lain sehingga setiap input dalam fase pembangunan tabel akan memicu perluasan tabel yang signifikan, sekaligus menunjukkan kemampuan program untuk menyimpan dan memanfaatkan bilangan prima yang ditemukan. Selain itu, bilangan-bilangan ini representatif untuk menunjukkan skenario di mana bilangan uji sudah ada di tabel (Fase Pengujian Kedua), memvalidasi efisiensi akses langsung. Meskipun pengujian benchmarking standar seringkali melibatkan rentang bilangan yang jauh lebih besar dan lebih banyak titik data, pilihan ini cukup untuk memvalidasi hipotesis utama penelitian mengenai efisiensi memoization dalam konteks trial division bertahap.

Fase Pengujian Pertama (Membangun Tabel): File prime_table.bin dipastikan tidak ada atau dihapus sebelum pengujian dimulai. Program dijalankan, dan bilangan N diinputkan secara berurutan: 1201, 7879, dan 9001. Waktu eksekusi untuk setiap input dicatat. Pada fase ini, program akan secara aktif mencari dan menyimpan bilangan prima baru ke dalam tabel.

Fase Pengujian Kedua (Memanfaatkan Tabel): File prime_table.bin yang telah terisi dari fase pertama tidak dihapus. Program dijalankan kembali, dan bilangan N diinputkan secara berurutan: 9001, 7879, dan 1201. Waktu eksekusi untuk setiap input dicatat. Pada fase ini, diharapkan program akan memanfaatkan tabel yang sudah ada untuk mempercepat proses.
Pengukuran waktu dilakukan menggunakan modul time di Python, mencatat waktu sebelum dan sesudah panggilan fungsi is_prime_dp.

3.2. Hasil Pengujian

Tabel 1 merupakan table hasil pengujian. Waktu aktual dapat bervariasi tergantung pada spesifikasi perangkat keras dan beban sistem, namun pola efisiensi yang disajikan akan konsisten. Berdasarkan tabel hasil pengujian pada Tabel 1, terlihat pola efisiensi program dengan penerapan *Dynamic Programming* dan penyimpanan tabel biner: Kinerja pada Pengujian Pertama (Membangun Tabel). Pada fase ini, setiap input N memerlukan waktu yang signifikan karena program harus melakukan dua tugas utama: (a) memeriksa primalitas N itu sendiri, dan (b) jika N lebih besar dari bilangan prima terbesar pada table. Program harus mencari dan menemukan semua bilangan prima di antara bilangan prima terbesar terakhir dan N. Proses pencarian dan penambahan bilangan prima baru ke tabel, serta operasi tulis ke file biner, berkontribusi pada waktu eksekusi yang lebih lama.

Misalkan, saat menginput 1201, program harus mengisi tabel dengan bilangan prima dari 2 hingga 1201. Kemudian, saat menginput 7879, program melanjutkan perluasan tabel dari 1201 hingga 7879. Waktu yang dibutuhkan mencerminkan upaya komputasi untuk membangun basis pengetahuan prima.

Peningkatan Kinerja pada Pengujian Kedua merupakan proses Memanfaatkan Tabel yang berisi bilangan prima. Peningkatan kecepatan yang dramatis terlihat pada fase pengujian kedua. Waktu eksekusi untuk semua input (9001, 7879, dan 1201) turun menjadi nilai yang sangat kecil (mendekati nol). Hal ini terjadi karena pada fase ini, file prime_table.bin sudah berisi semua bilangan prima hingga setidaknya 9001 (berkat pengujian pertama). Ketika program dijalankan, ia memuat tabel yang sudah lengkap ini ke memori.

Ketika N yang diinputkan (misalnya 9001) sudah ada di dalam prime_table, fungsi is_prime_dp langsung mengembalikan True tanpa perlu melakukan komputasi uji coba Program "mengingat" hasil sebelumnya, sehingga tidak perlu menghitung ulang. Bahkan untuk N=7879 dan N=1201, yang lebih kecil dari 9001, mereka juga ditemukan dengan cepat karena sudah ada di dalam tabel yang telah dimuat.

Tabel 1. Pengujian Skenario Waktu Eksekusi

Skenario Pengujian	Urutan Input	Bilangan N	Waktu Eksekusi (detik)	Penggunaan memori (MB)	Iterasi Perluasan	Total Iterasi Penentuan Primalitas N	Kompleksitas Waktu (Teoritis)	Kompleksitas Memori (Teoritis)	Kondisi Tabel Setelah Input	Keterangan
Pembanding Algoritma (Trial Division tanpa DP)	1	1201	0.000272	101.54	600	601	$O\left(N.\frac{\sqrt{N}}{\ln\sqrt{N}}\right)$	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel terbentuk hingga 1201	Pembentukan tabel dari [2] hingga 1201.
	2	7879	0.002878	101.54	3939	3940	$O\left(N.\frac{\sqrt{N}}{\ln\sqrt{N}}\right)$	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel terbentuk hingga 7879	Pembentukan tabel dari [2] hingga 7879.
	3	9001	0.002390	102.05	4500	4501	$O\left(N.\frac{\sqrt{N}}{\ln\sqrt{N}}\right)$	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel terbentuk hingga 9001	Pembentukan tabel dari [2] hingga 9001.
Pengujian Pertama (perluasan Tabel)	1	1201	0.000272	101.54	600	601	$O\left(N.\frac{\sqrt{N}}{\ln\sqrt{N}}\right)$	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel terbentuk hingga 1201	Pembentukan tabel dari [2] hingga 1201.
	2	7879	0.003748	102.05	3339	3384	O(rentang kspansi × N)	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel meluas hingga 7879	Perluasan tabel dari 1201 hingga 7879.
	3	9001	0.000878	102.31	561	609	O(rentang kspansi × N)	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel meluas hingga 9001	Perluasan tabel dari 7879 hingga 9001.
Pengujian Kedua (Memanfaatkan Tabel)	1	9001	0.000007	102.05	0	0	O(logK) atau O(1)	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel tetap sama	Bilangan sudah ada di tabel, akses langsung.
	2	7879	0.000006	102.05	0	0	O(logK) atau O(1)	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel tetap sama	Bilangan sudah ada di tabel, akses langsung.
	3	1201	0.000004	102.05	0	0	O(logK) atau O(1)	$O\left(\frac{N_{max}}{\ln N_{mas}}\right)$	Tabel tetap sama	Bilangan sudah ada di tabel, akses langsung.

Sebagai pembanding diperlihatkan pada bagian pertama tabel, adalah eksekusi primalitas bilangan N dengan menggunakan metode trial division murni tanpa dynamic programming. Waktu eksekusi yang diperoleh untuk tiap-tiap penentuan primalitas bilangan memiliki waktu yang cukup besar, dengan kompleksitas iterasi perluasan sejumlah bilangan prima di bawah $\sqrt{N}$

sedangkan pada perluasan tabel, memang waktu yang dikonsumsi mengalami peningkatan, karena sambil melakukan pemeriksaan tabel bilangan prima, tetapi pada kompleksitasnya, ketika tabel diperluas, berhasil mengurangi iterasi yang menghasilkan perbedaan yang cukup besar. Sedangkan pada pencarian primalitas bilangan N pada N yang sudah tersimpan pada memorized tabel, waktu pencarian menjadi sangat singkat. Dan iterasi perluasan tabel yang bernilai 0, dikarenakan proses hanya memeriksa apakah bilangan N sudah terdapat pada memoized tabel.

Secara keseluruhan, pengujian ini secara empiris membuktikan bahwa strategi Dynamic Programming dengan memoization dan persistensi data melalui file biner sangat efektif dalam meningkatkan kecepatan pemeriksaan primalitas untuk bilangan yang telah atau dapat dijangkau oleh tabel bilangan prima yang sudah dibangun jika dibandingkan dengan pendekatan serupa yang tidak mempergunakan dynamic programming. Waktu komputasi yang signifikan hanya diperlukan pada saat pertama kali tabel perlu diperluas ke rentang bilangan yang lebih tinggi. Setelah tabel diperluas, pemeriksaan bilangan dalam rentang tersebut menjadi hampir instan, menunjukkan manfaat nyata dari pendekatan *Dynamic programming*.

4. KESIMPULAN

Program pemeriksaan primalitas ini berhasil menerapkan pendekatan *Dynamic Programming* dengan memanfaatkan penyimpanan tabel bilangan prima secara persisten. Ini memberikan keuntungan dalam efisiensi untuk pemeriksaan berulang atau bilangan yang tidak jauh melampaui batas tabel yang sudah ada. Namun, untuk pemeriksaan bilangan prima yang sangat besar atau untuk menghasilkan daftar bilangan prima dalam rentang yang sangat luas, metode perluasan tabel saat ini menjadi tidak efisien secara komputasi.

Saran

Untuk mengatasi keterbatasan kinerja dalam menghasilkan daftar bilangan prima dalam rentang yang sangat luas atau untuk memeriksa primalitas bilangan yang jauh lebih besar secara lebih efisien, saat ini sedang dilakukan penelitian pendekatan Sieve of Eratosthenes yang lebih cepat daripada *trial division*.

DAFTAR PUSTAKA

[1] P. Doroszlai and H. Keller, “The Number of Primes,” *Advances in Pure Mathematics*, vol. 12, no. 02, pp. 81–95, 2022, doi: 10.4236/apm.2022.122008

[2] J. Saputra and J. A. Ginting, “Penyembunyian Data Menggunakan Metode Overwriting Metadata,” *JIKO (Jurnal Informatika dan Komputer)*, vol. 7, no. 2, p. 296, Sep. 2023, doi: 10.26798/jiko.v7i2.692

[3] J. A. Ginting and I. G. G. Ngurah Suryantara, “PENGUJIAN KERENTANAN SISTEM DENGAN MENGGUNAKAN METODE PENETRATION TESTING DI UNIVERSITAS XYZ,” *Infotech: Journal of Technology Information*, vol. 7, no. 1, pp. 41–46, Jun. 2021, doi: 10.37365/jti.v7i1.105

[4] T. M. S. Mulyana, “PENGGUNAAN NILAI SKALA KEABUAN DARI CITRA WATERMARK SEBAGAI SEBAGAI CETAK BIRU DARI VISIBLE WATERMARKING Teady,” *Seminar Nasional Informatika*, vol. 2013, no. semnasIF, pp. 23–30, 2013, [Online]. Available: http://jurnal.upnyk.ac.id/index.php/semnasif/article/view/976

[5] M. S. Dairi, M. Setiani Asih, and corespondent author, “Implementasi Algoritma Kriptografi RSA Dalam Aplikasi Sistem Informasi Perpustakaaan Implementation Of RSA

Cryptographic Algorithms in Library Information System Applications," 2022. [Online]. Available: https://jurnal.unity-academy.sch.id/index.php/jirsi/index98

[6] N. P. Utomo, N. Fahriani, and M. Amirul, "Implementasi Kriptografi Dengan Metode RSA Untuk Keamanan Data Pada Email Berbasis PHP," 2023

[7] J. Maghfiroh, T. Turmudi, and E. Susanti, "Pengamanan Pesan Menggunakan Algoritma One Time Pad dengan Linear Congruential Generator sebagai Pembangkit Kunci," *Jurnal Riset Mahasiswa Matematika*, vol. 2, no. 3, pp. 122–131, Mar. 2023, doi: 10.18860/jrmm.v2i3.16770

[8] H. Gunawan, A. Setia Budi, and R. Primananda, "Penerapan Algoritma Diffie Hellman Key Exchange dalam Komunikasi Data Antarnode pada Wireless Sensor Network," 2022. [Online]. Available: http://j-ptiik.ub.ac.id

[9] M. Pundir, A. Kumar, and S. Choudhary, "Efficient Diffie Hellman Two Round Secret Key Agreement Protocol," in *2023 1st International Conference on Innovations in High Speed Communication and Signal Processing (IHCSP)*, 2023, pp. 7–10. doi: 10.1109/IHCSP56702.2023.10127113

[10] G. A. Onggo, D. E. Herwindiati, and J. Hendryli, "Analisis Security Voice Authentication pada Sistem Login 2-FA," *Computatio : Journal of Computer Science and Information Systems*, vol. 5, no. 1, p. 1, Sep. 2021, doi: 10.24912/computatio.v1i1.10915

[11] J. Fernandes Andry *et al.*, "KEBIJAKAN KEAMANAN TEKNOLOGI INFORMASI PADA PERANGKAT KERAS DI PERUSAHAAN DISTRIBUTOR SEPATU IT Security Policy of Hardware at A Shoe Distributor Company." [Online]. Available: http://journal.ubm.ac.id/

[12] A. Milson, D. E. Herwindiati, and N. J. Perdana, "Penerapan Klasifikasi Suara Sebagai Autentikasi Keamanan Sistem Login Menggunakan Gaussian Mixture Models," *Computatio : Journal of Computer Science and Information Systems*, vol. 8, no. 1, pp. 104–109, Apr. 2024, doi: 10.24912/computatio.v8i1.16229

[13] F. An, J. Ye, and Z. Yang, "Data Transmission Error Detection and Correction with Cyclic Redundancy Check and Polar Code Integration with Successive Cancellation Decoding Algorithm," *Applied Sciences*, vol. 15, no. 3, p. 1124, Jan. 2025, doi: 10.3390/app15031124

[14] A. Fanani, "PENGACAKAN SOAL PADA SISTEM COMPUTER BASED TEST (CBT) DENGAN METODE LINEAR CONGRUENTIAL GENERATOR (LCG) DI SMA NEGERI JOGOROTO," *SUBMIT (Jurnal Ilmiah Teknologi Informasi dan Sains)*, vol. I No. I, pp. 50–56, 2021, [Online]. Available: http://ejurnal.unim.ac.id/index.php/submit

[15] N. Khairina, "The Comparison of Methods for Generating Prime Numbers between The Sieve of Eratosthenes, Atkins, and Sundaram," *SinkrOn*, vol. 3, no. 2, p. 293, Apr. 2019, doi: 10.33395/sinkron.v3i2.10129

[16] K. Banerjee, S. Nath Mandal, and S. Kumar Das, "Improved Trial Division Technique for Primality Checking in RSA Algorithm," *International Journal of Computer Network and Information Security*, vol. 5, no. 9, pp. 51–57, Jul. 2013, doi: 10.5815/ijcnis.2013.09.07

[17] A. Elhakeem Abd Elnaby and A. H. El-Baz, "A new explicit algorithmic method for generating the prime numbers in order," *Egyptian Informatics Journal*, vol. 22, no. 1, pp. 101–104, Mar. 2021, doi: 10.1016/j.eij.2020.05.002

[18] I. G. A. W. Wardhana and A. Abdurahim, "SUBMODUL PRIMA, PRIMA LEMAH DAN HAMPIR PRIMA DARI MODUL MATRIKS BILANGAN BULAT MODULO," *Jurnal Riset dan Aplikasi Matematika (JRAM)*, vol. 8, no. 2, pp. 136–141, Oct. 2024, doi: 10.26740/jram.v8n2.p136-141

[19] I. D. Shkredov, I. E. Shparlinski, and A. Zaharescu, "On the distribution of modular square roots of primes," *Mathematische Zeitschrift*, vol. 306, no. 3, p. 43, Mar. 2024, doi: 10.1007/s00209-024-03436-5

[20] H.-L. Li, S.-C. Fang, and W. Kuo, "The Periodic Table of Primes," *Advances in Pure Mathematics*, vol. 14, no. 05, pp. 394–419, 2024, doi: 10.4236/apm.2024.145023

[21] A. Tarek, H. Elsayed, M. Rashad, M. Hassan, and P. el kafrawy, "Dynamic Programming Applications: A Suvrvey," in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, 2020, pp. 380–385. doi: 10.1109/NILES50944.2020.9257968

[22] N. Islamov, *Dynamic programming: Explore the concepts of dynamic programming, memoization, and tabulation, and apply these techniques to optimize solutions to problems like the knapsack problem, edit distance, and longest common subsequence*. 2023. doi: 10.13140/RG.2.2.29569.43369

[23] D. Llorens and J. M. Vilar, "Easily solving dynamic programming problems in Haskell by memoization of hylomorphisms," *Softw Pract Exp*, vol. 50, no. 12, pp. 2193–2211, Dec. 2020, doi: 10.1002/spe.2887

[24] S. S. Skiena, "Dynamic Programming," in *The Algorithm Design Manual*, S. S. Skiena, Ed., Cham: Springer International Publishing, 2020, pp. 307–353. doi: 10.1007/978-3-030-54256-6_10

[25] M. ERWIG and P. KUMAR, "Explainable dynamic programming," *Journal of Functional Programming*, vol. 31, p. e10, May 2021, doi: 10.1017/S0956796821000083

[26] T. Guo, "Using Larger Wheel to Accelerate the Trial Division Algorithm in Integer Factorization Method ToolBox," in *2024 7th International Conference on Computer Information Science and Application Technology (CISAT)*, 2024, pp. 136–139. doi: 10.1109/CISAT62382.2024.10695357

[27] I. D. Shkredov, I. E. Shparlinski, and A. Zaharescu, "On the distribution of modular square roots of primes," *Mathematische Zeitschrift*, vol. 306, no. 3, Mar. 2024, doi: 10.1007/s00209-024-03436-5

[28] F. Yuwono, V. Noviantri, A. A. S. Gunawan, and H. Ngarianto, "ANALISIS NUMERIK MODEL SEIQRS-V UNTUK PENYEBARAN VIRUS PADA JARINGAN KOMPUTER," *Computatio : Journal of Computer Science and Information Systems*, vol. 2, no. 2, p. 126, Oct. 2018, doi: 10.24912/computatio.v2i2.2347