

**PERBANDINGAN PERFORMANSI METODE STRING MATCHING  
MENGUNAKAN METODE *NAIVE STRING MATCHING*, *KNUTH  
MORRIS PRATT*, *BOYER MOORE*, *RABIN KARP* DAN SQL QUERY LIKE  
UNTUK PENCARIAN DATA KONSUMEN**

**SKRIPSI**

Disusun sebagai salah satu syarat untuk memperoleh Gelar Sarjana Teknik pada  
Program Studi Teknik Informatika Universitas Sangga Buana YPKP

Disusun Oleh :

**ISEP LUTPI NUR**

**2113191079**



**FAKULTAS TEKNIK  
PROGRAM STUDI TEKNIK INFORMATIKA  
UNIVERSITAS SANGGA BUANA YPKP  
2023**

## **LEMBAR PERSEMBAHAN**

Teriring puji dan syukur kepada Allah SWT. Karya tulis ini saya persembahkan kepada:

1. Kepada Orang tua beserta keluarga yang selalu mendorong semangat dan motivasi, sehingga penulisan skripsi ini dapat diselesaikan tepat waktu.
2. Seorang gadis yang selalu mendorong semangat, memberikan dukungan, kesabaran dan motivasi yang tiada henti “Siti Nurlaela”.
3. Kawan-kawan seperjuangan, CV. Adikarya Infinit (Soni Setiawan, M. Ilham Solehudin, Adje Abdul Azis, Fiqih Kholifah, M. Taufiq Hidayatuloh dan Handip Yusuf Kurniawan).
4. Kawan-kawan seperjuangan, mahasiswa teknik informatika angkatan 2019 yang sama-sama saling bahu membahu untuk berjuang hingga saat ini. Terutama Farhan Aziz, Vakrun Nisah dan lainnya.
5. Kawan-kawan seperjuangan, CEMPOR Dinas Pemuda dan Olahraga Kota Bandung (Mohammad Zamzam Badaruzzaman, SE. dan Moch Ibnu Sina Alzaenabi).
6. Kepada orang-orang dibelakang saya yang selalu mendukung dan membantu disaat situasi yang berat.

## LEMBAR PERNYATAAN KEASLIAN SKRIPSI

Yang bertanda tangan di bawah ini:

Nama : Isep Lutpi Nur  
NPM : 2113191079  
Program Studi : Teknik Informatika  
Alamat : Kp. Tipar Rt/Rw 23/12, Desa. Mekarwangi, Kec. Cikadu,  
Kab. Cianjur Jawa Barat 43272

Dengan ini menyatakan bahwa penelitian yang saya buat dengan judul **“PERBANDINGAN PERFORMANSI METODE STRING MATCHING MENGGUNAKAN METODE NAIVE STRING MATCHING, KNUTH MORRIS PRATT, BOYER MOORE, RABIN KARP DAN SQL QUERY LIKE UNTUK PENCARIAN DATA KONSUMEN”** adalah asli atau tidak menjiplak (plagiat) dan belum pernah di publikasikan di mana pun dan dalam bentuk apa pun.

Demikian surat pernyataan ini saya buat dengan sebenarnya tanpa ada paksaan dan tekanan dari pihak mana pun dan apabila dikemudian hari ternyata ada pihak lain yang mengklaim judul dan isi penelitian ini atau saya memberi keterangan palsu maka saya bersedia kelulusan saya dari program studi Teknik Informatika dibatalkan.

Dibuat di : Bandung  
Tanggal : 10 Agustus 2023  
Yang Menyatakan,

**Isep Lutpi Nur**  
NPM. 2113191079

## LEMBAR PERSETUJUAN DAN PENGESAHAN SKRIPSI

Yang bertanda tangan di bawah ini:

Nama : Isep Lutpi Nur  
NPM : 2113191079  
Program Studi : Teknik Informatika  
Judul : PERBANDINGAN PERFORMANSI METODE STRING  
MATCHING MENGGUNAKAN METODE NAIVE  
STRING MATCHING, KNUTH MORRIS PRATT,  
BOYER MOORE, RABIN KARP DAN SQL QUERY  
LIKE UNTUK Pencarian Data Konsumen

Untuk dipertahankan pada sidang Skripsi Semester Ganjil Tahun 2023 di hadapan para penguji dan diterima sebagai bagian dari persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik (S.T) pada Fakultas Teknik program Studi S1 Teknik Informatika Universitas Sangga Buana YPKP.

Bandung, 10 Agustus 2023  
Menyetujui,  
Pembimbing

**Gunawansyah, S.T., M.Kom., MOM., MCP., MTA.**  
NIDN: 0420027907

Penguji I

Penguji II

**Riffa Haviani Laluma, S.Kom., M.T., MTA.**  
NIDN: 0011067301

**Gunawan, S.T., M.Kom., MOS., MTA., MCE.**  
NIDN: 0404027604

Mengetahui:  
Ketua Program Studi S1 Teknik Informatika

**Gunawan, S.T., M.Kom., MOS., MTA., MCE.**  
NIDN: 0404027604

## KATA PENGANTAR

Segala puji dan syukur kehadiran Allah SWT atas berkah, rahmat dan hidayah-Nya yang senantiasa dilimpahkan kepada penulis. Sehingga dapat menyelesaikan skripsi dengan judul “*Perbandingan Performansi Metode String Matching Menggunakan Metode Naive String Matching, Knuth Morris Pratt, Boyer Moore, Rabin Karp dan SQL Query Like Untuk Pencarian Data Konsumen*” sebagai syarat untuk memperoleh Gelar Sarjana Teknik pada Program Studi Teknik Informatika Universitas Sangga Buana YPKP.

Dalam penyusunan skripsi ini banyak hambatan serta rintangan yang penulis hadapi namun pada akhirnya dapat melaluinya berkat adanya bimbingan dan bantuan dari berbagai pihak baik secara moral maupun spiritual. Untuk itu pada kesempatan ini penulis menyampaikan ucapan terima kasih kepada :

1. Kedua orang tua, yang telah menjadi motivasi kuat dan memberikan dukungan baik moral maupun material serta doa yang tiada henti-hentinya kepada penulis.
2. Gunawan, S.T., M.Kom., MOS., MTA., MCE. selaku Ketua Program Studi Jurusan S-1 Teknik Informatika.
3. Gunawansyah, S.T., M.Kom., MOM., MCP., MTA. selaku dosen pembimbing dalam penulisan skripsi ini yang telah memberikan sumbangan pemikiran. Terima kasih atas dukungan, pendidikan, kesabaran dan bimbingannya.
4. Semua pihak yang tidak dapat penulis sebutkan satu persatu yang telah membantu selama perkuliahan hingga terselesaikannya skripsi ini.

Penulis menyadari sepenuhnya bahwa dalam penyusunan skripsi ini masih jauh dari sempurna dan memiliki banyak kekurangan baik dalam teknik penulisan, penyajian materi, maupun pembahasan yang dikarenakan oleh keterbatasan penulis. Dengan demikian, penulis berharap adanya saran dan kritik yang membangun sehingga skripsi ini diharapkan dapat memberikan manfaat bagi semua pihak terutama bagi penulis.

## ABSTRAK

*Aktivitas mencari data atau informasi adalah yang umum dilakukan oleh banyak individu. Salah satu jenis pencarian data yang sering digunakan adalah string matching atau pola dalam sebuah teks. Pada umumnya string matching menggunakan metode SQL Query Like. Dalam era digital seperti sekarang ini, pencarian data atau informasi menjadi sangat penting dan sering dilakukan oleh banyak orang. Terdapat beberapa metode algoritma yang dapat digunakan dalam pencarian string, antara lain Boyer Moore, Naive String Matching, Knuth-Morris-Pratt dan Rabin-Karp Algorithm. Dari pengujian yang telah dilakukan didapatkan hasil algoritma Boyer More lebih cepat dibanding algoritma yang lain dengan selisih 104ms dengan SQL Query Like atau 39.25% lebih cepat berdasarkan pengujian performansi dengan skenario kasus rata-rata untuk pencarian data dengan pattern/kata kunci “ANDY” selama 5 kali dengan jumlah data 250,000. Kemudian untuk penggunaan memory rata-rata memori yang digunakan SQL lebih banyak dibandingkan algoritma yang lain.*

**Kata Kunci:** *String Matching, Boyer Moore, Naive String Matching, Knuth-Morris-Pratt, Rabin-Karp Algorithm.*

## ABSTRACT

*The activity of searching for data or information is a common practice among numerous individuals. One common type of data search is string matching, a pattern in a text. Generally, string matching utilizes SQL Query Like method. In the digital age such as now, data or information searching has become extremely crucial and is regularly performed by many people. There are several algorithmic methods that can be used in string search, among them are Boyer Moore, Naive String Matching, Knuth-Morris-Pratt, and Rabin-Karp Algorithm. From tests that have been carried out, it was found that the Boyer Moore algorithm is faster than the other algorithms by a margin of 161.8ms with SQL Query Like, or 39.25% faster based on performance testing with average case scenarios for data searches with the pattern/keyword "ANDY" performed 5 times with a data count of 250,000. Then, for memory usage, on average, SQL uses more.*

**Keywords:** *String Matching, Boyer Moore, Naive String Matching, Knuth-Morris-Pratt, Rabin-Karp Algorithm.*

## DAFTAR ISI

LEMBAR PERSEMBAHAN .....	i
LEMBAR PERNYATAAN KEASLIAN SKRIPSI .....	ii
LEMBAR PERSETUJUAN DAN PENGESAHAN SKRIPSI .....	iii
KATA PENGANTAR .....	iv
ABSTRAK .....	v
ABSTRACT .....	vi
DAFTAR ISI .....	vii
DAFTAR GAMBAR .....	x
DAFTAR TABEL .....	xiii
BAB I PENDAHULUAN .....	1
1.1. Latar Belakang Masalah .....	1
1.2. Rumusan Masalah .....	3
1.3. Batasan Masalah .....	3
1.4. Maksud dan Tujuan .....	4
1.5. Metode Penelitian .....	4
A. Metode Pengumpulan Data .....	4
B. Metode Pengembangan Sistem .....	5
C. Metode Pengembangan Perangkat Lunak .....	5
1.6. Sistematika Penulisan .....	6
BAB II LANDASAN TEORI .....	7
2.1. Data Konsumen .....	7
2.2. Algoritma .....	7
2.3. Algoritma <i>String Matching</i> .....	8
2.4. Algoritma <i>Naive String Matching</i> .....	8
2.5. Algoritma <i>Knuth Morris Pratt</i> .....	8
2.6. Algoritma <i>Boyer Moore</i> .....	9
A. Mekanisme umum algoritma Boyer Moore .....	10
B. Fungsi Last Occurrence .....	13
2.7. Algoritma <i>Rabin Karp</i> .....	14



2.8.	SQL Query Like.....	14
2.9.	UML (Unified Modelling Language) .....	15
A.	<i>Use Case Diagram</i> .....	15
B.	<i>Activity Diagram</i> .....	16
C.	<i>Sequence Diagram</i> .....	17
D.	<i>Class Diagram</i> .....	18
2.10.	PHP (PHP: Hypertext Preprocessor) .....	19
2.11.	Basis Data (MYSQL).....	19
2.12.	Penelitian Terdahulu .....	20
BAB III ANALISIS DAN PERANCANGAN SISTEM .....		23
3.1.	Sistem Berjalan .....	23
3.2.	Sistem yang diusulkan .....	23
A.	Data Penelitian .....	24
B.	Data Preprocessing .....	25
C.	Metode penelitian .....	26
3.3.	Perancangan UML .....	41
A.	Use Case Diagram .....	41
B.	Activity Diagram .....	45
3.4.	Perancangan Antarmuka .....	48
D.	Rancangan Tampilan Halaman Testing .....	48
E.	Rancangan Tampilan Halaman Daftar Hasil Testing .....	48
F.	Rancangan Tampilan Halaman Detail Hasil Testing Sebelumnya.....	49
3.5.	Skenario Pengujian .....	50
A.	Variabel Pengujian .....	50
B.	Parameter yang Digunakan.....	50
C.	Skenario.....	51
BAB IV IMPLEMENTASI DAN PENGUJIAN SISTEM.....		52
4.1.	Batasan Implementasi .....	52
A.	Komputer perangkat keras .....	52
B.	Perangkat lunak .....	52
4.2.	Implementasi Antarmuka.....	53

A. Halaman Testing.....	53
2. Halaman Daftar Hasil Testing .....	54
3. Halaman Detail Hasil Testing Sebelumnya.....	54
4.3. Pengujian Sistem.....	55
A. Pengujian Blackbox.....	55
B. Pengujian Beta.....	57
C. Hasil Penelitian.....	61
BAB V PENUTUP.....	68
5.1. Kesimpulan .....	68
5.2. Saran .....	68
DAFTAR PUSTAKA .....	69
LAMPIRAN.....	71
A. Kartu Bimbingan.....	71
B. Surat Pernyataan Keabsahan Data Penelitian .....	72
C. Kuisisioner Pengujian Beta .....	73
D. Kode Program Algoritma String Matching.....	74
1. Algoritma Naïve String Matching .....	74
2. Algoritma Knuth Morris Pratt .....	75
3. Algoritma Boyer Moore .....	77
4. Rabin Karp .....	78

## DAFTAR GAMBAR

Gambar 1. 1 Metode Pengembangan Perangkat Lunak .....	5
Gambar 1. 2 Contoh Use Case Diagram .....	16
Gambar 2. 1 The looking-glass technique.....	10
Gambar 2. 2 Kasus 1 - The character-jump technique.....	11
Gambar 2. 3 Kasus 2 - The character-jump technique.....	12
Gambar 2. 4 Kasus 3 - The character-jump technique.....	12
Gambar 2. 5 Variasi karakter Boyer moore .....	13
Gambar 2. 6 Last Occurrence.....	13
Gambar 2. 7 Contoh Activity Diagram .....	17
Gambar 2. 8 Contoh Sequence Diagram.....	18
Gambar 2. 9 Contoh Class Diagram .....	18
Gambar 3. 1 Pencarian Data Menggunakan MySQL Dengan PHP.....	23
Gambar 3. 2 Sistem yang diusulkan.....	24
Gambar 3. 3 Variasi karakter Boyer moore .....	26
Gambar 3. 4 Tabel Last Occurence.....	26
Gambar 3. 5 Langkah 1 contoh algoritma naive string matching.....	27
Gambar 3. 6 Langkah 2 contoh algoritma naive string matching.....	28
Gambar 3. 7 Langkah 3 contoh algoritma naive string matching.....	28
Gambar 3. 8 Langkah 4 contoh algoritma naive string matching.....	28
Gambar 3. 9 Langkah 5 contoh algoritma naive string matching.....	29
Gambar 3. 10 Langkah 6 contoh algoritma naive string matching.....	29
Gambar 3. 11 Langkah 1 contoh algoritma KMP .....	32
Gambar 3. 12 Langkah 2 contoh algoritma KMP .....	32
Gambar 3. 13 Langkah 3 contoh algoritma KMP .....	32
Gambar 3. 14 Langkah 4 contoh algoritma KMP .....	33
Gambar 3. 15 Langkah 5 contoh algoritma KMP .....	33
Gambar 3. 16 Langkah 6 contoh algoritma KMP .....	33
Gambar 3. 17 Variasi karakter Boyer moore .....	34
Gambar 3. 18 Last Occurrence.....	34

Gambar 3. 19 Langkah 1 Contoh Algoritma Boyer Moore .....	35
Gambar 3. 20 Langkah 2 Contoh Algoritma Boyer Moore .....	35
Gambar 3. 21 Langkah 3 Contoh Algoritma Boyer Moore .....	35
Gambar 3. 22 Langkah 4 Contoh Algoritma Boyer Moore .....	36
Gambar 3. 23 Langkah 5 Contoh Algoritma Boyer Moore .....	36
Gambar 3. 24 Langkah 6 Contoh Algoritma Boyer Moore .....	37
Gambar 3. 25 Langkah 1 Contoh Algoritma Rabin Karp .....	37
Gambar 3. 26 Langkah 2 Contoh Algoritma Rabin Karp .....	38
Gambar 3. 27 Langkah 3 Contoh Algoritma Rabin Karp .....	38
Gambar 3. 28 Langkah 4 Contoh Algoritma Rabin Karp .....	39
Gambar 3. 29 Langkah 5 Contoh Algoritma Rabin Karp .....	39
Gambar 3. 30 Langkah 6 Contoh Algoritma Rabin Karp .....	39
Gambar 3. 31 Langkah 7 Contoh Algoritma Rabin Karp .....	40
Gambar 3. 32 Langkah 8 Contoh Algoritma Rabin Karp .....	40
Gambar 3. 33 Rancangan UML Use Case Diagram .....	42
Gambar 3. 34 Activity Diagram Testing Algoritma .....	45
Gambar 3. 35 Activity Diagram Simpan Hasil Testing .....	46
Gambar 3. 36 Activity Diagram Lihat Daftar Hasil Testing.....	46
Gambar 3. 37 Activity Diagram Hapus Hasil Testing .....	47
Gambar 3. 38 Activity Diagram Detail Hasil Testing Sebelumnya.....	47
Gambar 3. 39 Rancangan Tampilan Halaman Testing .....	48
Gambar 3. 40 Rancangan Tampilan Halaman Daftar Hasil Testing.....	48
Gambar 3. 41 Rancangan Tampilan Halaman Detail Hasil Testing Sebelumnya	49
Gambar 4. 1 Halaman Testing Tabel 10.000-100.000 Data .....	53
Gambar 4. 2 Halaman Testing Diagram Garis Kecepatan Waktu dan Penggunaan Memory .....	53
Gambar 4. 3 Halaman Daftar Hasil Testing .....	54
Gambar 4. 4 Halaman Detail Testing Sebelmunya - Tabel 100.000-250.000 Data .....	54
Gambar 4. 5 Halaman Testing Sebelumnya - Diagram Garis Kecepatan Waktu dan Penggunaan Memory.....	55

Gambar 4. 6 Hasil Pengujian dengan skenario kasus rata-rata (Kecepatan).....	62
Gambar 4. 7 Hasil Pengujian dengan skenario kasus rata-rata (Memori).....	63
Gambar 4. 8 Pengujian Dengan Karakteristik Teks Khusus (Kecepatan) .....	63
Gambar 4. 9 Pengujian Dengan Karakteristik Teks Khusus (Memory) .....	64
Gambar 4. 10 Hasil Skenario Pengujian Dengan Banyak Pengguna.....	65
Gambar 4. 11 Hasil pengujian Memory dengan 5 pengguna sekaligus.....	66
Gambar 4. 12 Pengujian dengan dataset yang diambil dari MySQL langsung tanpa Temporary JSON .....	67
Gambar 4. 13 Pengujian dengan dataset yang diambil dari MySQL langsung tanpa Temporary JSON (Memori).....	67

## DAFTAR TABEL

Tabel 2. 1 Penelitian Terdahulu .....	20
Tabel 3. 1 Daftar beberapa data konsumen yang akan digunakan. ....	24
Tabel 3. 2 Border Function .....	25
Tabel 3. 3 Tabel Customer .....	41
Tabel 3. 4 Skenario use case mengelola data penyakit .....	42
Tabel 3. 5 Skenario Usecase Lihat Daftar Hasil Testing .....	43
Tabel 3. 6 Skenario use case Simpan hasil testing .....	43
Tabel 3. 7 Skenario Use Case Hapus Hasil Testing .....	44
Tabel 3. 8 Skenario Use Case Hapus Hasil Testing .....	44
Tabel 4. 1 Komputer perangkat keras .....	52
Tabel 4. 2 Perangkat Lunak .....	52
Tabel 4. 3 Hasil pengujian blackbox pada halaman Testing .....	56
Tabel 4. 4 Halaman Daftar Testing .....	56
Tabel 4. 5 Halaman Detail Testing Sebelumnya .....	57
Tabel 4. 6 Skala likert dan interval .....	57
Tabel 4. 7 Daftar pertanyaan .....	58
Tabel 4. 8 Kuesioner pertanyaan 1 .....	58
Tabel 4.9 Kuesioner pertanyaan 2 .....	59
Tabel 4.10 Kuesioner pertanyaan 3 .....	59
Tabel 4.11 Kuesioner pertanyaan 4 .....	60
Tabel 4.12 Kuesioner pertanyaan 5 .....	60
Tabel 4. 13 Hasil Pengujian dengan skenario kasus rata-rata (Kecepatan) .....	61
Tabel 4. 14 Hasil Pengujian dengan skenario kasus rata-rata (Memori) .....	62
Tabel 4. 15 Hasil Skenario Pengujian Dengan Banyak Pengguna .....	64
Tabel 4. 16 Pengujian dengan dataset yang diambil dari MySQL langsung tanpa Temporary JSON .....	66

# BAB I

## PENDAHULUAN

### 1.1. Latar Belakang Masalah

Teknologi informasi dan komunikasi telah mengubah cara orang mencari, memproses, dan mengakses informasi. Dalam era digital ini data dapat ditemukan dengan lebih mudah dan cepat daripada sebelumnya. Hal ini memberikan dampak signifikan pada banyak aspek kehidupan manusia, termasuk dalam bidang pendidikan, bisnis, hiburan, dan lain-lain.

Dalam era digital seperti sekarang ini, pencarian data atau informasi menjadi sangat penting dan sering dilakukan oleh banyak orang. Seiring dengan semakin banyaknya data yang tersedia, pencarian data menjadi semakin kompleks dan memerlukan algoritma yang efektif dan efisien untuk dapat mencari data dengan cepat.

Data konsumen menjadi salah satu faktor penting dalam bisnis. Data konsumen dapat membantu perusahaan dalam mengambil keputusan yang lebih tepat dan memaksimalkan potensi bisnis. Salah satu cara untuk mengumpulkan data konsumen adalah dengan melakukan pencarian data pada *database* perusahaan.

Salah satu jenis pencarian data yang sering digunakan adalah pencarian *string* atau pola dalam sebuah teks atau data. Pencarian *string* dapat digunakan dalam berbagai aplikasi, seperti deteksi *plagiarism*, analisis teks, pencarian teks di dalam *database*, dan lain sebagainya.

Terdapat penelitian *string matcing* untuk Pencarian Informasi Data Zakat dan Aktivitas Sosial dengan menggunakan algoritma *Knuth Morris Pratt*. Dalam penelitian tersebut menunjukkan bahwa performa algoritma KMP sangat baik dengan rata-rata waktu eksekusi dalam lima kali pengujian yaitu 0.03 ms, 0.03 ms, 0.02 ms, 0.02 ms dan 0.03 ms [1].

Terdapat beberapa metode algoritma yang dapat digunakan dalam pencarian string, antara lain *Boyer Moore*, *Naive String Matching*, *Knuth-Morris-Pratt* dan *Rabin-Karp Algorithm*. Masing-masing algoritma memiliki kelebihan dan kelemahan tertentu, serta berbeda dalam hal waktu eksekusi dan *space complexity*.

Penelitian ini bertujuan untuk membandingkan performa algoritma string matching yang populer, yaitu *Naive String Matching*, *Knuth Morris Pratt*, *Boyer Moore*, dan *Rabin-Karp* dalam mencari pola pada sebuah *string*. Performa algoritma akan diukur menggunakan metrik seperti waktu eksekusi dan *space complexity*. Penelitian ini juga akan membahas tentang performa algoritma *SQL Query Like* dalam mencari pola pada data yang tersimpan di dalam *database* dan membandingkannya dengan algoritma *string matching* yang telah disebutkan sebelumnya.

Dalam penelitian ini, akan dikaji kelebihan dan kelemahan masing-masing algoritma string matching dan *SQL Query Like* dalam konteks pencarian data konsumen. Selain itu, penelitian ini juga akan memberikan rekomendasi mengenai algoritma yang tepat digunakan dalam pencarian data berdasarkan karakteristik data dan kebutuhan pengguna.

Diharapkan hasil penelitian ini dapat memberikan kontribusi dalam pengembangan teknologi pencarian data di masa depan, terutama dalam hal pengembangan algoritma string matching dan *SQL Query Like* yang lebih efektif dan efisien. Selain itu, diharapkan penelitian ini dapat memberikan kontribusi dalam bidang ilmu komputer khususnya dalam pengolahan string dan algoritma. Oleh karena itu, pada penelitian ini dibuat dengan judul “PERBANDINGAN PERFORMANSI METODE STRING MATCHING MENGGUNAKAN METODE NAIVE STRING MATCHING, KNUTH MORRIS PRATT, BOYER MOORE, RABIN KARP DAN SQL QUERY LIKE UNTUK PENCARIAN DATA KONSUMEN”



## 1.2. Rumusan Masalah

Berdasarkan uraian di atas, terdapat beberapa hal yang dapat disimpulkan untuk dijadikan sebagai rumusan masalah yang selanjutnya akan dibuatkan laporan penelitian ini di antaranya:

1. Bagaimana performa dari algoritma string matching *Naive String Matching*, *Boyer Moore*, *Knuth Morris Pratt*, *Rabin-Karp* dan *SQL Query Like* dalam mencari pola pada sebuah string untuk mencari data konsumen dalam jumlah besar?
2. Bagaimana mengukur Performa algoritma tersebut dengan menggunakan metrik seperti waktu eksekusi dan *space complexity*?
3. Bagaimana menentukan algoritma yang paling tepat dalam pencarian data berdasarkan karakteristik data dan kebutuhan pengguna?

## 1.3. Batasan Masalah

Ada beberapa batasan masalah dalam menyelesaikan penelitian ini yaitu sebagai berikut :

1. Penelitian ini hanya akan membahas performa dari algoritma string matching *SQL Query Like* dan *Naive String Matching*, *Boyer Moore*, *Knuth Morris Pratt*, *Rabin-Karp* dan *SQL Query Like* dalam mencari pola pada sebuah string.
2. Penelitian ini dilakukan dengan membandingkan waktu eksekusi dan space complexity yang dibutuhkan algoritma tersebut.
3. Penelitian ini hanya akan membahas algoritma string matching *SQL Query Like* dan *Naive String Matching*, *Boyer Moore*, *Knuth Morris Pratt*, *Rabin-Karp* dan *SQL Query Like* yang paling tepat dalam pencarian data berdasarkan karakteristik data dan kebutuhan pengguna.

#### **1.4. Maksud dan Tujuan**

Maksud penelitian ini adalah untuk melakukan perbandingan dari beberapa algoritma *string matcing*.

Kemudian tujuan dari penelitian ini adalah:

1. Membandingkan performa dari lima algoritma *string matching* yang populer, yaitu *SQL Query Like* dan *Naive String Matching*, *Knuth Morris Pratt*, *Boyer Moore*, *Rabin-Karp* dalam mencari pola pada sebuah string.
2. Memberikan rekomendasi mengenai algoritma yang tepat digunakan dalam pencarian data berdasarkan karakteristik data dan kebutuhan pengguna khususnya yang memiliki data berukuran besar dan diakses oleh banyak pengguna sekaligus misalnya toko online.

#### **1.5. Metode Penelitian**

Metodologi penelitian yang digunakan terdiri dari tiga komponen utama, yaitu metode pengumpulan data, metode pembangunan sistem, dan metode pengembangan perangkat lunak.

##### **A. Metode Pengumpulan Data**

Dalam penelitian ini, metode pengambilan data yang digunakan adalah sebagai berikut:

1. Studi pustaka

Pada tahapan ini, penulis mengeksplorasi referensi teoretis yang berkaitan dengan studi kasus atau masalah yang dihadapi. Sumber-sumber referensi ini berasal dari jurnal, buku, situs web, dan artikel laporan penelitian.

2. Eksperimen

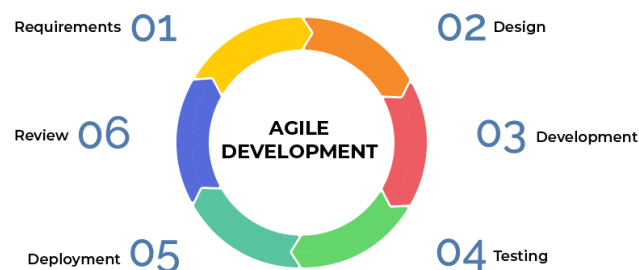
Proses pengumpulan data dalam penelitian ini melibatkan penerapan eksperimental pada subyek tertentu, yang mencakup pemilihan sampel objek (dataset), diikuti oleh kegiatan observasi, pelatihan, dan pencatatannya yang relevan dengan penelitian.

## B. Metode Pengembangan Sistem

Dalam penelitian ini, pendekatan yang digunakan dalam pengembangan sistem melibatkan lima metode pencocokan string, yaitu Algoritma Naive String Matching, Knuth Morris Pratt, Boyer Moore, Rabin-Karp, dan SQL Query Like.

## C. Metode Pengembangan Perangkat Lunak

Dalam penelitian ini, metode pengembangan perangkat lunak yang diterapkan adalah metode Agile. Metode ini menitikberatkan pada keunggulan teknis dalam proses pengembangan perangkat lunak dan menghargai kesederhanaan untuk mengoptimalkan penggunaan sumber daya. Tiap tim pengembang yang menerapkan metode Agile melaksanakan introspeksi untuk meningkatkan efektivitas kerja dan menciptakan pola kerja yang optimal. [2].



Gambar 1. 1 Metode Pengembangan Perangkat Lunak

Adapun tahapan – tahapan dari metode *Agile* adalah sebagai berikut :

1. **Requirements:** Langkah pertama untuk mengidentifikasi kebutuhan dalam pengembangan.
2. **Design:** pada langkah ini dilakukan dalam desain visual dan arsitektur aplikasi.
3. **Development:** Tahapan ini untuk penulisan kode dan tulang punggung dari keseluruhan proses.
4. **Testing:** Langkah ini untuk pengujian dan menentukan kualitas dari perangkat lunak yang dibuat.
5. **Development:** Langkah ini untuk peluncuran perangkat lunak ke pengguna.
6. **Review:** Merupakan langkah untuk menilai atau mengulas aplikasi dan meninjau perangkat lunak yang dibangun.

## **1.6. Sistematika Penulisan**

Dalam penelitian ini, sistematika penulisannya dibagi menjadi sejumlah bab, di antaranya:

### **BAB I: PENDAHULUAN**

Bagian pendahuluan mencakup konteks dari persoalan yang ditangani, perumusan isu, pembatasan masalah, maksud dan tujuan, metodologi penelitian, serta struktur penulisan.

### **BAB II: LANDASAN TEORI**

Bagian Landasan Teori mencakup pembahasan mengenai teori-teori yang berkaitan dengan penelitian yang sedang dilaksanakan, termasuk konsep dasar pengembangan sistem, prinsip-prinsip fundamental pemrograman, dan alat pendukung sistem yang menjadi acuan dalam melaksanakan penelitian.

### **BAB III: ANALISIS DAN PERANCANGAN**

Bagian Analisis dan Perancangan berisi penjelasan rinci mengenai analisis dan rancangan sistem yang akan ditetapkan, yang mencakup analisis sistem berjalan, sistem yang diajukan, perancangan UML, rancangan basis data, dan perancangan antarmuka.

### **BAB IV: IMPLEMENTASI DAN PENGUJIAN SISTEM**

Bagian ini mencakup proses implementasi, pengujian, serta analisis terhadap hasil penelitian yang diarahkan kepada sistem yang telah dikonstruksi. Tujuan ini adalah untuk memahami seberapa efektif sistem tersebut dalam menyelesaikan permasalahan saat ini dan apakah ia sesuai dengan objektif dari penelitian ini.

### **BAB V: PENUTUP**

Bagian ini mengandung kesimpulan yang ditarik dari pelaksanaan penelitian ini. Penutup juga mencakup rekomendasi, yang mana rekomendasi tersebut diharapkan dapat memberikan manfaat bagi penyempurnaan sistem di masa mendatang.

## **BAB II**

### **LANDASAN TEORI**

#### **2.1. Data Konsumen**

Data konsumen adalah informasi yang di terima saat konsumen berinteraksi dengan media bisnis. Media ini dapat berupa situs web, aplikasi seluler, halaman media sosial, halaman survei, kampanye, dan versi online atau offline lainnya dari upaya pemasaran bisnis [3].

Data konsumen penting karena memungkinkan bisnis untuk mengoptimalkan platform melalui pemahaman berbagai aspek interaksi konsumen. Dengan menggunakan data konsumen, bisnis tidak hanya dapat merampingkan situs web yang mereka miliki dan tunjukkan kepada konsumen, tetapi mereka juga dapat mengoptimalkan iklan dan saran produk untuk konsumen tertentu menggunakan data mereka.

Data konsumen yang berjumlah besar biasanya terdapat di toko online. Data konsumen di toko online dapat mencakup berbagai jenis informasi yang dikumpulkan dari interaksi konsumen dengan platform toko online seperti Data Informasi pribadi, Data transaksi, Data Produk dan Data Perilaku.

#### **2.2. Algoritma**

Algoritma adalah urutan instruksi atau aturan yang sistematis, terstruktur, dan terdefinisi dengan baik untuk menyelesaikan sebuah masalah atau tugas secara efektif dan efisien. Algoritma digunakan dalam berbagai bidang, termasuk matematika, ilmu komputer, teknologi informasi, fisika, dan banyak lagi [4].

Algoritma sering digunakan dalam pemrograman komputer sebagai panduan untuk menyelesaikan suatu tugas atau masalah tertentu. Algoritma dapat ditulis dalam bahasa yang mudah dimengerti oleh manusia, yang kemudian diterjemahkan ke dalam bahasa yang dimengerti oleh komputer, seperti bahasa pemrograman.

Algoritma biasanya memiliki tujuan tertentu dan dapat digunakan untuk menyelesaikan berbagai masalah, mulai dari yang sederhana hingga yang kompleks. Beberapa contoh masalah yang dapat diselesaikan dengan algoritma

adalah sorting(pengurutan) data, pencarian data, enkripsi data, dan optimisasi pemrosesan data.

### **2.3. Algoritma *String Matching***

Algoritma *string matching* adalah algoritma untuk mencari keberadaan sebuah pola atau substring dalam sebuah string. Pola atau substring tersebut dapat berupa satu karakter atau beberapa karakter yang harus ditemukan dalam sebuah string.

Algoritma string matching sangat umum digunakan dalam pengembangan perangkat lunak, seperti dalam aplikasi pencarian atau manipulasi string. Contoh penggunaan algoritma string matching adalah ketika kita ingin mencari sebuah kata tertentu dalam sebuah teks [5].

### **2.4. Algoritma *Naive String Matching***

Naive string matching adalah salah satu algoritma string matching yang paling sederhana dan mudah dipahami. Algoritma naive string matching membandingkan setiap karakter dari sebuah pola dengan setiap karakter di dalam sebuah string. Algoritma ini memulai dengan menempatkan pola di awal string dan kemudian membandingkan karakter demi karakter. Jika karakter-karakter tidak cocok, maka pola dipindahkan satu karakter ke kanan dan proses pencocokan kembali dilakukan [6].

Algoritma naive string matching memiliki kompleksitas waktu sebesar  $O(mn)$ , di mana  $m$  adalah panjang pola dan  $n$  adalah panjang string. Algoritma ini bekerja dengan baik pada string yang relatif pendek dan pola yang singkat.

### **2.5. Algoritma *Knuth Morris Pratt***

Algoritma Knuth-Morris-Pratt (KMP) adalah metode efisien yang digunakan dalam teknik pencarian string dalam komputasi. Dikembangkan oleh Donald Knuth, Vaughan Pratt, dan James H. Morris pada tahun 1977, teknik ini

mengeliminasi kebutuhan untuk mundur ke titik awal dalam string yang dicari, sehingga meningkatkan efisiensi pencarian secara signifikan.

Dalam konteks pencarian string, algoritma konvensional sering menghadapi kendala berupa berulang kali mengulang proses pencarian dari awal string setelah setiap kesalahan yang ditemukan. Ini menghasilkan efisiensi yang rendah. Algoritma KMP, sebaliknya, memindahkan titik pencarian maju setelah setiap kesalahan, sehingga mengurangi repetisi.

Algoritma KMP mengandalkan teknik bernama "preprocessing". Dalam tahap ini, algoritma membangun "fungsi pinggiran" (Border Function) dari string yang dicari, yang digunakan untuk menentukan sejauh mana kursor harus digeser jika ada ketidakcocokan.

Berikut adalah langkah-langkah operasional algoritma KMP:

1. Mulai dari kiri, bandingkan setiap karakter string yang dicari dengan string yang sedang dikombinasikan.
2. Jika ada ketidakcocokan:
  - Lihat tabel lompatan untuk karakter yang tidak cocok.
  - Geser string yang dicari sejauh yang ditentukan oleh tabel, atau geser ke kanan jika tidak ada entri di tabel.
3. Ulangi proses ini sampai string yang dicari dipindahkan sepenuhnya melalui string yang sedang dikombinasikan. [7].

## **2.6. Algoritma *Boyer Moore***

Algoritma Boyer Moore diciptakan oleh Bob Boyer dan J. Strother Moore pada tahun 1977. Algoritma ini mencocokkan string mulai dari akhir string kunci, bergerak menuju awalnya. Bila terdapat variasi antara karakter akhir string kunci dan string yang dicocokkan, maka setiap karakter dalam potongan string yang dicocokkan diperiksa satu demi satu. Langkah ini bertujuan untuk mengidentifikasi apakah terdapat karakter dalam potongan string tersebut yang identik dengan karakter dalam string kunci.

Jika ada kemiripan, maka string kunci akan diatur ulang sedemikian rupa sehingga memposisikan karakter yang sama secara seimbang, kemudian proses

pencocokan karakter terakhir string kunci dijalankan kembali. Sementara itu, jika tidak ada karakter yang cocok, maka semua karakter string kunci akan di-shift ke arah kanan sepanjang  $m$  karakter, di mana  $m$  adalah jumlah karakter dalam string kunci. [8].

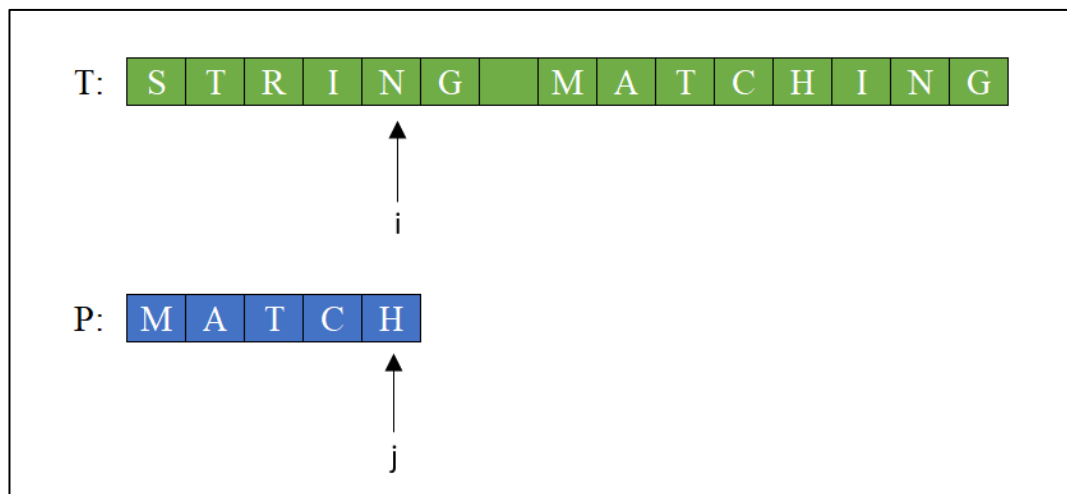
#### A. Mekanisme umum algoritma Boyer Moore

Dalam penggunaan algoritma boyer moore secara umum terdiri dari dua teknik yang harus dilakukan yaitu:

##### 1. The looking-glass technique

Teknik cermin (looking glass technique) diaplikasikan untuk menguji kesesuaian pola  $P$  terhadap teks  $T$ , dengan memulai pemeriksaan dari index paling akhir pada  $P$ . Proses inspeksi terhadap  $T$  tetap berasal dari permulaan, sehingga dalam situasi ini, index  $I$  akan dimulai pada nilai  $m - 1$ .

Contoh :



Gambar 2. 1 The looking-glass technique

Dalam gambar diatas index  $i$  dimulai dari karakter **N** di Teks dan index  $j$  dimulai dari karakter **H** di Pattern.

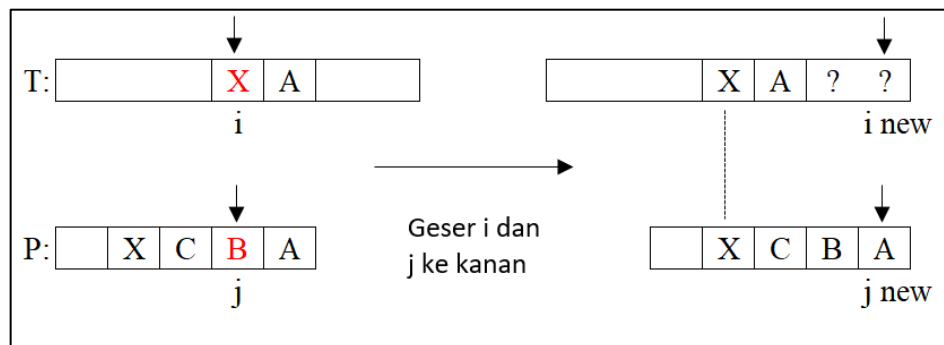


## 2. The character-jump technique

Character jump technique merupakan Teknik Ketika terjadi mismatch antara ( $P[j] \neq T[i]$  dengan pada saat itu  $T[i] = x$ ) maka algoritma ini akan melakukan lompatan atau pergeseran pada posisi tertentu. Untuk menentukan posisi lompatan ini terdapat 3 kasus yang mungkin terjadi yaitu diantaranya yaitu:

### a) Kasus 1

Dalam kasus ini mismatch terjadi pada  $T[i]$  dan  $P[j]$  dan karakter apada  $T[i]$  yaitu x, kemudian jika terdapat karakter x di P dengan posisi index yang lebih kecil dari j, kemudian geser P ke kanan agar posisi x di  $T[i]$  sejajar dengan posisi kemunculan terakhir/Last Occurrence(LO) x di P.

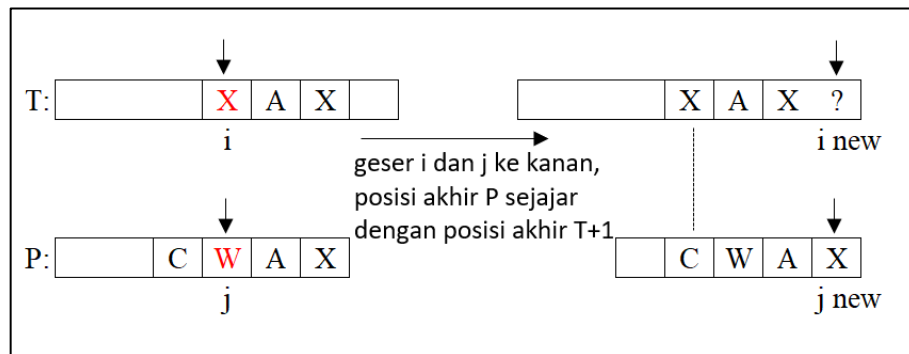


Gambar 2. 2 Kasus 1 - The character-jump technique

Pada pemeriksaan selanjutnya, index j selalu dimulai pada posisi index terakhir (m-1) yang di geser adalah posisi index i dengan nilai i yang baru  $i = i + (m-1) - LO = i+m-(lo+1)$ . Pada contoh gambar diatas i yang baru adalah  $= i + (4-1)-0=i+3$ .

### b) Kasus 2

Dalam kasus ini mismatch terjadi pada  $T[i]$  dan  $P[j]$  dan karakter apada  $T[i]$  adalah x, kemudian jika terdapat karakter x di P tapi pada posisi index yang lebih besar daripada j. geser P satu karakter ke kanan agar posisi index terakhir P sejajar dengan posisi akhir T sebelumnya + 1.

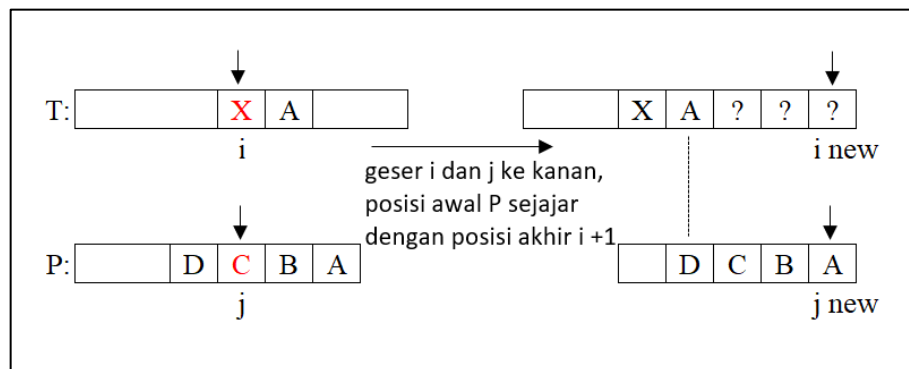


Gambar 2. 3 Kasus 2 - The character-jump technique

Pemeriksaan index berikutnya tetap dimulai dari Pattern  $j = m-1$  kemudian yang digesera adalah nilai i yang baru adalah  $i = i + m - j$ , pada kasus dalam gambar diatas i yang baru adalah  $i = i + 4 - 1 = i + 3$ .

### c) Kasus 3

Pada kasus ini mismatch terjadi pada  $T[i]$  dan  $P[j]$  dan karakter apada  $T[i]$  adalah x, kemudian jika tidak ditemukan karakter x pada P maka geser P agar posisi index pertama P sejajar dengan index  $i + 1$ .



Gambar 2. 4 Kasus 3 - The character-jump technique

Pemeriksaan index berikutnya tetap dimulai dari index terakhir Pattern  $j = m-1$ , kemudian yang digeser adalah nilai i yang baru adalah  $i = i + m$ . Pada kasus ini i yang baru adalah  $i = i + 4$ .

## B. Fungsi Last Occurrence

Fungsi last occurrence digunakan sebagai pre-procesing dalam algoritma ini. Ketiga kasus *The character Jump* memerlukan informasi mengenai di mana karakter pada Teks yang mismatch tersebut kemunculan terakhir karakternya pada Pattern.

Informasi ini dapat diproses saat Pattern sudah diketahui, fungsi ini bertugas untuk menentukan posisi kemunculan terakhir semua karakter pada Teks (T) di dalam Pattern (P). Jika karakter Teks tidak pernah muncul di Pattern maka nilai nya adalah -1. Dibawah merupakan contoh dari fungsi Last occurrence  $L(x)$ .

Variasi karakter pada  $T:A=\{a,b,c,d\}$

P	a	b	a	c	a	b
	0	1	2	3	4	5

Gambar 2. 5 Variasi karakter Boyer moore

$L(a) = 4 \rightarrow$  Last occurrence karakter a pada P ada di index 4

$L(b) = 3 \rightarrow$  Last occurrence karakter b pada P ada di index 3

$L(c) = 5 \rightarrow$  Last occurrence karakter c pada P ada di index 5

$L(d) = -1 \rightarrow$  Last occurrence karakter d tidak muncul

Semua parameter dari Fungsi  $L(x)$  adalah semua karakter pada T. Kemudian Semua nilai disimpan dalam tabel atau larik

x	a	b	c	d
$L(x)$	4	5	3	-1

Gambar 2. 6 Last Occurrence

## 2.7. Algoritma Rabin Karp

Algoritma Rabin-Karp mewakili teknik lanjutan dalam domain pencocokan string. Dikembangkan oleh Michael O. Rabin dan Richard M. Karp, algoritma ini memberikan inovasi substansial dalam domain pencocokan string melalui pendekatan berfokus pada hashing.

Hashing dalam konsep Rabin-Karp sangat berbeda dengan strategi hashing tradisional. Hashing di sini diaplikasikan untuk masing-masing substring dalam teks yang dicocokkan. Teknik berfokus pada pembuatan 'nilai hash' yang khusus dari string kunci dan substring yang dicocokkan dalam teks.

Eksekusi algoritma Rabin-Karp melibatkan perbandingan nilai hash string kunci dengan nilai hash dari masing-masing substring dalam teks. Apabila terdapat kecocokan nilai hash, algoritma akan melanjutkan ke tahap selanjutnya yaitu pencocokan karakter demi karakter.

Keunggulan utama dari algoritma Rabin-Karp adalah efisiensi operasional. Algoritma ini secara kasar mencapai kompleksitas waktu  $O(n+m)$  dalam kasus rata-rata, di mana  $n$  dan  $m$  merujuk pada panjang teks dan string kunci secara berturut-turut. Namun, algoritma Rabin-Karp juga memiliki keterbatasan, yakni dalam kasus terburuk, algoritma ini dapat memiliki kompleksitas waktu sebesar  $O(nm)$  [9].

## 2.8. SQL Query Like

*SQL Query Like* adalah sebuah pernyataan (statement) yang digunakan dalam bahasa SQL untuk melakukan pencarian data yang cocok dengan pola tertentu pada sebuah tabel atau database. Dalam *SQL Query Like*, kita dapat menggunakan wildcard characters atau karakter joker seperti % (untuk merepresentasikan nol atau lebih karakter) dan \_ (untuk merepresentasikan satu karakter) untuk mencari data yang cocok dengan pola yang diinginkan [10].

Contohnya, kita dapat menggunakan pernyataan *SQL Query Like* untuk mencari semua data yang memiliki kata "apple" pada nama buah, seperti:

```
SELECT * FROM fruits WHERE name LIKE '%apple%';
```

Pernyataan ini akan mengembalikan semua data pada tabel "fruits" yang memiliki kata "apple" pada kolom "name".

## **2.9. UML (Unified Modelling Language)**

Untuk merancang sistem yang berorientasi objek, dibutuhkan suatu metode pemodelan secara visual, metode ini dinamakan Bahasa Pemodelan Terpadu (UML). UML adalah standar bahasa untuk pendokumentasian, perancangan, dan visualisasi.

Tujuan dibuatnya UML adalah agar mempermudah dalam mengembangkan suatu perangkat lunak. Selain itu UML juga diharapkan dapat mempermudah semua kebutuhan pengguna dengan tepat, lengkap dan efektif. merancang dan juga memodelkan sistem secara matang akan menghasilkan sistem yang baik. UML penting sekali bagi para pengembang sistem karena UML akan menjadi jembatan untuk menerjemahkan antara pengembang sistem dengan pengguna [11] Beberapa macam UML yang ada antara lain:

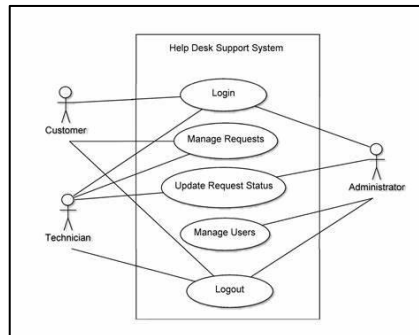
### **A. *Use Case Diagram***

Use case diagram adalah diagram UML (Unified Modeling Language) yang digunakan dalam rekayasa perangkat lunak untuk menggambarkan interaksi antara sistem atau aplikasi dengan pengguna atau aktor-aktor lain yang terkait. Use case diagram dapat digunakan untuk menggambarkan berbagai skenario penggunaan (use case) dari sebuah sistem atau aplikasi, sehingga dapat membantu dalam memahami kebutuhan fungsional dari sistem atau aplikasi tersebut.

Dalam use case diagram, setiap use case direpresentasikan sebagai oval dan setiap aktor direpresentasikan sebagai kotak. Garis yang menghubungkan use case dan aktor menunjukkan interaksi antara mereka. Use case diagram dapat digunakan untuk mengidentifikasi dan memvisualisasikan aktor-aktor yang terlibat dalam sebuah sistem atau aplikasi, serta skenario-skenario penggunaan yang dapat terjadi.

Use case diagram juga dapat digunakan untuk mengidentifikasi dan memperjelas persyaratan sistem atau aplikasi dengan menggambarkan interaksi antara sistem atau aplikasi dan pengguna atau aktor-aktor lain yang terkait. Use case diagram sering digunakan sebagai langkah awal dalam proses pengembangan

perangkat lunak, dan dapat digunakan sebagai alat komunikasi yang efektif antara pengembang perangkat lunak, klien, dan pemangku kepentingan lainnya. [12].



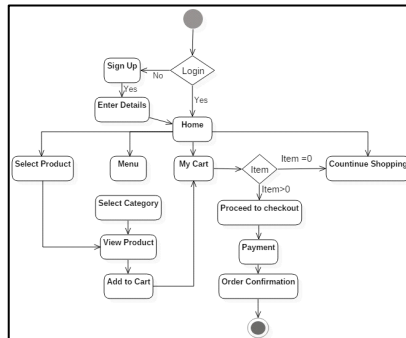
Gambar 1. 2 Contoh Use Case Diagram

## B. Activity Diagram

Activity diagram adalah jenis diagram UML (Unified Modeling Language) yang digunakan untuk menggambarkan alur aktivitas atau tindakan dalam sebuah proses bisnis atau sistem. Diagram ini memperlihatkan urutan tindakan yang terjadi dalam sebuah proses atau sistem dan kondisi yang terjadi pada setiap langkah. Activity diagram sangat berguna dalam menggambarkan aliran kerja sistem, urutan aktivitas yang terjadi, serta pengambilan keputusan dalam suatu proses.

Pada activity diagram, aktivitas direpresentasikan oleh persegi panjang dengan nama aktivitas di dalamnya. Tindakan sederhana dapat direpresentasikan oleh lingkaran kecil di dalam aktivitas. Keputusan direpresentasikan oleh berlian dan memiliki beberapa jalur keluar. Sedangkan fork digunakan untuk merepresentasikan percabangan dalam sebuah proses.

Activity diagram sangat berguna dalam menggambarkan aktivitas dan tindakan dalam sebuah proses, menggambarkan kondisi yang mungkin terjadi, dan menggambarkan percabangan dalam suatu proses. Diagram ini dapat membantu pengembang perangkat lunak untuk memahami alur kerja dalam sistem, mengidentifikasi masalah dan perbaikan dalam alur kerja, serta merancang sistem yang lebih efisien dan efektif. Activity diagram sering digunakan dalam analisis dan perancangan sistem, dan dapat digunakan sebagai alat komunikasi yang efektif antara pengembang perangkat lunak, klien, dan pemangku kepentingan lainnya.



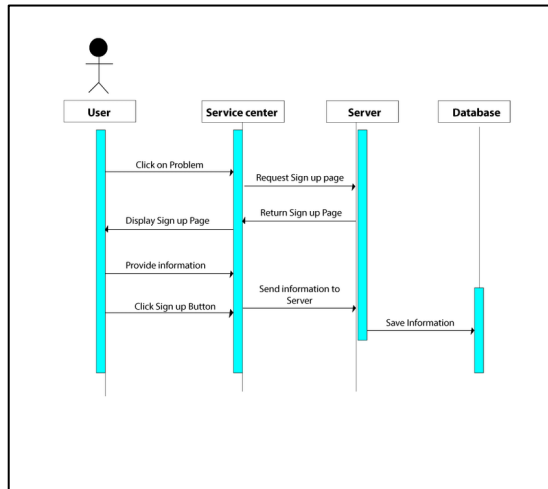
Gambar 2. 7 Contoh Activity Diagram

### C. *Sequence Diagram*

Sequence diagram adalah jenis diagram UML (Unified Modeling Language) yang digunakan untuk menggambarkan interaksi antara objek dalam sebuah sistem. Diagram ini memperlihatkan urutan pemanggilan metode antara objek, serta urutan pesan yang dikirimkan antara objek dalam sebuah skenario. Sequence diagram sangat berguna dalam memodelkan interaksi antara objek dalam sistem dan memperlihatkan urutan pemanggilan metode yang terjadi.

Pada sequence diagram, objek direpresentasikan oleh sebuah kotak vertikal, dengan nama objek di bagian atas. Urutan pemanggilan metode antara objek direpresentasikan oleh panah horizontal, dengan tanda kurung kurawal di atasnya untuk menunjukkan nama metode. Pesan yang dikirim antara objek direpresentasikan oleh panah vertikal dengan tanda kurung kurawal di atasnya untuk menunjukkan nama pesan.

Sequence diagram berguna dalam menggambarkan interaksi antara objek dalam sebuah sistem, serta memperlihatkan urutan pemanggilan metode yang terjadi. Diagram ini dapat membantu pengembang perangkat lunak untuk memahami bagaimana objek dalam sistem saling berinteraksi, mengidentifikasi masalah dan perbaikan dalam interaksi antara objek, serta merancang sistem yang lebih efisien dan efektif. Sequence diagram sering digunakan dalam analisis dan perancangan sistem, dan dapat digunakan sebagai alat komunikasi yang efektif antara pengembang perangkat lunak, klien, dan pemangku kepentingan lainnya. [12].

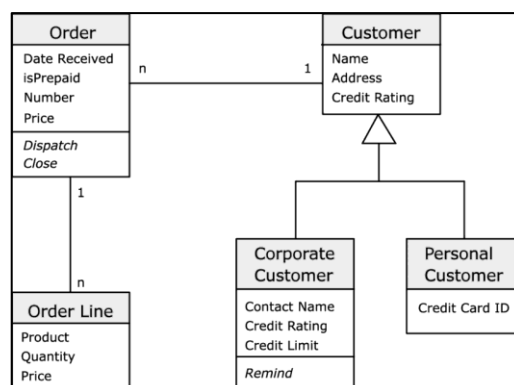


Gambar 2. 8 Contoh Sequence Diagram

#### D. *Class Diagram*

Class diagram adalah jenis diagram UML (Unified Modeling Language) yang digunakan untuk memodelkan struktur kelas atau objek dalam sebuah sistem. Diagram ini memperlihatkan kelas-kelas dalam sistem, serta hubungan antara kelas-kelas tersebut. Class diagram sangat berguna dalam memodelkan struktur kelas dalam sistem dan memperlihatkan hubungan antara kelas-kelas tersebut.

Pada class diagram, kelas direpresentasikan oleh sebuah persegi panjang, dengan nama kelas di dalamnya. Atribut kelas direpresentasikan oleh nama atribut di dalam kelas, sedangkan metode kelas direpresentasikan oleh nama metode di dalam kelas. Hubungan antara kelas direpresentasikan oleh panah yang mengarah dari kelas yang satu ke kelas yang lain [12].



Gambar 2. 9 Contoh Class Diagram



## **2.10.PHP (PHP: Hypertext Preprocessor)**

PHP adalah singkatan dari "Hypertext Preprocessor". Sejarahnya bermula pada tahun 1994, ketika Rasmus Lerdorf menciptakannya untuk mengelola statistik pribadi pada laman web-nya. Seiring perkembangan, PHP telah menjelma menjadi bahasa skrip server-side yang populer untuk pengembangan web.

PHP adalah bahasa pemrograman berbasis skrip yang ditafsirkan pada saat runtime, berarti kode PHP dievaluasi dan dijalankan di server, bukan di sisi klien. Salah satu keunikan PHP adalah bahwa kode dapat ditanamkan langsung ke dalam HTML, yang memungkinkan integrasi yang mulus antara back-end dan front-end pada aplikasi web.

Selanjutnya, PHP mendukung pemrograman berorientasi objek (OOP), memungkinkan pengembang untuk mengorganisir kode dalam bentuk kelas, objek, dan metode. Fitur ini membantu meningkatkan modularitas dan keterbacaan kode, serta memfasilitasi pemrograman skala besar.

Hingga sekarang, PHP tetap menjadi salah satu bahasa pemrograman dominan dalam bidang pengembangan web, yang dikagumi karena fleksibilitas dan kemudahannya dalam mengintegrasikan dengan HTML. Meski demikian, seperti bahasa pemrograman lainnya, PHP memiliki kekuatan dan kelemahan tersendiri, dan keefektifan penggunaannya bergantung pada konteks spesifik proyek atau aplikasi.

## **2.11. Basis Data (MYSQL)**

MySQL dikembangkan oleh Swedish MySQL AB mereka, yang kemudian diakuisisi oleh Oracle. Pasca akuisisi, beberapa pengembang dari MySQL memutuskan untuk mencabangkan MySQL dan menciptakan MariaDB, melahirkan server basis data relasional alternatif yang mengutamakan komunitas dan pengembangan dengan sumber terbuka.

MariaDB adalah sebuah sistem manajemen basis data relasional (RDBMS) yang berfungsi untuk menyimpan, menyalurkan, dan mengelola data dalam struktur tabel. MariaDB bertujuan untuk menjaga kompatibilitas penurunan dengan

MySQL, memastikan bahwa aplikasi dan perangkat lunak yang ditulis untuk MySQL tetap berjalan dengan baik di MariaDB.

Pada dasarnya, basis data MariaDB menyimpan data dalam tabel relasional, yang diorganisir dalam kolom dan baris. Karakteristik utama yang membedakan MariaDB dari MySQL termasuk paket tambahan berorientasi keamanan, peningkatan kemampuan baru, serta teknologi penyimpanan yang diperkenalkan oleh komunitas pengembang.

Salah satu kekuatan utama MariaDB terletak pada skalabilitasnya. MariaDB memiliki kemampuan untuk menangani basis data dalam berbagai ukuran, membentuk jalan yang layak diteroka bagi organisasi yang menangani pengelolaan data dalam skala besar.

Meskipun MariaDB merupakan turunan dari MySQL, namun fitur dan optimalisasi baru yang ditawarkan dapat mempengaruhi pilihan antara kedua sistem tersebut. Oleh karenanya, MariaDB telah meraih kepercayaan dan adopsi luas dalam industri, berkat pendekatannya terhadap kompatibilitas, sumber terbuka, dan partisipasi komunitas.

## 2.12. Penelitian Terdahulu

Pada penelitian ini menggunakan beberapa referensi dari penelitian terdahulu yang dapat dilihat pada tabel berikut:

Tabel 2. 1 Penelitian Terdahulu

<b>Judul (Tahun) Penulis</b>	<b>Tujuan Penelitian</b>	<b>Kesimpulan</b>
ANALISIS STRING MATCHING PADA JUDUL SKRIPSI DENGAN 2017 Wistiani Astuti	Pada penelitian ini menggunakan Algoritma Knuth-Morris- Pratt (KMP) untuk menganalisis bagaimana proses pencocokan string yang	Berdasarkan hasil penelitian yang dilakukan maka dapat diambil kesimpulan bahwa hasil analisis algoritma knuth-morris pratt (KMP) dapat

<b>Judul (Tahun) Penulis</b>	<b>Tujuan Penelitian</b>	<b>Kesimpulan</b>
ALGORITMA KNUTH- MORRIS PRATT (KMP)	dihasilkan dan membandingkan sejauh mana nilai kemiripan dari beberapa judul yang sama dan serupa sehingga dapat memberikan suatu informasi yang efektif bagi mahasiswa.	diketahui kualitas pencocokan string pada judul skripsi atau topik penelitian yang diajukan mahasiswa sehingga setiap mahasiswa bisa mengetahui judul- judul skripsi yang sudah ada dan yang belum ada pada sistem yang dibangun.
Perbandingan Algoritma Boyer Moore dan Algoritma Rabin- Karp Terhadap Kode Pos Wilayah Aceh (2021) Dwi Riesky Chandra Wiradhika <sup>1</sup> , Yudha Nurdin <sup>2</sup> , Fardian <sup>3</sup>	Penelitian ini merancang sebuah aplikasi pencarian berbasis Android untuk kode pos wilayah Aceh, menggunakan proses string matching dengan algoritma Boyer Moore dan Rabin- Karp. String matching bertujuan mencari semua kemunculan suatu pola dalam string yang lebih panjang.	Penelitian ini dirancang untuk mengembangkan aplikasi Android menggunakan Android Studio, menerapkan algoritma Boyer Moore dan Rabin-Karp untuk pencarian string. Melalui tiga putaran uji, aplikasi menghasilkan Real Running Time rata-rata 5,53 ms dengan Boyer Moore dan 6,96 ms dengan Rabin-Karp. Hasilnya menunjukkan bahwa algoritma Boyer Moore 21,33% lebih cepat

<b>Judul (Tahun) Penulis</b>	<b>Tujuan Penelitian</b>	<b>Kesimpulan</b>
		dalam mencari pola dibandingkan Rabin-Karp
Studi Perbandingan Implementasi Algoritma Boyer Moore, Turbo Boyer Moore, dan Tuned Boyer Moore dalam Pencarian String (2013) Vina Sagita, Maria Irmina Prasetiyowati	Penelitian ini berkaitan dengan pencarian string, proses umum yang digunakan komputer dalam pengolahan teks sebagai bentuk utama penyimpanan data. Fokus penelitian adalah evaluasi performa beberapa algoritma pencarian string, yakni Boyer Moore, Turbo Boyer Moore, dan Tuned Boyer Moore, terutama dalam konteks waktu pencarian.	Dari hasil penelitian yang didapat, bisa disimpulkan bahwa algoritma Boyer Moore memiliki waktu pencarian tercepat dari tiga varian Boyer Moore yang telah ditentukan. Algoritma Turbo Boyer Moore merupakan algoritma tercepat kedua dan yang paling lambat adalah Tuned Boyer Moore

## BAB III

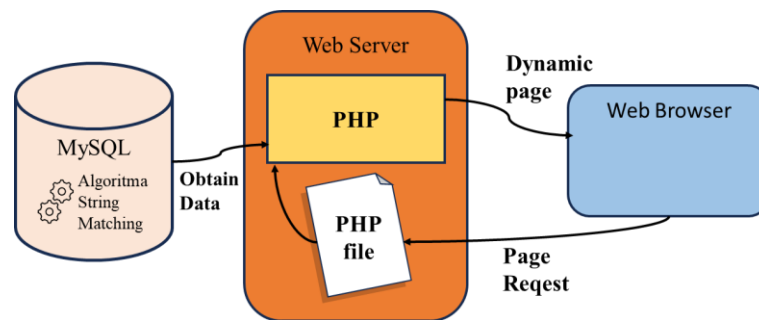
### ANALISIS DAN PERANCANGAN SISTEM

#### 3.1. Sistem Berjalan

Salah satu metode umum yang digunakan dalam pencarian data adalah menggunakan query dengan operator LIKE. Operator LIKE memungkinkan pencarian data berdasarkan pola string tertentu dalam kolom yang ditentukan.

Dalam SQL ketika kita ingin mencari data yang cocok dengan pola tertentu, kita dapat menggunakan pernyataan SELECT dengan klausa WHERE yang mengandung operator LIKE. Operator ini memungkinkan kita untuk mencocokkan pola string dengan data yang ada. Terdapat beberapa algoritma yang bisa digunakan untuk ini mencocokkan pola string yaitu algoritma *Naive String Matching*, *Boyer Moore*, *Knuth Morris Pratt* dan *Rabin-Karp*.

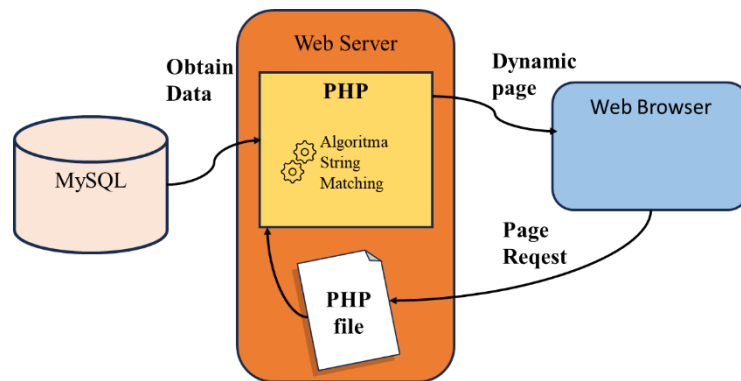
Dalam penelitian ini akan dibuat sistem yang digunakan untuk melakukan pengujian perbandingan string matching dengan menggunakan algoritma *Naive String Matching*, *Boyer Moore*, *Knuth Morris Pratt* *Rabin-Karp* dan *SQL Query Like* untuk melihat performa dari algoritma tersebut.



Gambar 3. 1 Pencarian Data Menggunakan MySQL Dengan PHP

#### 3.2. Sistem yang diusulkan

Sistem yang akan dibuat bertujuan untuk membandingkan performansi berbagai metode string matching yang digunakan dalam pencarian data konsumen dalam basis data. Metode yang akan dibandingkan termasuk *Naive String Matching*, *Boyer Moore*, *Knuth Morris Pratt* *Rabin-Karp*, dan *SQL Query LIKE*. Berikut merupakan flowchart dari sistem yang diusulkan dalam penelitian ini.



Gambar 3. 2 Sistem yang diusulkan.

Dalam gambar diatas bisa dilihat algoritma string matching dilakukan di bahasa pemrograman PHP.

#### A. Data Penelitian

Sumber data yang digunakan dalam penelitian ini diambil dari Kaggle. Kaggle merupakan platform yang menyediakan akses ke ribuan dataset yang dikumpulkan dari berbagai sumber, termasuk sumber publik, organisasi, dan individu yang ingin berbagi data mereka. Dataset yang diperoleh dari Kaggle umumnya dilengkapi dengan deskripsi, atribut, dan metadatanya, yang memberikan pemahaman awal terhadap data yang digunakan.

Data konsumen yang terdiri dari id, nama, email, alamat, nomor telepon dan jenis kelamin dengan jumlah data 250.000 dan dipecah menjadi 25 tabel dengan skenario 10.000 per tabel yaitu 10.000, 20.000, 30.000, 40.000 sampai 250.000. Hal ini bertujuan untuk melihat bagaimana performansi dari algoritma dengan jumlah data yang berbeda. Dibawah merupakan contoh data yang digunakan.

Tabel 3. 1 Daftar beberapa data konsumen yang akan digunakan.

id	Name	email	address	phone	gender
1	Ms. Aliyah Runolfsdottir	evan780@example.com	9195 Schoen Lodge Apt. 690 Bechtelarland, MA 23828	580-739-6844	female
2	Theron Sanford	aidan.rutherford1@example.com	8722 Reichert Ramp Laronton, CT 00628	1-203-689-8215	male
3	Jaunita Brown	daryl.blanda2@example.com	397 Feeney Shores Apt. 368	1.831E+10	female

id	Name	email	address	phone	gender
			Port Daveland, ID 31389-6584		
...	...	...	...	...	...
250 000	Courtney Brakus	hahn.bernardo249999@example.com	63453 Luz Motorway O'Reillyland, SC 79560	1-559-352-4733	female

## B. Data Preprocessing

Dari 5 metode string matching dalam penelitian ini hanya terdapat 3 metode algoritma yang menggunakan data preprocessing yaitu:

### a) Algoritma Knut Morris Pratt

Dalam algoritma ini, terdapat tahap preprocessing yang melibatkan perhitungan Failure Function atau Border Function terlebih dahulu. Fungsi pinggiran ini bertugas untuk mengaudit pola demi menemukan kesesuaian awalan dari pola tersebut dengan pola itu sendiri. Misalnya, jika memiliki pola "acabaca," dapat dijelaskan sebagai berikut:

Tabel 3. 2 Border Function

j	0	1	2	3	4	5	6
P[j]	a	c	a	b	a	c	a
K	-	0	1	2	3	4	5
b(k)	-	0	1	0	1	2	3

### b) Algoritma Boyer More

Fungsi last occurrence sebagai data preprocessing untuk menentukan posisi kemunculan terakhir semua karakter pada Teks (T) di dalam Pattern (P). Contoh apabila kita mempunyai teks "abacaabadcabacabaabb" dan pattern yang ingin dicari "abacab". Pertama kita harus menjalankan fungsi last occurrence dibawah ini.

Variasi karakter pada T: A={a,b,c,d}

P	a	b	a	c	a	b
	0	1	2	3	4	5

Gambar 3. 3 Variasi karakter Boyer moore

$L(a) = 4$  -> Last Occurrence karakter a pada P ada di index 4

$L(b) = 3$  -> Last Occurrence karakter b pada P ada di index 3

$L(c) = 5$  -> Last Occurrence karakter c pada P ada di index 5

$L(d) = -1$  -> Last Occurrence karakter d tidak muncul

Semua parameter dari Fungsi  $L(x)$  adalah semua karakter pada T. Kemudian Semua nilai disimpan dalam tabel atau larik.

x	a	b	c	d
$L(x)$	4	5	3	-1

Gambar 3. 4 Tabel Last Occurence

### c) Algoritma Rabin Karp

Fungsi hash untuk pattern terlebih dahulu sebelum membandingkannya dengan beberapa karakter di Teks. Contoh apabila kita mempunyai teks “ISEPLUTPINUR” dan pattern yang ingin dicari “PIN”. Pertama kita harus menjalankan fungsi hashing untuk pattern yang akan dicari.

Sebagai contoh dari pattern “PIN” dimasukan kedalam fungsi hash dan menghasilkan nilai “a8”. Berikut merupakan Langkah-langkah untuk melakukan algoritma String Matching Rabin Karp.

### C. Metode penelitian

Algoritma yang akan digunakan untuk string matching dalam penelitian ini yaitu Naive String Matching, Boyer Moore, Knuth Morris Pratt, Rabin-Karp dan *SQL Query Like* untuk data yang digunakan di database SQL.



### a) Algoritma Naive String Matching

Algoritma naive string matching memiliki kompleksitas waktu sebesar  $O(m \cdot (n - m))$ , di mana  $m$  adalah panjang pola dan  $n$  adalah panjang teks atau string. Algoritma ini bekerja dengan baik pada string yang relatif pendek dan pola yang singkat.

Berikut adalah pseudocode algoritma naive string matching:

#### NAIVE-STRING-MATCHING (T, P)

```
1. n ← length [T]
2. m ← length [P]
3. for i ← 0 to n - m
4.   do if P [1.....m] = T [i + 1.....i + m]
5.     then return i
```

Pseudocode di atas menjelaskan bagaimana algoritma naive string matching mencari keberadaan sebuah pola  $P$  dalam sebuah string  $T$ . Algoritma ini membandingkan setiap  $m$  karakter dalam  $P$  dengan setiap  $m$  karakter di  $T$ . Jika ada kecocokan, algoritma akan mencetak pesan bahwa pola ditemukan dengan pergeseran (shift) tertentu.

Contoh pencarian menggunakan algoritma *Naive String Matching*:

Teks = PLANINGANDANALYSIS

Pattern = AND

- **Langkah 1:** Indeks  $i = 0$  dan  $j = 0$ , karena terjadi ketidakcocokan maka  $i$  ditambah 1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S
0																	
A	N	D															
0	1	2															

Gambar 3. 5 Langkah 1 contoh algoritma naive string matching

- **Langkah 2:** Pada langkah ini kembali terjadi ketidakcocokan atau *mismatch* maka index i ditambah 1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

1		
A	N	D
0	1	2

Gambar 3. 6 Langkah 2 contoh algoritma naive string matching

- **Langkah 3:** Pada langkah ini terjadi kecocokan maka index j ditambah 1 dan dicocokkan ke depan sampai terjadi *mismatch* di index j ke 2 kemudian index I ditambah 1 dan j jadi 0 kembali.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

2   3   4		
A	N	D
0	1	2

Gambar 3. 7 Langkah 3 contoh algoritma naive string matching

- **Langkah 4:** Pada Langkah ini terjadi *mismatch* maka index i ditambah 1.

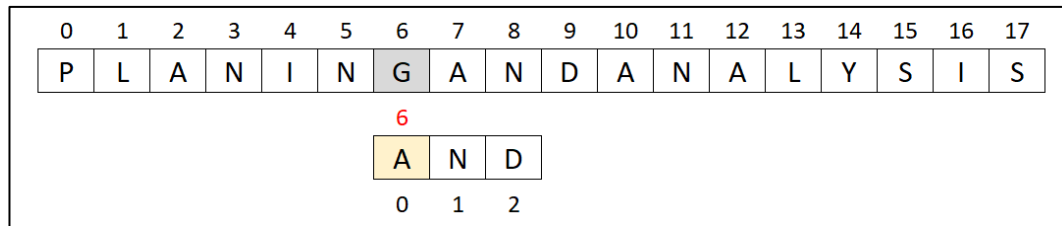
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

5		
A	N	D

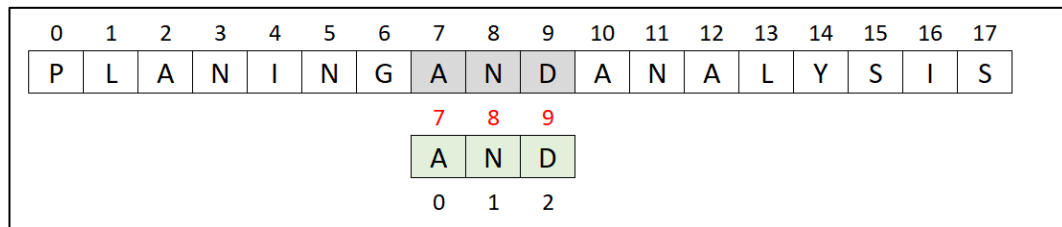
Gambar 3. 8 Langkah 4 contoh algoritma naive string matching

- **Langkah 5:** Pada Langkah ini terjadi *mismatch* maka index i ditambah 1.



Gambar 3. 9 Langkah 5 contoh algoritma naive string matching

- **Langkah 6:** Pada Langkah ini terjadi *match* kemudian di periksa sampai panjang m, kemudian semuanya match dan diambil index ke 7 sebagai hasil.



Gambar 3. 10 Langkah 6 contoh algoritma naive string matching

## b) Algoritma Knuth Morris Pratt

Dalam algoritma KMP, terdapat suatu fungsi yang disebut *Border Function* atau Failure Function KMP, yang biasa dikenal sebagai fungsi pinggiran KMP. Fungsi pinggiran ini bertugas untuk mengaudit pola dengan tujuan menemukan kesesuaian awalan dari pola itu sendiri dengan pola yang sama.  $b(k)$  melambangkan fungsi pinggiran dimana ukuran maksimum dari awalan pola  $P[0..k]$  yang juga merupakan akhiran dari pola  $P[1..k]$ , di mana  $k$  menjadi posisi sebelum insiden ketidaksesuaian yaitu  $j-1$ , di mana  $j$  adalah posisi di mana terjadi ketidaksesuaian.

Contoh apabila kita mempunyai sebuah *pattern* “acabaca”, dapat didefinisikan sebagai berikut:

Tabel 3 1 Tabel pattern “acabaca” algoritma KMP

j	0	1	2	3	4	5	6
P[j]	a	c	a	b	a	c	a
K	-	0	1	2	3	4	5

Perhitungan fungsi pinggirian dimulai dari  $j=0$ , yaitu ketika  $k$  tidak dapat didefinisikan maka nilai fungsi pinggirian juga tidak dapat didefinisikan.

Selanjutnya fungsi pinggirian pada  $j=1$ , kemudian  $k = 0$ . Prefix dari *pattern* adalah  $P[0..0]$  yaitu “a”. Suffix dari *pattern* adalah  $[1..1]$  yaitu “b”. Karena *suffix* tidak sama dengan *prefix*, maka nilai fungsi pinggirian adalah 0 atau  $b(0) = 0$ .

Selanjutnya fungsi pinggirian pada  $j=2$ , kemudian  $k=1$ . Prefix dari *pattern* adalah  $P[0..1]$  yaitu “a”, dan “ac”. Suffix dari *pattern* adalah  $P[1..2]$  yaitu “a”, dan “ca”. Karena terdapat kesamaan pada *prefix* dan *suffix* yang berpanjang 1 yaitu “a”, maka nilai fungsi pinggirian adalah 1 atau  $b(1) = 1$ .

Selanjutnya fungsi pinggirian pada  $j=3$ , kemudian  $k=2$ . Prefix dari *pattern* adalah  $P[0..2]$  yaitu “a”, “ac”, dan “aca”. Suffix dari *pattern* adalah  $P[1..3]$  yaitu “b”, “ba”, dan “cab”. Karena *suffix* tidak terdapat kesamaan *prefix*, maka nilai fungsi pinggirian adalah 2 atau  $b(2) = 0$ .

Selanjutnya fungsi pinggirian pada  $j=4$ , kemudian  $k=3$ . Prefix dari *pattern* adalah  $P[0..4]$  yaitu “a”, “ac”, “aca” dan “acab”. Suffix dari *pattern* adalah  $P[1..3]$  yaitu “a”, “ba”, “aba” dan “caba”. Karena terdapat kesamaan pada *prefix* dan *suffix* yang berpanjang 1 yaitu “a”, maka nilai fungsi pinggirian adalah 4 atau  $b(4) = 1$ .

Selanjutnya fungsi pinggirian pada  $j=5$ , kemudian  $k=4$ . Prefix dari *pattern* adalah  $P[0..5]$  yaitu “a”, “ac”, “aca”, “acab” dan “acaba”. Suffix dari *pattern* adalah  $P[1..4]$  yaitu “c”, “ac”, “bac”, “abac” dan “cabac”. Karena terdapat kesamaan pada *prefix* dan *suffix* yang berpanjang 2 yaitu “ac”, maka nilai fungsi pinggirian adalah 5 atau  $b(5) = 2$ .

Selanjutnya fungsi pinggirian pada  $j=6$ , kemudian  $k=5$ . Prefix dari *pattern* adalah  $P[0..6]$  yaitu “a”, “ac”, “aca”, “acab”, “acaba” dan “acabac”. Suffix dari *pattern* adalah  $P[1..5]$  yaitu “a”, “ca”, “aca”, “baca”, “abaca” dan “cabaca”. Karena terdapat kesamaan pada *prefix* dan *suffix* yang berpanjang 3 yaitu “aca”, maka nilai fungsi pinggirian adalah 6 atau  $b(6) = 3$ .

Dengan itu, bisa didefinisikan fungsi pembatas dari string “acabaca” adalah sebagai berikut:

Tabel 3 2 Tabel fungsi pattern “acabaca” algoritma KMP

j	0	1	2	3	4	5	6
P[j]	a	c	a	b	a	c	a
K	-	0	1	2	3	4	5
b(k)	-	0	1	0	1	2	3

Algoritma KMP mengimplementasikan pergeseran berdasarkan fungsi pinggiran yang telah ditetapkan pada pola. Kondisi-kondisi yang diperlukan untuk hal ini meliputi:

1. Ketika karakter P[j] terjadi ketidakcocokan yaitu  $P[j] \neq T[i]$ , dan  $k=j-1$ , maka nilai j menjadi b(k)
2. Ketika karakter P[j] terjadi kecocokan yaitu  $P[j]=T[i]$ , maka nilai i menjadi i+1, dan nilai j menjadi j+1

Contoh pergeseran pada algoritma Knuth Morris Pratt dengan pattern P “acabaca” dan teks T “abacaabacabacababa” sebagai berikut:

Tabel 3 3 Tabel Teks "abacaabacabacababa" Algoritma KMP

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T[j]	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	b	a

- **Langkah 1:** Pencocokan string dimulai dari  $i=0$  dan  $j=0$ , dimana ketika  $i=0$  dan  $j=0$  terjadi kecocokan, sehingga nilai i dan j masing-masing bertambah 1. Ketika  $i=1$  dan  $j=1$  terjadi kecocokan kembali sehingga i dan j masing-masing bertambah 1. Kemudian terjadi ketidakcocokan pada  $i=1$  dan  $j=1$  dimana  $T[i] = \text{“b”}$  dan  $P[j] = \text{“c”}$ . Karena terjadi ketidakcocokan maka pattern harus digeser sesuai dengan tabel fungsi pembatas ketika  $j=1$  yaitu 0, sehingga mengulangi pengecekan pada  $i=1$  dan  $j=0$ .

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T[j]	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	b	a

	1	2					
a	c	a	b	a	c	a	
0	1	2	3	4	5	6	

Gambar 3. 11 Langkah 1 contoh algoritma KMP

- **Langkah 2:** Kemudian pada pengecekan selanjutnya terjadi ketidakcocokan dimana  $T[i] = "b"$  dan  $P[j] = "a"$ . karena  $j=0$  maka nilai  $i$  menjadi  $i+1$  dan nilai  $j$  menjadi  $j=0$  karena pada fungsi pembatas jika  $j=0$  nilai nya tidak terdefinisikan.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T[j]	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	b	a

	3						
a	c	a	b	a	c	a	
0	1	2	3	4	5	6	

Gambar 3. 12 Langkah 2 contoh algoritma KMP

- **Langkah 3:** Pengecekan selanjutnya terjadi Ketika  $i=5$  dan  $j=3$  dimana terjadi ketidakcocokan sehingga terjadi pergeseran Kembali. Fungsi pembatas Ketika  $j=3$  yaitu  $b(2)=0$ .

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T[j]	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	b	a

	4	5	6	7			
a	c	a	b	a	c	a	
0	1	2	3	4	5	6	

Gambar 3. 13 Langkah 3 contoh algoritma KMP

- **Langkah 4:** Pengecekan selanjutnya terjadi Ketika  $i=6$  dan  $j=1$  dimana terjadi ketidakcocokan sehingga terjadi pergeseran Kembali. Fungsi pembatas Ketika  $j=1$  yaitu  $b(0)=0$ .

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T[j]	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	b	a

	8	9					
	a	c	a	b	a	c	a
	0	1	2	3	4	5	6

Gambar 3. 14 Langkah 4 contoh algoritma KMP

- **Langkah 5:** Pengecekan selanjutnya terjadi Ketika  $i=6$  dan  $j=0$  dimana terjadi ketidakcocokan sehingga terjadi pergeseran Kembali. ketika  $j=0$  maka  $i+1$  karena  $k(-1)$  tidak terdefinisi.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T[j]	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	b	a

	10						
	a	c	a	b	a	c	a
	0	1	2	3	4	5	6

Gambar 3. 15 Langkah 5 contoh algoritma KMP

- **Langkah 6:** Pada pengecekan selanjutnya terjadi kecocokan semua pattern. sehingga terdapat sebuah pattern “acabaca” pada string/teks “abacaabacabacababa”.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T[j]	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	b	a

	11	12	13	14	15	16	17
	a	c	a	b	a	c	a
	0	1	2	3	4	5	6

Gambar 3. 16 Langkah 6 contoh algoritma KMP

### c) Algoritma Boyer Moore

Dalam algoritma ini terdapat fungsi last occurrence untuk menentukan kemunculan terakhir Teks di Pattern, fungsi ini merupakan pre-procesing dari algoritma Boyer Moore. Kemudian secara umum mekanisme algoritma boyer moore menggunakan Teknik the looking glass technique dan the character jump.

Contoh apabila kita mempunyai teks “abacaabadcabacabaabb” dan pattern yang ingin dicari “abacab”. Pertama kita harus menjalankan fungsi last occurrence dibawah ini.

Variasi karakter pada  $T:A=\{a,b,c,d\}$

P	a	b	a	c	a	b
	0	1	2	3	4	5

Gambar 3. 17 Variasi karakter Boyer moore

$L(a) = 4$  -> Last Occurrence karakter a pada P ada di index 4

$L(b) = 3$  -> Last Occurrence karakter b pada P ada di index 3

$L(c) = 5$  -> Last Occurrence karakter c pada P ada di index 5

$L(d) = -1$  -> Last Occurrence karakter d tidak muncul

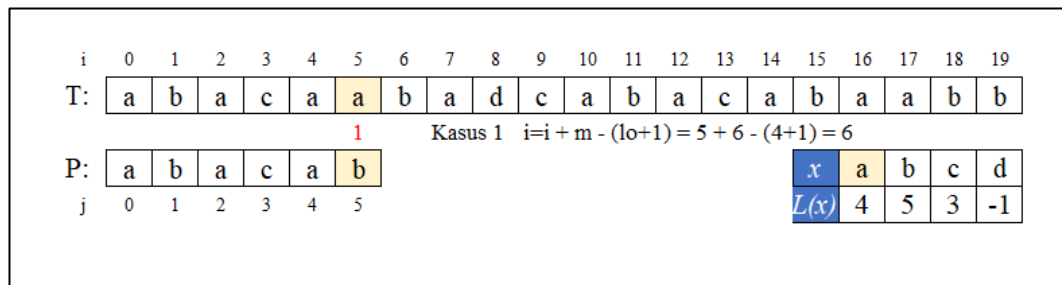
Semua parameter dari Fungsi  $L(x)$  adalah semua karakter pada T. Kemudian Semua nilai disimpan dalam tabel atau larik.

x	a	b	c	d
$L(x)$	4	5	3	-1

Gambar 3. 18 Last Occurrence

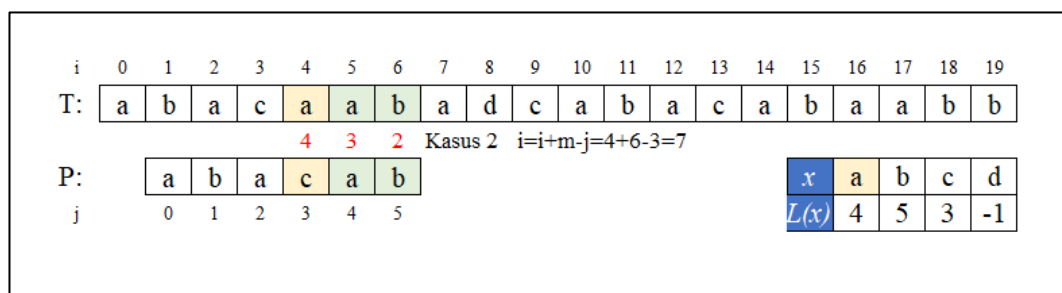
Setelah diketahui last occurrence dapat dilakukan pemeriksaan teks seperti di bawah ini.





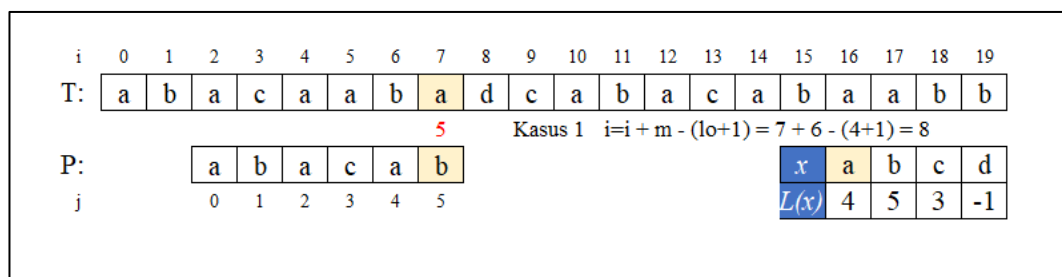
Gambar 3. 19 Langkah 1 Contoh Algoritma Boyer Moore

Pada pemeriksaan pertama terjadi mismatch yaitu di Teks index ke 5 dengan karakter a dan jika dilihat dari tabel LO maka nilai LO dari a adalah 4 yang berarti masuk ke kasus 1 yaitu Ketika lo lebih kecil daripada index i. kemudian geser index i dengan rumus  $i = i + m - (lo + 1)$ , hasil yang didapat dari rumus tersebut adalah index 6.



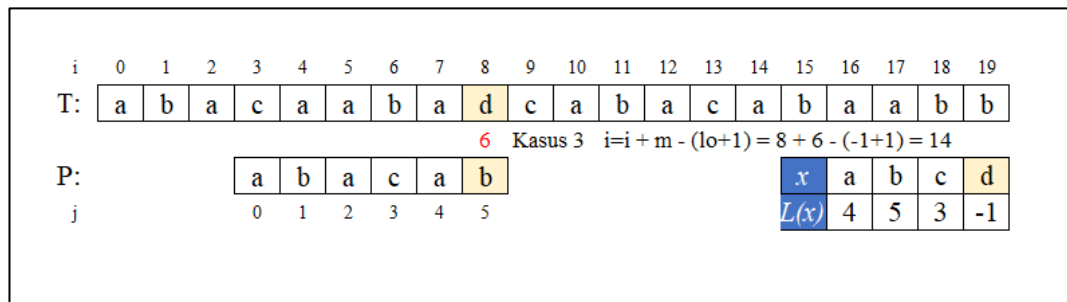
Gambar 3. 20 Langkah 2 Contoh Algoritma Boyer Moore

Pada pemeriksaan selanjutnya terjadi mismatch di Langkah ke 4 index ke 4 dengan karakter teks a yang nilai lo nya 4 ketika di cek ternyata nilai lo 4 sedangkan index j kurang dari index I maka ini termasuk kedalam kasus 2 yaitu dengan rumus  $i = i + m - j$  yang hasilnya didapat index 7.



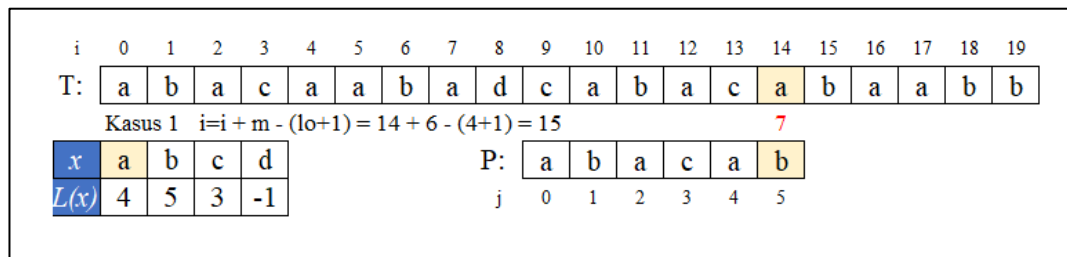
Gambar 3. 21 Langkah 3 Contoh Algoritma Boyer Moore

Pada pemeriksaan selanjutnya terjadi mismatch di Langkah Ke 5 yaitu karakter a pada index ke 5 j kemudian nilai lo dari 4 kemudian karena lo kurang dari index 5 maka termasuk ke dalam kasus 1 dan hasil dari perhitungan kasus 1 adalah index 8.



Gambar 3. 22 Langkah 4 Contoh Algoritma Boyer Moore

Pada pemeriksaan selanjutnya terjadi mismatch di index 8 yaitu karakter d kemudian di pattern tidak ada karakter d maka termasuk kasus 3 yaitu loncat dengan nilai m (panjang pattern) dengan menggunakan rumus kasus 3 maka hasilnya adalah index ke 14.



Gambar 3. 23 Langkah 5 Contoh Algoritma Boyer Moore

Pada pemeriksaan selanjutnya terjadi mismatch di index 14 yaitu karakter a kemudian nilai lo dari karakter a yaitu 4 dan kasus 1 setelah di hitung maka di dapat index ke 15.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T:	a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b
											13	12	11	10	9	8				
x	a	b	c	d							a	b	a	c	a	b				
L(x)	4	5	3	-1																
P:											a	b	a	c	a	b				
j											0	1	2	3	4	5				

Gambar 3. 24 Langkah 6 Contoh Algoritma Boyer Moore

Pada pemeriksaan selanjutnya semua pemeriksaan tidak terdapat mismatch maka dalam pencarian string ini didapat nilai index ke 10 sebagai hasil dari pencarian.

#### d) Algoritma Rabin Karp

Algoritma Rabin Karp mengaplikasikan fungsi hash sebagai alat perbandingan antara string target (m) dan substring pada teks (n). Apabila kedua nilai hash ini cocok, maka perbandingan lebih lanjut akan dilakukan terhadap karakter-karakter masing-masing. Jika hasil keduanya tidak cocok, maka substring akan mengalami pergeseran ke arah kanan. Proses pergeseran ini akan terjadi sebanyak (n-m) kali. Efisiensi dalam menghitung nilai hash saat pergeseran berpengaruh terhadap kinerja algoritma ini.

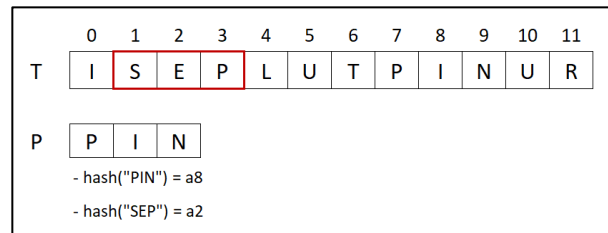
Contoh apabila kita mempunyai teks “ISEPLUTPINUR” dan pattern yang ingin dicari “PIN”. Pertama kita harus menjalankan fungsi hashing untuk pattern yang akan dicari. Sebagai contoh dari pattern “PIN” dimasukkan kedalam fungsi hash dan menghasilkan nilai “a8”. Berikut merupakan Langkah-langkah untuk melakukan algoritma String Matching Rabin Karp.

	0	1	2	3	4	5	6	7	8	9	10	11
T	I	S	E	P	L	U	T	P	I	N	U	R
P	P	I	N									
	- hash("PIN") = a8											
	- hash("ISE") = a1											

Gambar 3. 25 Langkah 1 Contoh Algoritma Rabin Karp

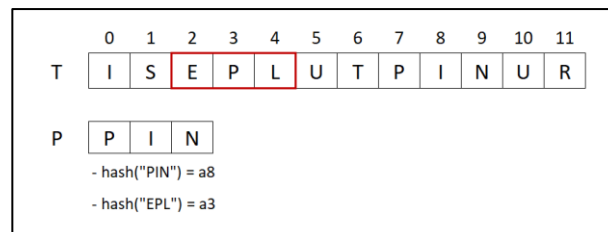
Pada Langkah pertama dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang

dicari (n). Dalam kasus diatas hash dari “ISE” adalah “a1” kemudian hash pattern yang dicari adalah “a8” maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.



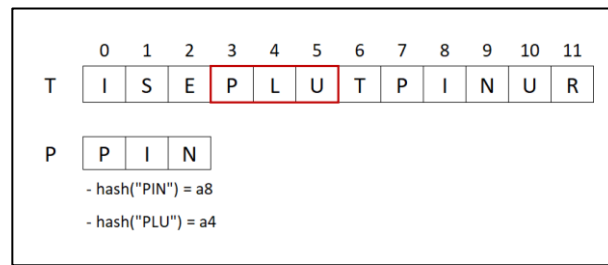
Gambar 3. 26 Langkah 2 Contoh Algoritma Rabin Karp

Kemudian dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang dicari (n). Dalam kasus diatas hash dari “SEP” adalah “a2” kemudian hash pattern yang dicari adalah “a8” maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.



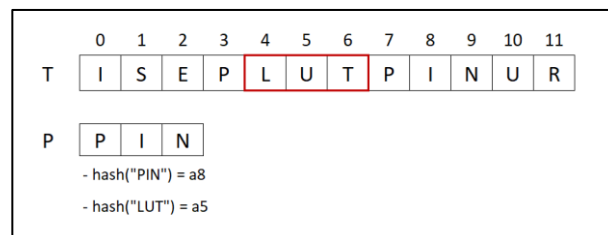
Gambar 3. 27 Langkah 3 Contoh Algoritma Rabin Karp

Kemudian dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang dicari (n). Dalam kasus diatas hash dari “EPL” adalah “a3” kemudian hash pattern yang dicari adalah “a8” maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.



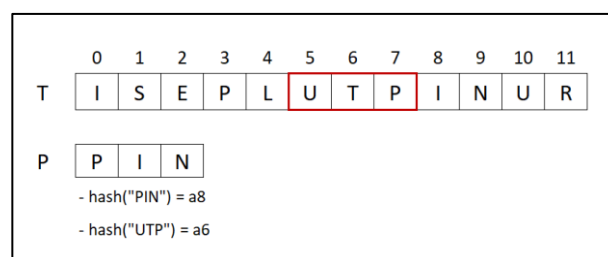
Gambar 3. 28 Langkah 4 Contoh Algoritma Rabin Karp

Kemudian dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang dicari (n). Dalam kasus diatas hash dari "PLU" adalah "a4" kemudian hash pattern yang dicari adalah "a8" maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.



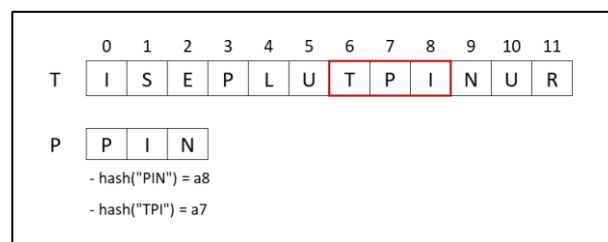
Gambar 3. 29 Langkah 5 Contoh Algoritma Rabin Karp

Kemudian dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang dicari (n). Dalam kasus diatas hash dari "LUT" adalah "a5" kemudian hash pattern yang dicari adalah "a8" maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.



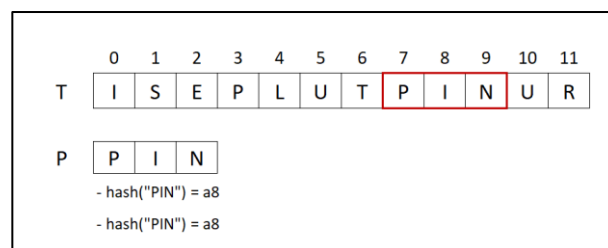
Gambar 3. 30 Langkah 6 Contoh Algoritma Rabin Karp

Kemudian dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang dicari (n). Dalam kasus diatas hash dari “UTP” adalah “a5” kemudian hash pattern yang dicari adalah “a8” maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.



Gambar 3. 31 Langkah 7 Contoh Algoritma Rabin Karp

Kemudian dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang dicari (n). Dalam kasus diatas hash dari “TPI” adalah “a7” kemudian hash pattern yang dicari adalah “a8” maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.



Gambar 3. 32 Langkah 8 Contoh Algoritma Rabin Karp

Kemudian dalam gambar diatas algoritma ini membuat hash dari beberapa karakter Pattern yang akan dicari sesuai dengan panjang pattern yang dicari (n). Dalam kasus diatas hash dari “PIN” adalah “a8” kemudian hash pattern yang dicari adalah “a8” maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.

#### e) SQL Query Like

Dalam penggunaan database MySQL terdapat operator Like yang dapat digunakan untuk mencari data dengan pola(pattern) tertentu pada sekumpulan data. Contoh penggunaan SQL Query Like sebagai berikut:

Terdapat Tabel Customer dengan beberapa data

Tabel 3. 3 Tabel Customer

id	Name	email	address	phone	gender
1	Ms. Aliyah Runolfsdotir	evan780@example.com	9195 Schoen Lodge Apt. 690 Bechtelarland, MA 23828	580-739-6844	female
2	Theron Sanford	aidan.rutherford1@example.com	8722 Reichert Ramp Laronton, CT 00628	1-203-689-8215	male
3	Jaunita Brown	daryl.blanda2@example.com	397 Feeney Shores Apt. 368 Port Daveland, ID 31389-6584	1.831E+10	female
...	...	...	...	...	...
250000	Courtney Brakus	hahn.bernardo249999@example.com	63453 Luz Motorway O'Reillyland, SC 79560	1-559-352-4733	female

Dalam kasus ini untuk mencari data dengan pattern “sandf” dalam tabel diatas yaitu dengan sintaks dibawah

**select \* from customer where ‘%sandf%’**

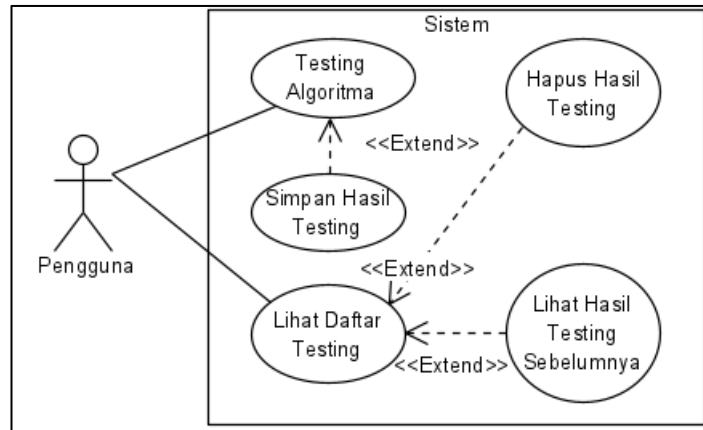
### 3.3. Perancangan UML

UML yang digunakan dalam perancangan sistem ini terdiri atas perancangan *Use Case Diagram* dan *Activity Diagram*.

#### A. Use Case Diagram

Dalam use case diagram sistem testing algoritma *string matching* terdapat aktor yaitu Pengguna yang mempunyai beberapa use case yaitu Testing Algoritma, Simpan Hasil Testing, Lihat Daftar Testing, Lihat Hasil Testing Sebelumnya,

Hapus Hasil Testing Sebelumnya dan Lihat halaman tentang. Untuk gambar use case diagram sistem testing algoritma *string matching* dapat dilihat dibawah.



Gambar 3. 33 Rancangan UML Use Case Diagram

#### a) Skenario Use Case Testing Algoritma

Deskripsi mengenai Skenario Use Case Testing Algoritma bisa di lihat pada tabel berikut:

Tabel 3. 4 Skenario use case mengelola data penyakit

<b>Nama</b>	Testing Algoritma		
<b>Aktor</b>	Pengguna		
<b>Deskripsi</b>	Merupakan menu untuk testing perbandingan algoritma <i>string matching</i> .		
<b>Skenario</b>			
<b>Aksi Aktor</b>		<b>Reaksi Sistem</b>	
Klik Menu Testing		Muncul tampilan halaman testing perbandingan algoritma <i>string matching</i> . Dengan input data pattern dan jumlah data yang akan di testing.	
Mengisi formulir input data			
Klik tombol Test		Melakukan testing dan perbandingan algoritma <i>string matching</i> .	
		Menampilkan hasil dari testing dan perbandingan algoritma <i>string matching</i> .	



**b) Skenario Use Case Simpan Hasil Testing**

Deskripsi mengenai Skenario *Use Case* Simpan Hasil Testing bisa di lihat pada tabel berikut:

Tabel 3. 5 Skenario Usecase Lihat Daftar Hasil Testing

<b>Nama</b>	Simpan Hasil Testing		
<b>Aktor</b>	Pengguna		
<b>Deskripsi</b>	Merupakan menu untuk melihat daftar testing sebelumnya yang sudah tersimpan.		
<b>Skenario</b>			
<b>Aksi Aktor</b>		<b>Reaksi Sistem</b>	
Klik Menu Daftar Hasil Testing		Muncul data daftar hasil testing sebelumnya	

**c) Skenario Use Case Lihat Daftar Hasil Testing**

Deskripsi mengenai Skenario Use Case Simpan Hasil Testing bisa di lihat pada tabel berikut:

Tabel 3. 6 Skenario use case Simpan hasil testing

<b>Nama</b>	Simpan Hasil Testing	
<b>Aktor</b>	Pengguna	
<b>Deskripsi</b>	Merupakan tombol untuk menyimpan hasil testing	
<b>Skenario</b>		
<b>Aksi Aktor</b>		<b>Reaksi Sistem</b>
Melakukan testing di halaman Testing		Setelah Menampilkan hasil dari testing dan perbandingan algoritma <i>string matching</i> .
Klik tombol simpan		Menyimpan data hasil testing sebelumnya.
Klik tombol Test		Menampilkan Peringatan bahwa data berhasil disimpan.

**d) Skenario Use Case Hapus Hasil Testing**

Deskripsi mengenai Skenario *Use Case* Hapus Hasil Testing bisa di lihat pada tabel berikut:

Tabel 3. 7 Skenario Use Case Hapus Hasil Testing

<b>Nama</b>	Hapus Hasil Testing		
<b>Aktor</b>	Pengguna		
<b>Deskripsi</b>	Merupakan tombol untuk menghapus data testing yang sudah tersimpan sebelumnya.		
<b>Skenario</b>			
<b>Aksi Aktor</b>		<b>Reaksi Sistem</b>	
Klik menu Daftar Hasil Testing		Muncul data daftar hasil testing sebelumnya.	
Klik tombol hapus di data yang ingin di hapus.		Menghapus file data testing yang sudah tersimpan sebelumnya.	

**e) Skenario Use Case Detail Hasil Testing Sebelumnya**

Deskripsi mengenai Skenario *Use Case* Detail Hasil Testing Sebelumnya bisa di lihat pada tabel berikut:

Tabel 3. 8 Skenario Use Case Hapus Hasil Testing

Nama	Detail Hasil Testing Sebelumnya	
Aktor	Pengguna	
Deskripsi	Merupakan halaman untuk melihat detail testing yang sudah di simpan sebelumnya.	
Skenario		
Aksi Aktor		Reaksi Sistem

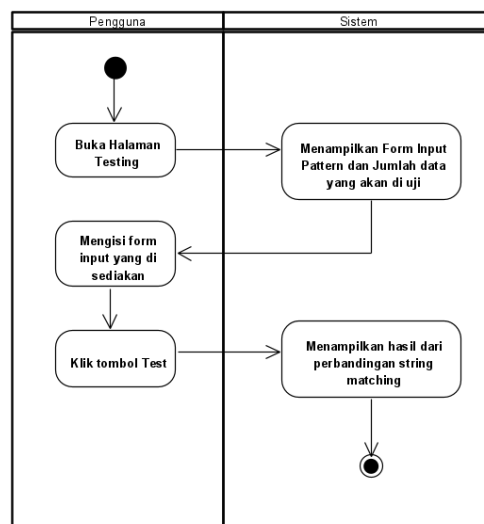
Klik Menu Daftar Hasil Testing	Muncul data daftar hasil testing sebelumnya.
Klik nama salasatu data yang sudah di simpan.	Menampilkan halaman detail data hasil testing sebelumnya.

## B. Activity Diagram

Activity Diagram menunjukan aktivitas yang dilaksanakan oleh pengguna terhadap sistem perbandingan *algorithm string matching*.

### a) Activity Diagram Testing Algoritma

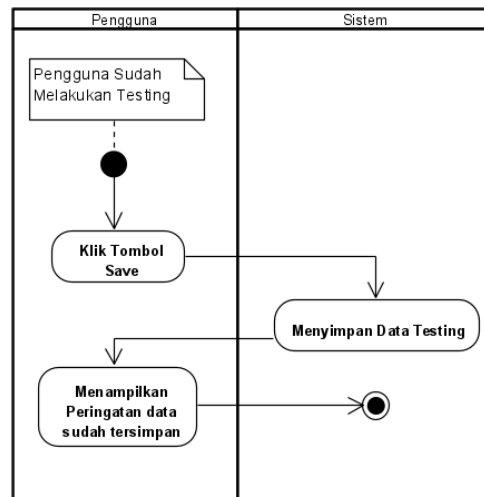
Diagram Aktivitas untuk melakukan testing algoritma yang dilaksanakan oleh pengguna pada sistem.



Gambar 3. 34 Activity Diagram Testing Algoritma

**b) Activity Diagram Simpan Hasil Testing**

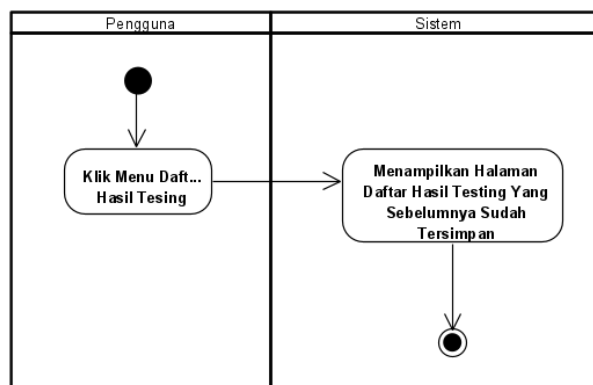
Diagram Aktivitas untuk melakukan Simpan Hasil Testing yang dilaksanakan oleh pengguna pada sistem.



*Gambar 3. 35 Activity Diagram Simpan Hasil Testing*

**c) Activity Diagram Lihat Daftar Hasil Testing**

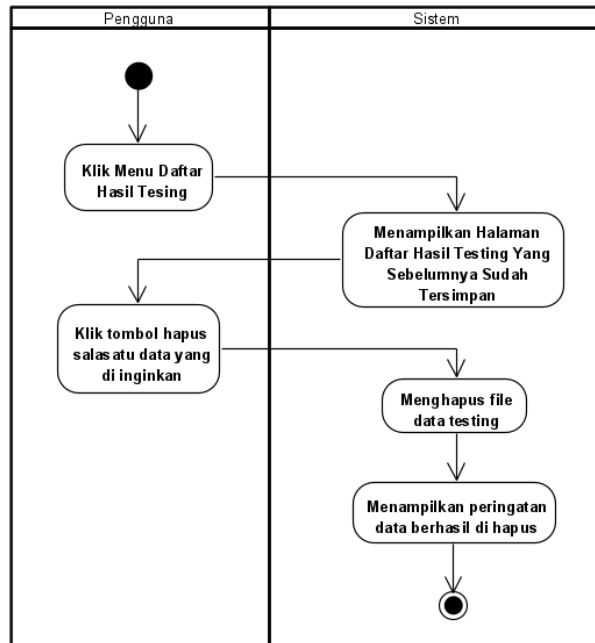
Diagram Aktivitas untuk melakukan Lihat Daftar Hasil Testing yang dilaksanakan oleh pengguna pada sistem.



*Gambar 3. 36 Activity Diagram Lihat Daftar Hasil Testing*

**d) Activity Diagram Hapus Hasil Testing**

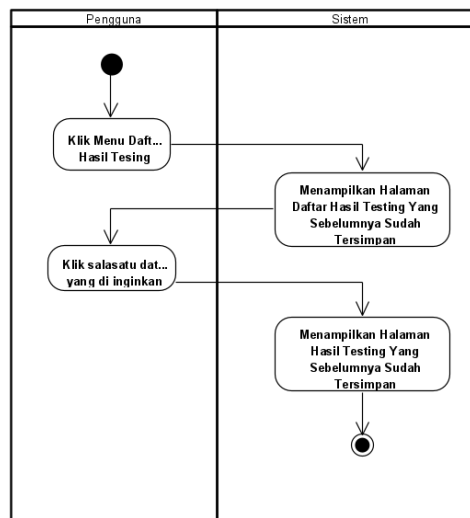
Diagram Aktivitas untuk melakukan Hapus Hasil Testing yang dilaksanakan oleh pengguna pada sistem.



Gambar 3. 37 Activity Diagram Hapus Hasil Testing

**e) Activity Diagram Detail Hasil Testing Sebelumnya**

Diagram Aktivitas untuk melakukan Detail Hasil Testing Sebelumnya yang dilaksanakan oleh pengguna pada sistem.



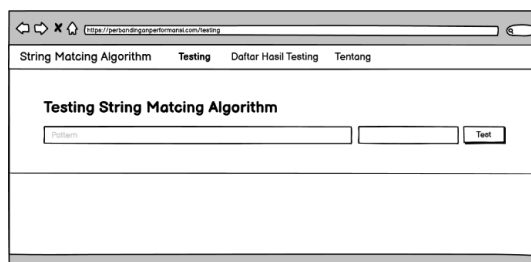
Gambar 3. 38 Activity Diagram Detail Hasil Testing Sebelumnya

### 3.4. Perancangan Antarmuka

Perancangan antarmuka atau tampilan pada pengguna yang dibuat untuk Sistem Perbandingan String Matching.

#### D. Rancangan Tampilan Halaman Testing

Dibawah ini merupakan gambar perancangan antar muka untuk halaman testing.

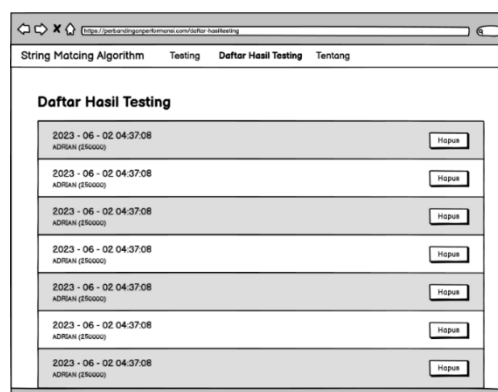


Gambar 3. 39 Rancangan Tampilan Halaman Testing

Dalam gambar rancangan diatas terdapat beberapa menu navigasi diantaranya Menu testing untuk melakukan pengujian perbandingan algoritma string matching, Kemudian ada menu Daftar Hasil Testing yaitu menu untuk melihat daftar Hasil testing yang disimpan. Kemudian di Tengah nya terdapat kolom inputan dan kolom pilihan jumlah data untuk pengujian dan disampingnya terdapat tombol testing untuk melakukan testing perbandingan algoritma string matching.

#### E. Rancangan Tampilan Halaman Daftar Hasil Testing

Dibawah ini merupakan perancangan antar muka untuk halaman daftar hasil testing.

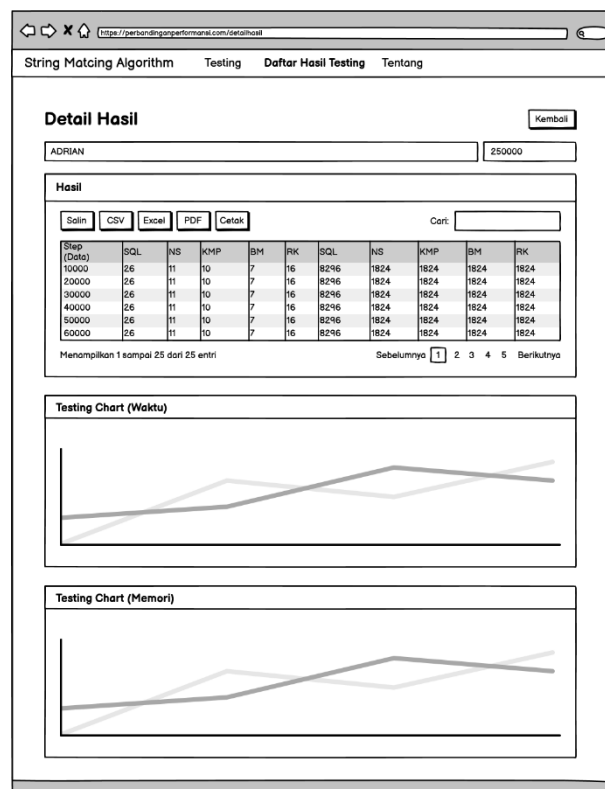


Gambar 3. 40 Rancangan Tampilan Halaman Daftar Hasil Testing

Dalam gambar diatas terdapat daftar hasil testing yang sebelumnya sudah disimpan di halaman testing. Kemudian terdapat juga tombol hapus di tiap-tiap data testing yang sudah tersimpan.

## F. Rancangan Tampilan Halaman Detail Hasil Testing Sebelumnya

Dibawah ini merupakan perancangan antar muka untuk halaman detail hasil testing Sebelumnya.



Gambar 3. 41 Rancangan Tampilan Halaman Detail Hasil Testing Sebelumnya

Dalam gambar diatas terdapat hasil testing sebelumnya yaitu Tabel Hasil Testing Kemudian chart waktu yang digunakan dari tiap-tiap metode algoritma string matching, kemudian dibawahnya untuk penggunaan memory terhadap algoritma pencarian string yang sudah dilakukan.

### 3.5. Skenario Pengujian

Pengujian ini bertujuan untuk membandingkan performansi dari berbagai algoritma string matching yaitu Algoritma Naive String Matching, Knuth Morrin Pratt, Boyer Moore, Rabin Karp dan SQL Query Like dalam berbagai skenario teks berbeda.

#### A. Variabel Pengujian

Dalam Penelitian ini terdapat variable pengujian yang terdiri dari beberapa variable meliputi:

- **Jenis Algoritma:** Variabel ini mencakup berbagai algoritma string matching yang akan diuji yaitu Algoritma Naive String Matching, Knuth Morrin Pratt, Boyer Moore, Rabin Karp dan SQL Query Like.
- **Ukuran Input:** Pengujian akan dilakukan pada teks berukuran variatif, dari panjang teks yang sangat pendek hingga teks dengan ukuran sangat besar.
- **Karakteristik Teks:** Variabel ini mencakup jenis dan distribusi karakter dalam string input, seperti apakah teks berisi banyak pengulangan, apakah teks acak, apakah teks beregu termasuk kata-kata yang umum, dll.

#### B. Parameter yang Digunakan

Dalam pengujian ini, ada dua parameter utama yang akan digunakan untuk mengevaluasi performansi algoritma:

- **Waktu Komputasi:** Waktu total yang dibutuhkan oleh setiap algoritma untuk mencapai solusi.
- **Pemakaian Memori:** Jumlah memori yang digunakan selama proses pencarian.



### C. Skenario

Untuk pengujian algoritma string matching terdapat 2 skenario yang akan dijalankan sebagai berikut:

- **Pengujian Kasus Rata-rata (Average Case):** Pengujian ini membandingkan algoritma pada input acak.
- **Pengujian Dengan Karakteristik Teks Khusus:** Pengujian ini melibatkan teks dengan karakteristik khusus seperti simbol angka dan lain sebagainya.
- **Pengujian Dengan Banyak Pengguna:** Pengujian dilakukan dengan lebih dari 5 pengguna dalam satu waktu secara bersamaan.
- **Pengujian Perbandingan Media Penyimpanan Data:** Pengujian ini dilakukan dengan membandingkan pengambilan dataset dari database MySQL secara langsung dengan dataset dari file temporary JSON.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN SISTEM

#### 4.1. Batasan Implementasi

Dalam penerapan perangkat lunak, perlengkapan perangkat keras dan perangkat lunak yang memadai sangat diperlukan untuk menyokong terus-menerus selama proses pembuatan program. Komponen yang digunakan saat melakukan implementasi meliputi:

##### A. Komputer perangkat keras

Adapun komputer yang digunakan memiliki detail teknis sebagai berikut :

Tabel 4. 1 Komputer perangkat keras

No	Perangkat Keras	Spesifikasi
1	Processor	Intel(R) Core(TM) i3-1005G1 2 CPU @ 1.20GHz
2	Harddisk	SSSTC CL1-4D512 512GB
3	Ram	12 GB DDR4 2666 MHz
4	VGA	NVIDIA GeForce MX330 2 GB
5	Monitor	15.6 inch
6	Lan Card	Intel(R) Wireless-AC 9560

##### B. Perangkat lunak

Untuk mengimplementasikan rancangan yang telah disusun, dibutuhkan sejumlah perangkat lunak untuk operasionalisasi program aplikasi, yang mencakup:

Tabel 4. 2 Perangkat Lunak

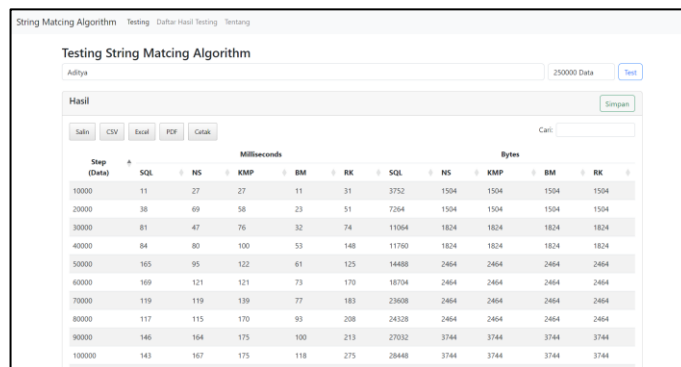
No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	Windows 11 Pro v22621.1778
2	Bahasa Pemrograman	PHP v8.1.12
3	Database	MariaDB v15.1
4	Server	Apache v2.4.54
5	Browser	Microsoft Edge v13.0.1774.57

## 4.2. Implementasi Antarmuka

Tahap implementasi sistem mengacu pada tahap di mana desain yang dibuat berdasarkan analisis ditransformasikan ke dalam bahasa pemrograman tertentu serta aplikasi perangkat lunak yang diciptakan diterapkan pada lingkungan yang sebenarnya. Sistem Perbandingan String Matching ini diterapkan dalam kerangka sistem Berbasis Web.

### A. Halaman Testing

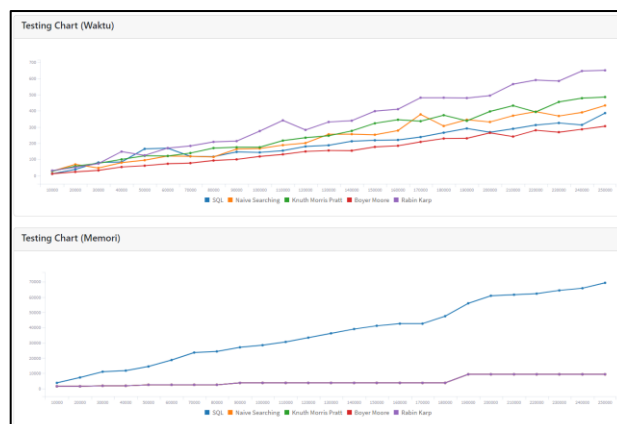
Berikut ini adalah contoh pengujian string matching dengan kata kunci Aditya Degnan menggunakan 250.000 data. Deskripsi seperti di bab 3.4



Step (Data)	Milliseconds					Bytes				
	SQL	NS	KMP	BM	RK	SQL	NS	KMP	BM	RK
10000	11	27	27	11	31	3752	1504	1504	1504	1504
20000	38	69	58	23	51	7264	1504	1504	1504	1504
30000	81	47	76	32	74	11064	1824	1824	1824	1824
40000	84	80	100	53	148	11760	1824	1824	1824	1824
50000	165	95	122	61	125	14488	2464	2464	2464	2464
60000	169	121	121	73	170	18704	2464	2464	2464	2464
70000	119	119	139	77	183	23608	2464	2464	2464	2464
80000	117	115	170	93	208	24328	2464	2464	2464	2464
90000	146	164	175	100	213	27032	3744	3744	3744	3744
100000	143	167	175	118	275	28448	3744	3744	3744	3744

Gambar 4. 1 Halaman Testing Tabel 10.000-100.000 Data

Dalam gambar diatas memperlihatkan hasil implementasi dari perancangan sebelumnya, dalam pengujian tersebut digunakan Pattern “Aditya” dengan 250000 Data.



Gambar 4. 2 Halaman Testing Diagram Garis Kecepatan Waktu dan Penggunaan Memory

Dalam gambar diatas memperlihatkan hasil chart hasil dari pengujian untuk mempermudah keterbacaan saat menganalisa data hasil pengujian.

## 2. Halaman Daftar Hasil Testing

Halaman ini menampilkan daftar testing yang sudah disimpan pada testing sebelumnya. Untuk gambar nya bisa di lihat di bawah.

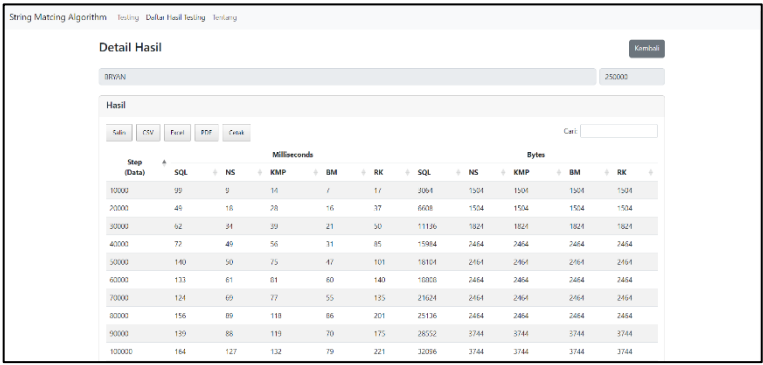


Daftar Hasil Testing		
2023-06-02 04:22:39	BRYAN (250000)	Hapus
2023-06-02 04:23:29	BRYAN (250000)	Hapus
2023-06-02 04:24:25	BRYAN (250000)	Hapus
2023-06-02 04:25:18	BRYAN (250000)	Hapus
2023-06-02 04:26:21	BRYAN (250000)	Hapus
2023-06-02 04:27:57	ANDY (250000)	Hapus
2023-06-02 04:28:42	ANDY (250000)	Hapus
2023-06-02 04:29:50	ANDY (250000)	Hapus
2023-06-02 04:30:34	ANDY (250000)	Hapus
2023-06-02 04:33:37		

Gambar 4. 3 Halaman Daftar Hasil Testing

## 3. Halaman Detail Hasil Testing Sebelumnya

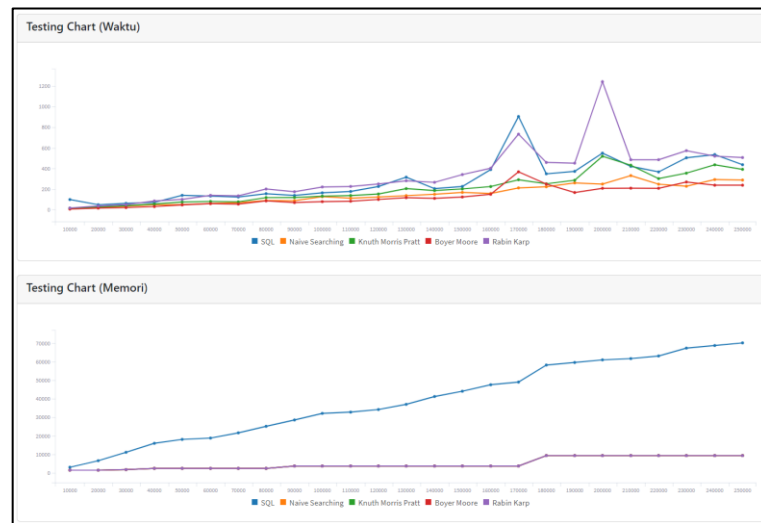
Berikut ini adalah halaman data hasil pengujian string matching yang sudah tersimpan dengan kata kunci Aditya Degnan menggunakan 250.000 data.



Detail Hasil													
													Kembali
BRYAN													250000
Hasil													
													Cari
Step (Data)	A	SQL	NS	KMP	BM	RK	SQL	NS	KMP	BM	RK		
10000	39	9	14	7	17	3064	1301	1301	1301	1301	1301		
20000	48	18	28	16	37	6008	1504	1504	1504	1504	1504		
30000	62	24	39	21	50	11136	1821	1821	1821	1821	1821		
40000	77	49	56	31	85	15984	2464	2464	2464	2464	2464		
50000	140	59	75	47	101	18104	2161	2161	2161	2161	2161		
60000	133	61	81	60	140	18000	2464	2464	2464	2464	2464		
70000	124	69	77	55	135	21624	2161	2161	2161	2161	2161		
80000	156	89	119	86	201	25136	2464	2464	2464	2464	2464		
90000	199	86	119	70	175	28552	3744	3744	3744	3744	3744		
100000	164	127	132	79	221	32096	3744	3744	3744	3744	3744		

Gambar 4. 4 Halaman Detail Testing Sebelumnya - Tabel 100.000-250.000 Data

Dalam gambar diatas memperlihatkan data hasil pengujian yang telah dilakukan sebelumnya.



Gambar 4. 5 Halaman Testing Sebelumnya - Diagram Garis Kecepatan Waktu dan Penggunaan Memory

### 4.3. Pengujian Sistem

Langkah pengujian merupakan langkah yang ditempuh setelah penyelesaian tahap implementasi. Fungsi dari bagian ini adalah untuk memverifikasi kesesuaian antara sistem/aplikasi yang telah dikembangkan dengan niat perancangan awal. Pengujian sistem merangkum serangkaian uji coba yang dilakukan pada masing-masing komponen dalam sebuah sistem/aplikasi. Alasan diadakannya evaluasi terhadap suatu aplikasi adalah untuk menemukan dan memperbaiki kesalahan atau kekurangan yang berpotensi terjadi pada aplikasi tersebut. Metode pengujian yang diterapkan dalam studi ini melibatkan uji hitam alias *blackbox* dan pengujian beta.

#### A. Pengujian Blackbox

Uji *blackbox* merupakan teknik evaluasi yang menekankan pada kepuasan kriteria fungsi dari sebuah aplikasi. Pembuktian dilaksanakan dengan memasukkan sebuah rangkaian proses input dan kemudian mengevaluasi hasil output dari proses input tersebut.

Berikut adalah hasil dari pemanfaatan uji *blackbox* pada Sistem Perbandingan String Matching ini.

### 1. Halaman Testing

Hasil pengujian melalui metode *blackbox* yang diaplikasikan pada halaman testing dapat disimpulkan sebagai berikut.

Tabel 4. 3 Hasil pengujian blackbox pada halaman Testing

Pengujian	Fungsi	Hasil Pengujian
Pattern	Untuk menampilkan inputan pattern	Sesuai
Jumlah Data Testing	Untuk menampilkan dropdown jumlah input data testing	Sesuai
Tombol Testing	Untuk Memulai Testing kemudian menampilkan hasil	Sesuai
Tombol Simpan	Untuk menyimpan hasil testing yang sudah dilakukan sebelumnya	Sesuai

### 2. Halaman Daftar Testing

Hasil pengujian melalui metode *blackbox* yang diaplikasikan pada halaman daftar testing dapat disimpulkan sebagai berikut.

Tabel 4. 4 Halaman Daftar Testing

Pengujian	Fungsi	Hasil Pengujian
Daftar Testing Sebelumnya	Untuk menampilkan daftar testing yang sudah dilakukan sebelumnya	Sesuai
Hapus Data Testing	Untuk menghapus data testing yang sudah di lakukan sebelumnya	Sesuai
Link Detail Data Testing	Untuk Berpindah halaman melihat detail data testing	Sesuai

### 3. Halaman Detail Testing Sebelumnya

Hasil pengujian melalui metode *blackbox* yang diaplikasikan pada halaman detail testing sebelumnya dapat disimpulkan sebagai berikut.

Tabel 4. 5 Halaman Detail Testing Sebelumnya

Pengujian	Fungsi	Hasil Pengujian
Detail Testing	Menampilkan Data Testing yang sudah tersimpan sebelumnya.	Sesuai

### B. Pengujian Beta

Dalam penelitian ini, pengujian beta dilakukan dengan cara membiarkan pengguna mencoba aplikasi, kemudian pengisian kuesioner oleh pengguna. Kuesioner ini bertujuan untuk mengumpulkan dan mengatur data yang telah diperoleh yang nantinya akan diolah untuk mendukung hasil penelitian dan memungkinkan penarikan kesimpulan.

Kuesioner ini berjudul Perbandingan Performansi Metode String Matching Menggunakan Metode Naive String Matching, Knuth Morris Pratt, Boyer Moore, Rabin Karp dan SQL Query Like Untuk Pencarian Data Konsumen. Kuesioner ini akan disajikan kepada responden dengan 5 pertanyaan. Jawaban akan mengikuti skala Likert dari 1 hingga 5. Berikut adalah penjelasan rinci tentang skala Likert:

Tabel 4. 6 Skala likert dan interval

Jawaban	Skor	Interval Penilaian
Sangat Setuju	5	Index 80 % - 100 %
Setuju	4	Index 60 % - 79,99 %
Netral	3	Index 40 % - 59,99 %
Tidak Setuju	2	Index 20 % - 39,99 %
Sangat Tidak Setuju	1	Index 0 % - 19,99 %

Untuk menghitung index persentase, digunakan rumus di bawah.

$$Index (\%) = \frac{Total\ Skor \times 100\%}{Skor\ Maksimum}$$

Rumus berikut digunakan untuk mengkalkulasi nilai maksimum.

$$\begin{aligned}\text{Skor Maksimum} &= \text{Total Responder} \times \text{Jawaban Maksimum} \\ &= 20 \times 5 \\ &= 100\end{aligned}$$

Skor maksimum pada kasus ini adalah 100.

Berikut adalah daftar pertanyaan yang disajikan kepada responden dalam tabel berikut.

Tabel 4. 7 Daftar pertanyaan

No	Skor
1	Apakah tampilan dari Sistem yang dibangun menarik?
2	Apakah tampilan Sistem yang dibangun sesuai dengan fungsinya?
3	Apakah Sistem yang dibangun mudah untuk digunakan?
4	Apakah Sistem yang dibangun berjalan lancar dan tidak ada error saat digunakan?
5	Apakah aplikasi Sistem yang dibangun sudah layak dipublikasikan dan siap digunakan?

Kuesioner ini diberikan kepada 20 individu yang berasal dari berbagai disiplin ilmu, termasuk individu yang berkecimpung dalam bidang teknologi dan juga individu non-teknis. Perkiraan persentase untuk setiap jawaban dapat dilihat di bawah ini.

1. Apakah tampilan dari Sistem yang dibangun menarik?

Tabel 4. 8 Kuesioner pertanyaan 1

Kategori jawaban	Skor	Frekuensi Jawaban	Total Skor	Nilai Index Persentase
Sangat Setuju	5	6	30	= $80 / 100 \times 100 \%$ = 80 %
Setuju	4	9	36	
Netral	3	4	12	
Tidak Setuju	2	1	2	
Sangat Tidak Setuju	1	0	0	
Jumlah		20	80	



Dari penghitungan tabel diatas, terkumpul skor total sebesar 80. Selanjutnya, berdasarkan nilai persentasi responden sebesar 80% yang termasuk dalam rentang penilaian sangat setuju, maka dapat disarikan bahwa daya tarik tampilan dari aplikasi sistem pendukung keputusan ini telah terpenuhi.

2. Apakah tampilan Sistem yang dibangun sesuai dengan fungsinya?

Tabel 4.9 Kuesioner pertanyaan 2

Kategori jawaban	Skor	Frekuensi Jawaban	Total Skor	Nilai Index Persentase
Sangat Setuju	5	9	45	$= 89 / 100 \times 100 \%$ $= 89 \%$
Setuju	4	11	44	
Netral	3	0	0	
Tidak Setuju	2	0	0	
Sangat Tidak Setuju	1	0	0	
Jumlah		20	89	

Dari perhitungan yang terdapat pada tabel di atas, didapatkan skor keseluruhan sebesar 89. Hasil nilai persentase para responden, yaitu 89%, termasuk dalam rentang penilaian yang sangat setuju. Oleh karena itu, dapat disimpulkan bahwa tampilan aplikasi sistem pendukung keputusan ini telah sesuai dengan fungsi yang diharapkan.

3. Apakah Sistem yang dibangun mudah untuk digunakan?

Tabel 4.10 Kuesioner pertanyaan 3

Kategori jawaban	Skor	Frekuensi Jawaban	Total Skor	Nilai Index Persentase
Sangat Setuju	5	9	45	$= 86 / 100 \times 100 \%$ $= 86 \%$
Setuju	4	8	32	
Netral	3	3	9	
Tidak Setuju	2	0	0	
Sangat Tidak Setuju	1	0	0	
Jumlah		20	86	

Berdasarkan perhitungan pada tabel di atas, total skor yang didapat adalah 86 skor. Sedangkan hasil dari nilai persentase responden 86 % yang masuk pada interval penilaian sangat setuju. Maka dapat disimpulkan bahwa aplikasi sistem pendukung keputusan ini mudah digunakan.

4. Apakah Sistem yang dibangun berjalan lancar dan tidak ada error saat digunakan?

Tabel 4.11 Kuesioner pertanyaan 4

Kategori jawaban	Skor	Frekuensi Jawaban	Total Skor	Nilai Index Persentase
Sangat Setuju	5	6	30	$= 84 / 100 \times 100 \%$ $= 84 \%$
Setuju	4	12	48	
Netral	3	2	6	
Tidak Setuju	2	0	0	
Sangat Tidak Setuju	1	0	0	
Jumlah		20	84	

Berdasarkan perhitungan pada tabel di atas, total skor yang didapat adalah 84 skor. Sedangkan hasil dari nilai persentase responden 84 % yang masuk pada interval penilaian sangat setuju. Maka dapat disimpulkan bahwa aplikasi sistem pendukung keputusan ini sudah berjalan lancar dan tidak ada error.

5. Apakah aplikasi Sistem yang dibangun sudah layak dipublikasikan dan siap digunakan?

Tabel 4.12 Kuesioner pertanyaan 5

Kategori jawaban	Skor	Frekuensi Jawaban	Total Skor	Nilai Index Persentase
Sangat Setuju	5	6	30	$= 84 / 100 \times 100 \%$ $= 84 \%$
Setuju	4	12	48	
Netral	3	2	6	
Tidak Setuju	2	0	0	
Sangat Tidak Setuju	1	0	0	
Jumlah		20	84	

Berdasarkan perhitungan pada tabel di atas, total skor yang didapat adalah 84 skor. Sedangkan hasil dari nilai persentase responden 84 % yang masuk pada interval penilaian sangat setuju. Maka dapat disimpulkan bahwa aplikasi sistem pendukung keputusan ini sudah layak dipublikasikan dan siap digunakan.

Hasil perhitungan rata-rata persentase indeks nilai untuk tiap pertanyaan bisa dilihat pada kalkulasi berikut.

$$\text{Rata-rata index persentase} = \frac{84 \% + 84 \% + 86 \% + 89 \% + 80 \%}{5} = 84,6 \%$$

Rata-rata index persentase dari setiap pertanyaan adalah 84,6 % yang masuk pada interval penilaian sangat setuju. Maka dapat disimpulkan bahwa sistem Perbandingan Performansi Metode String Matching Menggunakan Metode Naive String Matching, Knuth Morris Pratt, Boyer Moore, Rabin Karp dan SQL Query Like Untuk Pencarian Data Konsumen, akurat dan objektif berdasarkan skenario pengujian yang sudah ditentukan.

### C. Hasil Penelitian

Hasil penelitian ini diambil dari skenario pengujian yang sudah di deskripsikan sebelumnya. Hasil dari skenario pengujian sebagai berikut:

#### 1. Pengujian Kasus Rata-rata (Average Case)

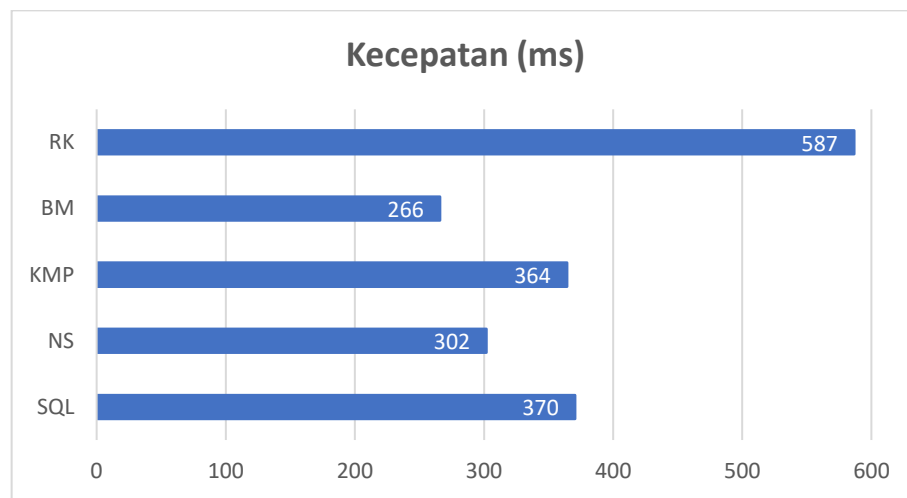
Pada skenario ini akan dilakukan pengujian dengan menggunakan kata kunci “ANDY” selama 5 kali dengan jumlah data 250.000, hasil dari pengujian sebagai berikut:

- **Kecepatan:** Pengujian ini menggunakan satuan milidetik sebagai berikut.

Tabel 4. 13 Hasil Pengujian dengan skenario kasus rata-rata (Kecepatan)

Pengujian	Kecepatan (ms)				
	SQL	NS	KMP	BM	RK
1	334	310	373	272	522
2	331	287	336	272	656
3	474	309	367	255	563
4	368	276	367	281	599
5	345	327	378	250	593
Rata-Rata	370	302	364	266	587

Dalam tabel diatas bisa diketahui hasil dari 5 kali pengujian algoritma Boyer More lebih cepat dibanding algoritma yang lain dengan selisih 104ms dengan SQL Query Like atau 39.25% lebih cepat.



Gambar 4. 6 Hasil Pengujian dengan skenario kasus rata-rata (Kecepatan)

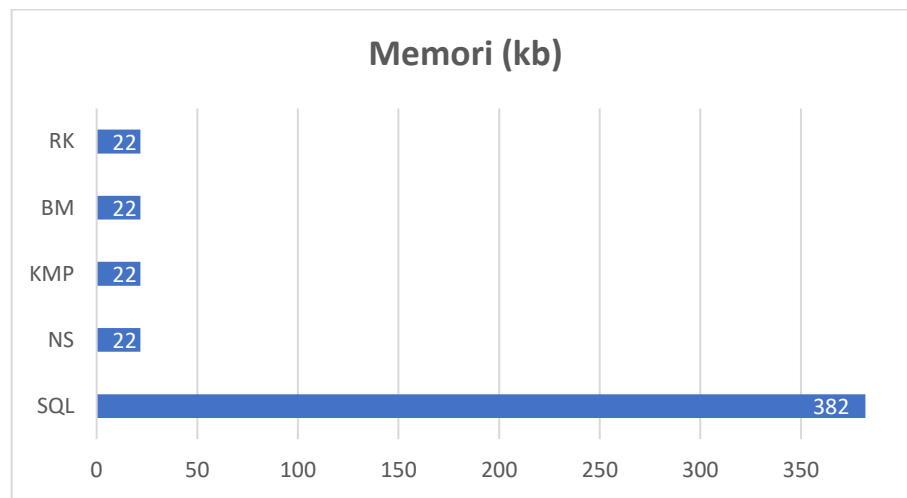
Dalam gambar diagram garis diatas bisa diketahui bahwa algoritma Boyer Moore 120,53% lebih cepat disbanding algoritma Rabin Karp dengan selisih 321 Milidetik lebih cepat.

- **Memori:** Pengujian ini menggunakan satuan bytes sebagai berikut.

Tabel 4. 14 Hasil Pengujian dengan skenario kasus rata-rata (Memori)

Pengujian	Memori (kb)				
	SQL	NS	KMP	BM	RK
1	382	22	22	22	22
2	382	22	22	22	22
3	382	22	22	22	22
4	382	22	22	22	22
5	382	22	22	22	22
Rata-Rata	382	22	22	22	22

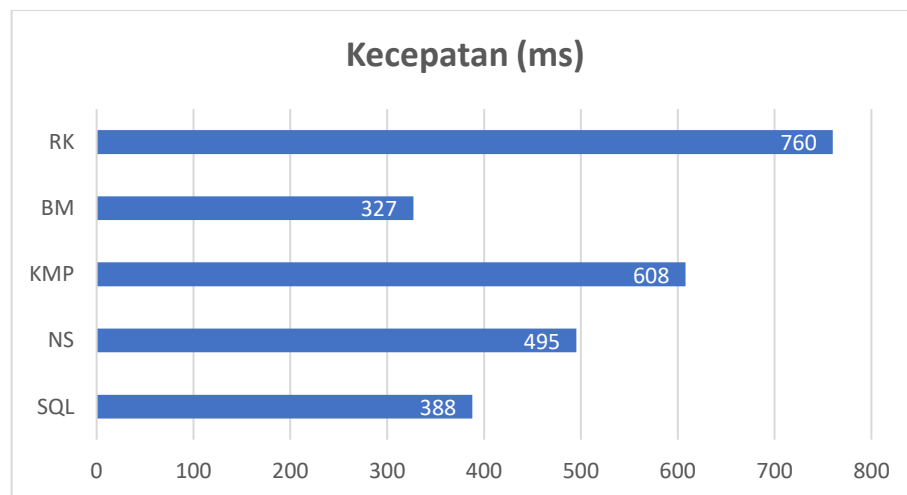
Dalam tabel diatas bisa diketahui hasil dari 5 kali pengujian algoritma rata-rata memori yang digunakan SQL lebih banyak dibandingkan algoritma yang lain.



Gambar 4. 7 Hasil Pengujian dengan skenario kasus rata-rata (Memori)

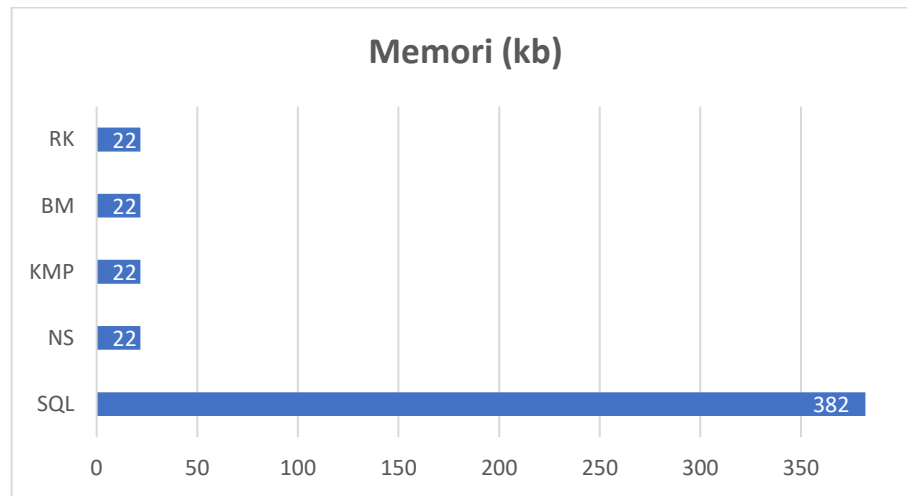
## 2. Pengujian Dengan Karakteristik Teks Khusus

Untuk pengujian dengan karakteristik teks khusus ini menggunakan pattern/Kata kunci “123” dengan jumlah pengujian selama 5 kali dengan jumlah data 250.000, hasil dari pengujian sebagai berikut:



Gambar 4. 8 Pengujian Dengan Karakteristik Teks Khusus (Kecepatan)

Dari hasil pengujian dalam gambar diatas bisa diketahui bahwa algoritma Boyer Moore lebih baik dari segi kecepatan dengan selisih 61ms atau 18,65% lebih cepat dibandingkan algoritma Rabin karp.



Gambar 4. 9 Pengujian Dengan Karakteristik Teks Khusus (Memory)

Dalam gambar diatas bisa diketahui hasil dari pengujian algoritma memori yang digunakan SQL lebih banyak dibandingkan algoritma yang lain.

### 3. Pengujian Dengan Banyak Pengguna

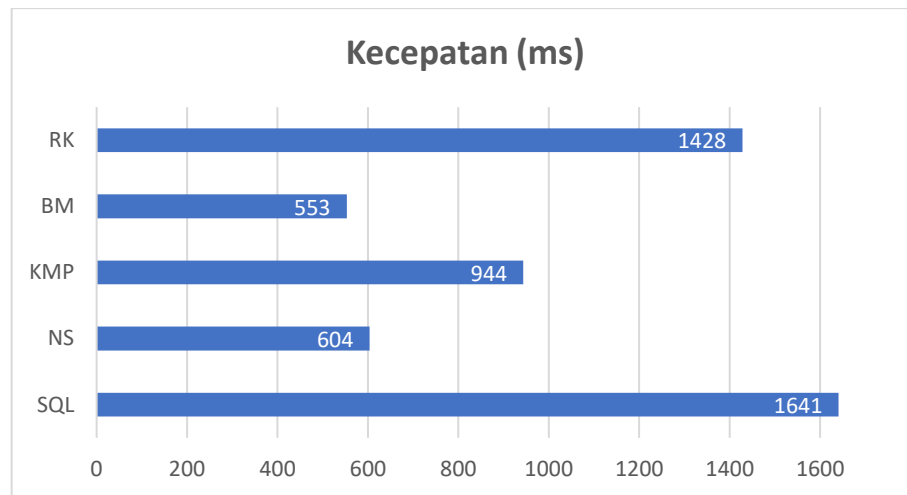
Untuk pengujian ini dilakukan dengan cara melakukan pengujian secara bersamaan dengan 6 pengguna sekaligus dalam waktu yang bersamaan. dengan menggunakan kata kunci “ANDY” selama 5 kali dengan jumlah data 250.000 Hasil dari pengujian sebagai berikut:

Tabel 4. 15 Hasil Skenario Pengujian Dengan Banyak Pengguna

Pengujian	Kecepatan (ms)				
	SQL	NS	KMP	BM	RK
1	1337	689	1219	568	1491
2	2143	473	614	497	1020
3	1337	689	1219	568	1491
4	1467	637	923	550	1431
5	1467	637	923	550	1431
6	2094	496	764	586	1705
Rata-Rata	1641	604	944	553	1428

Dari tabel diatas bisa diketahui untuk skenario pengujian ini algoritma Boyer Moore tetap menjadi algoritma yang paling cepat dibanding yang lain. Kemudian

pada pengujian ini selisih kecepatan antara algoritma Boyer Moore dengan SQL 1088ms atau 196,63% Lebih cepat,

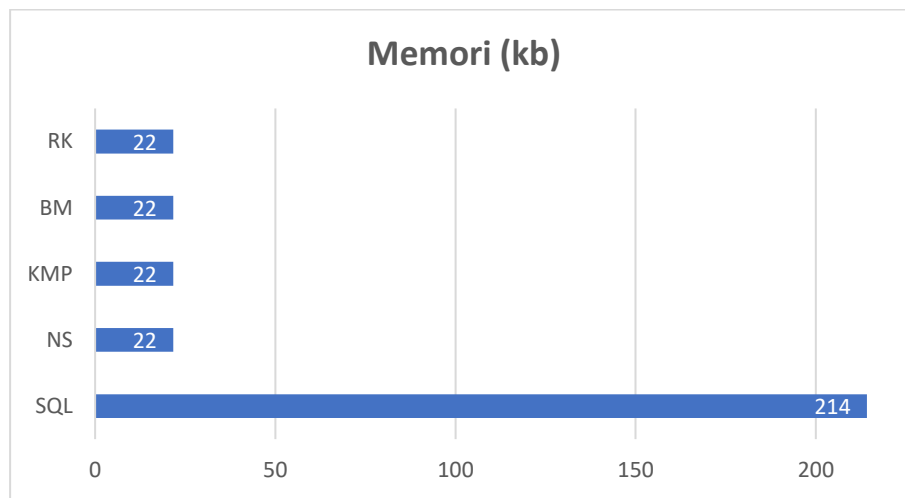


Gambar 4. 10 Hasil Skenario Pengujian Dengan Banyak Pengguna

Dari gambar diatas bisa diketahui kecepatan boyer moore dengan 6 pengguna sekaligus yaitu 553ms, kemudian untuk kecepatan algoritma boyer moore dengan 1 pengguna yaitu 266ms (dari skenario sebelumnya) dengan selisih 287ms atau 107,96% lebih cepat.

Kemudian untuk kecepatan SQL Query Like dengan 6 pengguna sekaligus yaitu 1641ms dan untuk kecepatan SQL Query Like dengan 1 pengguna sekaligus yaitu 370ms dengan selisih 1270ms atau 343% lebih cepat.

Kenaikan dari setiap penambahan pengguna bisa diambil dengan membagi jumlah pengguna. Dalam skenario ini jumlah penggunanya adalah 6 kemudian hasil kecepatan algoritma Boyer Moore 266ms atau 107,96% dibagi 6 hasilnya untuk setiap penambahan jumlah pengguna terjadi kenaikan 47,86ms atau 18%, sedangkan hasil kecepatan SQL Query Like 1270ms atau 343% dibagi 6 hasilnya untuk setiap penambahan jumlah pengguna terjadi kenaikan 273ms atau 57,16%.



Gambar 4. 11 Hasil pengujian Memory dengan 5 pengguna sekaligus

Dalam gambar diatas bisa diketahui hasil dari pengujian algoritma memori yang digunakan SQL lebih banyak dibandingkan algoritma yang lain.

#### 4. Pengujian Perbandingan Media Penyimpanan Data

Pengujian ini akan mengambil dataset dari database SQL secara langsung tanpa temporary file JSON, Untuk perbandingan pengujian dengan dataset dari temporary JSON diambil dari pengujian skenario sebelumnya. Pengujian ini dilakukan dengan menggunakan kata kunci “ANDY” selama 5 kali dengan jumlah data 250.000 Hasil dari pengujian sebagai berikut:

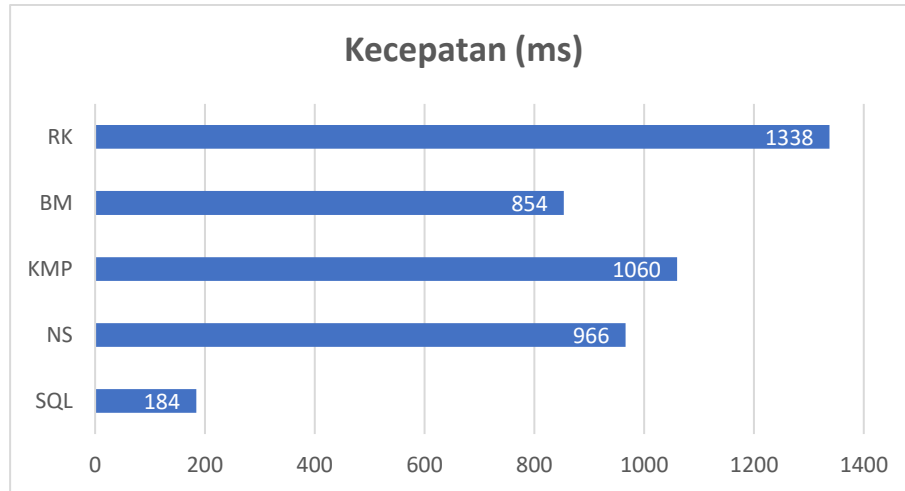
Tabel 4. 16 Pengujian dengan dataset yang diambil dari MySQL langsung tanpa Temporary JSON

Pengujian	Kecepatan (ms)				
	SQL	NS	KMP	BM	RK
1	184	875	1009	856	1313
2	184	875	1009	856	1313
3	195	1062	1190	933	1444
4	181	1046	1055	845	1434
5	177	973	1036	779	1185
Rata-Rata	184	966	1060	854	1338

Dari tabel diatas untuk pengujian ini bisa di ketahui bahwa rata-rata kecepatan metode boyer moore adalah 854ms dan pada pengujian skenario sebelumnya

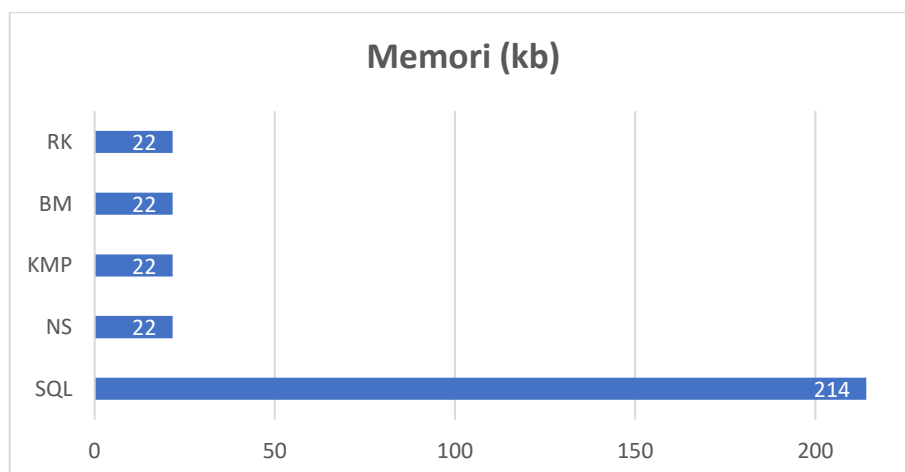


dengan menggunakan dataset dari temporary JSON yaitu 266ms dengan selisih 588ms atau 220,98% lebih cepat.



Gambar 4. 12 Pengujian dengan dataset yang diambil dari MySQL langsung tanpa Temporary JSON

Dari gambar diatas bisa dilihat SQL Query Like lebih cepat dibanding algoritma yang lain karena pada algoritma yang lain terjadi penambahan langkah.



Gambar 4. 13 Pengujian dengan dataset yang diambil dari MySQL langsung tanpa Temporary JSON (Memori)

Dalam gambar diatas bisa diketahui hasil dari pengujian algoritma memori yang digunakan SQL lebih banyak dibandingkan algoritma yang lain.

## **BAB V**

### **PENUTUP**

#### **5.1. Kesimpulan**

Berdasarkan hasil penelitian yang dilakukan maka dapat diambil kesimpulan sebagai berikut:

- Algoritma Boyer More lebih cepat dibanding algoritma yang lain. selisih kecepatan 104ms dengan SQL Query Like atau 39.25% lebih cepat berdasarkan pengujian performansi dengan skenario kasus rata-rata untuk pencarian data dengan pattern/kata kunci “ANDY” selama 5 kali dengan jumlah data 250,000.
- Penggunaan memory rata-rata memori yang digunakan SQL lebih banyak dibandingkan algoritma yang lain.

#### **5.2. Saran**

Berdasarkan hasil penelitian ini ini membuka ruang untuk beberapa peningkatan yang bisa diterapkan. Berikut ini adalah beberapa di antaranya:

1. Melakukan uji coba menggunakan spesifikasi perangkat keras yang lebih unggul dan lebih variatif.
2. Melakukan uji coba dengan melibatkan jumlah pengguna yang lebih besar pada waktu yang sama.
3. Uji coba dilakukan menggunakan hosting server.
4. Dalam penelitian ini, 250.000 data pelanggan digunakan dan jumlah ini dapat ditingkatkan.


## DAFTAR PUSTAKA

- [1] F. Riawan, "Knuth Morris Pratt String Matching Algorithm in Searching for Zakat Information and Social Activities," *Journal of Applied Data Sciences*, vol. 3, no. 1, 2022, doi: 10.47738/jads.v3i1.49.
- [2] K. Prihandani, "Tinjauan Kualitas Pengembangan Sistem Informasi Dengan Metode Agile .," *Tinjauan Kualitas Pengembangan Sistem Informasi Dengan Metode Agile .*, no. October, 2016.
- [3] Sugi Priharto, "Pentingnya Data Pelanggan untuk Meningkatkan Profit Bisnis di Era Digital." <https://kledo.com/blog/data-pelanggan-untuk-keuntungan-bisnis/> (accessed Jul. 07, 2023).
- [4] C. E. L. R. L. R. and C. S. Thomas H. Cormen, "Introduction to Algorithms, fourth edition," *April 5, 2022*. <https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/> (accessed Jul. 07, 2023).
- [5] Z. Zhang, "Review on String-Matching Algorithm," *SHS Web of Conferences*, vol. 144, 2022, doi: 10.1051/shsconf/202214403018.
- [6] A. P. Nemytykh, "On Specialization of a Program Model of Naive Pattern Matching in Strings (Extended Abstract)," Aug. 2021, [Online]. Available: <http://arxiv.org/abs/2108.10865>
- [7] A. Fatah *et al.*, "Application of knuth-morris-pratt algorithm on web based document search," in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Jun. 2019. doi: 10.1088/1742-6596/1175/1/012117.
- [8] C. Irawan and M. Riyan Pratama, "Perbandingan Algoritma Boyer Moore dan Brute Force pada Pencarian Kamus Besar Bahasa Indonesia Berbasis Android," *BIOS : Jurnal Teknologi Informasi dan Rekayasa Komputer*, vol. 1, no. 2, 2020.
- [9] L. Syafie, M. Resha, B. Besar Penelitian dan Pengembangan SDM KOMINFO Makassar, and S. AKBA Makassar, "RABIN-CARP IMPLEMENTATION IN MEASURING SIMILIRITY OF RESEARCH PROPOSAL OF STUDENTS."
- [10] W. Kim, "On Optimizing an SQL-like Nested Query."


- [11] N. Koch and A. Kraus, “The expressive power of uml-based web engineering,” *Second International Workshop on Weboriented Software Technology IWWOST02*, vol. 16, pp. 105–119, 2002.
- [12] Rendi Juliarto, “Apa itu UML? Beserta Pengertian dan Contohnya,” <https://www.dicoding.com/>, 2022. <https://www.dicoding.com/blog/apa-itu-uml/> (accessed Jan. 08, 2023).

## LAMPIRAN

### A. Kartu Bimbingan



**KARTU BIMBINGAN SKRIPSI**  
**S1 – TEKNIK INFORMATIKA**  
 UNIVERSITAS SANGGA BUANA - YPKP

TAHUN AJAR	Gerap 2022/2023	
NPM	2113191079	
NAMA	Isep Lupi Nur	
PEMBIMBING	Gunawansyah, S.T., M.Kom.	
JUDUL	Perbandingan Performansi Metode String Matching Menggunakan Metode Naïve String Matching Knuth Morris Pratt, Boyer Moore, Rabin Karp dan SQL Query Like Untuk Pencarian Data Konsumen	

NO	TANGGAL	POKOK BAHASAN	PARAF PEMBIMBING
1	2023-07-22	Pembahasan hasil seminar proposal, Penentuan dan Pembuatan Jadwal	
2	2023-07-11	Pembahasan judul BAB1 dan BAB2	
3	2023-07-28	BAB2 sistem yg diusulkan	
4	2023-08-24	Revisi BAB3 mencari pola penelitian	
5	2023-08-20	Revisi BAB3 Data penelitian	
6	2023-08-14	Revisi BAB3 Pembahasan metode	
7	2023-08-20	Revisi BAB3 contoh dari tiap metode	
8	2023-08-18	Revisi melengkapi Bab 2.3.4	
9	2023-08-29	Revisi Bab 3.4 Hasil penelitian	
10	2023-08-05	Revisi Bab 4.5 dan kelegkapan lain	


  

Catatan:

1. Minimal bimbingan sebanyak 8x.
2. Kartu ini dikumpulkan sebagai syarat sidang beserta berkas yang lainnya.

Bandung, 05 Agustus 2023

Pembimbing

  
 [Gunawansyah, S.T., M.Kom.]

## B. Surat Pernyataan Keabsahan Data Penelitian

### Surat Pernyataan Keabsahan Data Penelitian

Yang bertanda tangan di bawah ini:

Nama : Isep Lutpi Nur

NIM : 2113191079

Program Studi : S1 Teknik Informatika

Dengan ini menyatakan bahwa:

1. Data yang digunakan dalam penelitian ini untuk penyusunan skripsi dengan judul: *"Perbandingan Performansi Metode String Matching Menggunakan Metode Naive String Matching, Knuth Morris Pratt, Boyer-Moore, Rabin Karp Dan Sql Query Like Untuk Pencarian Data Konsumen"* adalah data yang valid dan telah melalui proses pengumpulan, pengolahan, serta analisis secara benar dan sesuai dengan metode yang telah ditetapkan.
2. Data yang digunakan dalam penelitian ini telah disetujui oleh pembimbing skripsi yang bertanda tangan di bawah ini:

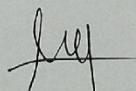
Nama Pembimbing: Gunawansyah, S.T., M.Kom

NIDN : 0420027907

Demikian surat pernyataan ini dibuat dengan sebenarnya dan apabila di kemudian hari ditemukan adanya ketidakbenaran dalam pernyataan ini, maka saya bersedia menerima sanksi yang berlaku sesuai dengan ketentuan yang berlaku di Universitas Sangga Buana YPKP.

Bandung, 10 Agustus 2023

Peneliti/ Mahasiswa

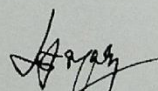


**Isep Lutpi Nur**

NPM. 2113191079

Mengetahui:

Pembimbing Skripsi



**Gunawansyah, S.T., M.Kom**

NIDN : 0420027907



## C. Kuisioner Pengujian Beta

https://docs.google.com/spreadsheets/d/1zdnVgtHvrtfCQzT3h7R7enGrwQC0suXIMgLNjU9Y6/edit?resourcekey=gid=1902888792

Kuisioner sistem Perbandingan Performansi Metode String Matching Menggunakan Metode Naive String Matching, Knuth Morris Pratt, Boyer-Moore, Rabin Karp dan SQL

	A	B	C	D	E	F	G	H
1	Timestamp	Nama Lengkap	Email / No. Telp	Apakah tampilan dari Sistem yang dibangun menarik?	Apakah tampilan Sistem yang dibangun sesuai dengan fungsinya?	Apakah Sistem yang dibangun mudah untuk digunakan?	Apakah Sistem yang dibangun berjalan lancar dan tidak ada error saat digunakan?	Apakah aplikasi Sistem yang dibangun sudah layak dipublikasikan dan siap digunakan?
2	10/08/2023 15:09:10	Kenanga Ayu Mentari	kenangaayu.xpm2@gmail.com	Sangat Setuju	Sangat Setuju	Sangat Setuju	Sangat Setuju	Sangat Setuju
3	10/08/2023 15:09:43	Fadhil Muhammad Prasetyo	fadhilprasetyo123@gmail.com	Setuju	Setuju	Sangat Setuju	Setuju	Sangat Setuju
4	10/08/2023 15:10:39	Muhanmad Faza Alfariis	faza017910@gmail.com	Sangat Setuju	Sangat Setuju	Netral	Sangat Setuju	Sangat Setuju
5	10/08/2023 15:12:02	Dzakry Naufal Fauzyas	dzakrynaufal@gmail.com	Sangat Setuju	Sangat Setuju	Sangat Setuju	Setuju	Sangat Setuju
6	10/08/2023 15:13:01	Muhanmad Ihsan	m.ihsan.binja@gmail.com	Sangat Setuju	Setuju	Setuju	Setuju	Sangat Setuju
7	10/08/2023 15:13:41	Muhanmad Rifan Setiadi	rifanmuhammad354@gmail.com	Netral	Sangat Setuju	Setuju	Sangat Setuju	Sangat Setuju
8	10/08/2023 15:14:22	Nuriela Meliani	nrfmeliani@gmail.com	Setuju	Setuju	Sangat Setuju	Setuju	Netral
9	10/08/2023 15:15:03	Dita Ajeng Ramadhani	ditaaramadhani06@gmail.com	Sangat Setuju	Sangat Setuju	Netral	Sangat Setuju	Netral
10	10/08/2023 15:15:57	Dlabith Logictiawan	dlabithlogictiawan25@gmail.com	Netral	Sangat Setuju	Setuju	Sangat Setuju	Setuju
11	10/08/2023 15:16:35	Erik Faturohman	erikfaturohman54@upi.edu	Sangat Setuju	Setuju	Sangat Setuju	Setuju	Setuju
12	10/08/2023 15:17:25	Shina Nureni Nazilah	shinanazilah2@gmail.com	Netral	Sangat Setuju	Setuju	Sangat Setuju	Setuju
13	10/08/2023 15:18:00	Listia Andriani	lialisia42@gmail.com	Setuju	Setuju	Sangat Setuju	Netral	Setuju
14	10/08/2023 15:18:28	Silviyani Wijaya	silviyanivijaya2001@gmail.com	Setuju	Sangat Setuju	Setuju	Setuju	Setuju
15	10/08/2023 15:19:04	Alena Alifah	alenaalifah2004@gmail.com	Setuju	Setuju	Sangat Setuju	Netral	Setuju
16	10/08/2023 15:19:38	Agnia Nurul Fitri	agnia066@gmail.com	Netral	Sangat Setuju	Setuju	Setuju	Setuju
17	10/08/2023 15:20:10	Siti Ningrum	085863611038	Setuju	Setuju	Setuju	Setuju	Setuju
18	10/08/2023 15:20:58	Hendry Yana Abdul Hamid	62895336596699	Tidak Setuju	Setuju	Netral	Setuju	Setuju
19	10/08/2023 15:21:25	Neng Hilmi Numalhufah	088229478774	Setuju	Setuju	Setuju	Setuju	Setuju
20	10/08/2023 15:21:59	Ajeng Nurul Sholihah Ahmad	+6289609973216	Setuju	Setuju	Sangat Setuju	Setuju	Setuju
21	10/08/2023 15:22:21	Dinda Khairunnisa Sabila	dinkhasa25@gmail.com	Setuju	Setuju	Sangat Setuju	Setuju	Setuju
22								
23								
...								

#### D. Kode Program Algoritma String Matching

Berikut merupakan kode program dari algoritma string matching yang digunakan dalam skripsi ini, algoritma string matching ini menggunakan bahasa pemrograman PHP.

##### 1. Algoritma Naïve String Matching

```
function NaiveSearching($text, $pattern)
{
    // Ambil panjang pola dan teks
    $M = strlen($pattern);
    $N = strlen($text);

    // Putar satu demi satu
    for ($i = 0; $i <= $N - $M; $i++) {

        // Cek dengan putaran pattern
        for ($j = 0; $j < $M; $j++) {
            if ($text[$i + $j] != $pattern[$j]) break;
        }

        // Cek jika semuanya benar (nilai j = panjang pattern) maka
        // keluarkan indeks i
        // Variable j diambil dari perulangan diatas
        if ($j == $M) {
            return $i;
        }
    }
    return -1;
}

// contoh
$text = "Isep Lutpi Nur";
$pattern = "Lutpi";
NaiveSearching($text, $pattern);
```



## 2. Algoritma Knuth Morris Pratt

```
function KMP($text, $pattern, $border)
{
    $M = strlen($pattern);
    $N = strlen($text);

    // prefix dan suffix terpanjang/longest prefix suffix
    $lps = $border;

    $i = 0; // index untuk txt[]
    $j = 0; // index untuk pat[]
    while (($N - $i) >= ($M - $j)) {
        if ($pattern[$j] == $text[$i]) {
            $j++;
            $i++;
        }

        if ($j == $M) {
            return $i - $j;
            $j = $lps[$j - 1];
        }

        // jika terjadi mismatch setelah j matches
        else if ($i < $N && $pattern[$j] != $text[$i]) {
            // Do not match lps[0..lps[j-1]] characters,
            // they will match anyway
            if ($j != 0) {
                $j = $lps[$j - 1];
            } else {
                $i = $i + 1;
            }
        }
    }
    return -1;
}
```

```

function KMPComputeBorder($pattern)
{
    // ababababca
    $M = strlen($pattern);
    $len = 0; // panjang dari previous longest prefix suffix
    $lps[0] = 0; // lps[0] selalu 0
    $i = 1; // the loop calculates lps[i] for i = 1 to M-1
    while ($i < $M) {
        if ($pattern[$len] == $pattern[$i]) {
            $len++;
            $lps[$i] = $len;
            $i++;
        } else {
            // to search step.
            if ($len != 0) {
                $len = $lps[$len - 1];

                // Also, note that we do not increment
                // i here
            } else {
                // if (len == 0)
                $lps[$i] = 0;
                $i++;
            }
        }
    }
    return $lps;
}

```

```

// contoh
$text = "Isep Lutpi Nur";
$pattern = "Lutpi";
$border = KMPComputeBorder($pattern);
KMP($text, $pattern, $border);

```

### 3. Algoritma Boyer Moore

```
function BoyerMoore($text, $pattern, $lo)
{
    $patlen = strlen($pattern);
    $textlen = strlen($text);

    for ($i = $patlen - 1; $i < $textlen;) {
        $t = $i;
        for ($j = $patlen - 1; $pattern[$j] == $text[$i]; $j--, $i--)
        ) {
            if ($j == 0) {
                return $i;
            }
        }

        $i = $t;
        if (array_key_exists($text[$i], $lo)) {
            $i = $i + max($lo[$text[$i]], 1);
        } else {
            $i += $patlen;
        }
    }
    return -1;
}

function lo($string)
{
    $len = strlen($string);
    $table = [];
    for ($i = 0; $i < $len; $i++) {
        $table[$string[$i]] = $len - $i - 1;
    }
    return $table;
}

// contoh
$text = "Isep Lutpi Nur";
$pattern = "Lutpi";
$lo = lo($pattern);
BoyerMoore($text, $pattern, $lo);
```

#### 4. Rabin Karp

```
function RabinKarp($text, $pattern)
{
    $d = 256; // Number of characters in the input alphabet
    $q = 101; // A prime number

    $M = strlen($pattern);
    $N = strlen($text);
    $p = 0; // Hash value for pattern
    $t = 0; // Hash value for text
    $h = 1;

    // Calculate the hash value of pattern and first window of text
    for ($i = 0; $i < $M - 1; $i++) $h = ($h * $d) % $q;

    for ($i = 0; $i < $M; $i++) {
        $p = ($d * $p + ord($pattern[$i])) % $q;
        $t = ($d * $t + ord($text[$i])) % $q;
    }

    // Slide the pattern over text one by one
    for ($i = 0; $i <= $N - $M; $i++) {

        // Check the hash values of current window of text and
        pattern
        // If the hash values match, then only check for characters
        one by one
        if ($p == $t) {
            $match = true;

            // Check for characters one by one
            for ($j = 0; $j < $M; $j++) {
                if ($text[$i + $j] != $pattern[$j]) {
                    $match = false;
                    break;
                }
            }

            // Pattern found
            if ($match) return $i;
        }

        // Calculate the hash value for the next window of text
        if ($i < $N - $M) {
```

```

        $t = ($d * ($t - ord($text[$i]) * $h) + ord($text[$i +
$M])) % $q;

        // If the calculated hash value is negative, make it
positive
        if ($t < 0) $t = $t + $q;
    }
}

return -1;
}

// contoh
$text = "Isep Lutpi Nur";
$pattern = "Lutpi";
RabinKarp($text, $pattern);

```