**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**

**LAPORAN SKRIPSI**

Logo

Description automatically generated

**Oleh :**

**ISEP LUTPI NUR**

**2113191079**

**FAKULTAS TEKNIK**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**UNIVERSITAS SANGGA BUANA YPKP**

**2023**

# **DAFTAR ISI**

[**DAFTAR ISI** I](#_Toc136635777)

[**DAFTAR GAMBAR** III](#_Toc136635778)

[**DAFTAR TABEL** IV](#_Toc136635779)

[**BAB I PENDAHULUAN** 1](#_Toc136635780)

[**1.1.** **Latar Belakang Masalah** 1](#_Toc136635781)

[**1.2.** **Rumusan Masalah** 2](#_Toc136635782)

[**1.3.** **Batasan Masalah** 3](#_Toc136635783)

[**1.4.** **Maksud dan Tujuan** 3](#_Toc136635784)

[**1.5.** **Metode Penelitian** 4](#_Toc136635785)

[**A.** **Metode Pegumpulan Data** 4](#_Toc136635786)

[**B.** **Metode Pengembangan Sistem** 4](#_Toc136635787)

[**C.** **Metode Pengembangan Perangkat Lunak** 4](#_Toc136635788)

[**1.6.** **Sistematika Penulisan** 5](#_Toc136635789)

[**BAB II LANDASAN TEORI** 7](#_Toc136635790)

[**2.1.** **Data Konsumen** 7](#_Toc136635791)

[**2.2.** **Algoritma** 7](#_Toc136635792)

[***2.3.*** **Algoritma *String Matching*** 8](#_Toc136635793)

[**2.4.** **Algoritma *Naive String Matching*** 8](#_Toc136635794)

[**2.5.** **Algoritma *Knuth Morris Pratt*** 9](#_Toc136635795)

[**2.6.** **Algoritma *Boyer Moore*** 9](#_Toc136635796)

[**2.7.** **Algoritma *Rabin Karp*** 10](#_Toc136635797)

[**2.8.** **SQL Query Like** 10](#_Toc136635798)

[**2.9.** **UML (Unified Modelling Language)** 10](#_Toc136635799)

[**A.** ***Use Case* Diagram** 11](#_Toc136635800)

[**B.** ***Activity Diagram*** 12](#_Toc136635801)

[**C.** ***Sequence Diagram*** 13](#_Toc136635802)

[**D.** ***Class Diagram*** 14](#_Toc136635803)

[**2.10.** **PHP (PHP: Hypertext Preprocessor)** 15](#_Toc136635804)

[**2.11.** **Basis Data (MYSQL)** 15](#_Toc136635805)

[**BAB III ANALISIS DAN PERANCANGAN SISTEM** 17](#_Toc136635806)

[**3.2.** **Sistem Berjalan** 17](#_Toc136635807)

[**3.3.** **Data Penelitian** 17](#_Toc136635808)

[**3.4.** **Metode penelitian** 18](#_Toc136635809)

[**A.** **Algoritma Naive String Matching** 18](#_Toc136635810)

[**B.** **Algoritma Knuth Morris Pratt** 21](#_Toc136635811)

[**C.** **Algoritma Boyer Moore** 25](#_Toc136635812)

[**D.** **Algoritma Rabin Karp** 25](#_Toc136635813)

[**E.** **SQL Query Like** 25](#_Toc136635814)

[**3.5.** **Perancangan UML** 25](#_Toc136635815)

[**A.** **Use Case Diagram** 26](#_Toc136635816)

[**B.** **Activity Diagram** 29](#_Toc136635817)

[**3.6.** **Deskripsi Rencana Pengujian** 32](#_Toc136635818)

[**BAB IV IMPLEMENTASI DAN PENGUJIAN SISTEM** 33](#_Toc136635819)

[**4.1.** **Implementasi Sistem** 33](#_Toc136635820)

[**A.** **Halaman Testing** 33](#_Toc136635821)

[**A.** **Daftar Hasil Testing** 34](#_Toc136635822)

[**B.** **Detail Hasil Testing** 35](#_Toc136635823)

[**4.2.** **Batasan Implementasi** 37](#_Toc136635824)

[**A.** **Komputer perangkat keras** 37](#_Toc136635825)

[**B.** **Perangkat lunak** 37](#_Toc136635826)

[**4.3.** **Pengujian Sistem** 38](#_Toc136635827)

[**A.** **Pengujian Blackbox** 38](#_Toc136635828)

[**B.** **Pengujian Beta** 39](#_Toc136635829)

[**BAB V PENUTUP** 40](#_Toc136635830)

[**5.1.** **Kesimpulan** 40](#_Toc136635831)

[**5.2.** **Saran** 40](#_Toc136635832)

[**DAFTAR PUSTAKA** 41](#_Toc136635833)

# **DAFTAR GAMBAR**

# **DAFTAR TABEL**

# **BAB I PENDAHULUAN**

* 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 *Naive String Matching, Knuth-Morris-Pratt*, *Boyer-Moore*, 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. **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, Knuth Morris Pratt, Boyer-Moore, 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. **Batasan Masalah**

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

1. Penelitian ini hanya akan membahas performa dari algoritma string matching *Naive String Matching, Knuth Morris Pratt, Boyer-Moore, 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 *Naive String Matching, Knuth Morris Pratt, Boyer-Moore, Rabin-Karp* dan *SQL Query Like* yang paling tepat dalam pencarian data berdasarkan karakteristik data dan kebutuhan pengguna.
   1. **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 *Naive String Matching, Knuth Morris Pratt, Boyer-Moore, Rabin-Karp* dan *SQL Query Like* 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 penguna sekaligus misalnya toko online.
   1. **Metode Penelitian**

Metode penelitian yang diterapkan dibagi menjadi tiga bagian, yaitu metode pengumpulan data, metode pengembangan sistem dan metode pengembangan perangkat lunak.

* + 1. **Metode Pegumpulan Data**

Metode pengumpulan data yang diambil dalam penelitian ini adalah sebagai berikut:

* + - 1. Studi pustaka

Pada tahap ini penulis mencari referensi teori yang relevan dengan studi kasus atau permasalahan yang ditemukan. Referensi tersebut didapatkan dari jurnal, buku, situs-situs internet dan artikel laporan penelitian.

* + - 1. Eksperimen

Pengumpulan data penelitian dilakukan dengan melakukan percobaan terhadap suatu hal, yaitu pengambilan sampel objek (dataset) yang dilanjutkan dengan observasi, pelatihan, dan pencatatan hasil yang berkaitan dengan penelitian.

* + 1. **Metode Pengembangan Sistem**

Metode pengembangan sistem dalam penelitian ini menggunakan lima metode string matching yaitu Algoritma *Naive String Matching, Knuth Morris Pratt, Boyer-Moore, Rabin-Karp* dan *SQL Query Like.*

* + 1. **Metode Pengembangan Perangkat Lunak**

Metode pengembangan perangkat lunak dalam penelitian ini menggunakan metode Agile. Metode Agile adalah metode yang mengutamakan keunggulan teknis saat mengembangkan perangkat lunak. Kesederhanaan dianggap sangat penting bagi Agile dalam mengoptimalkan sumber daya yang dimiliki. Masing-masing tim pengembang Agile melakukan refleksi agar dapat bekerja secara efektif dan memiliki pola kerja yang baik [2].

Diagram

Description automatically generated

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. **Sistematika Penulisan**

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

**BAB I: PENDAHULUAN**

Pendahuluan berisi tentang latar belakang dari permasalahan yang dihadapi, rumusan masalah, batasan masalah, maksud dan tujuan, metode penelitian serta sistematika penulisan.

**BAB II: LANDASAN TEORI**

Landasan Teori tentang pembahasan tentang teori-teori yang ada kaitannya dengan penelitian yang sedang dilakukan, seperti konsep dasar model pengembangan sistem, konsep dasar pemrograman, dan peralatan pendukung sistem yang dijadikan sebagai rujukan dalam melakukan penelitian.

**BAB III: ANALISIS DAN PERANCANGAN**

Analisis serta perancangan isinya mengenai penjelasan tentang analisa perancangan sistem yang akan dibangun, yang meliputi analisis sistem berjalan, sistem yang diusulkan, perancangan UML, perancangan *database*, serta perancangan antar muka.

**BAB IV: IMPLEMENTASI DAN PENGUJIAN SISTEM**

Berisikan tentang implementasi, pengujian dan analisa hasil dari penelitian terhadap properti dari aplikasi android. Sehingga bisa diketahui apakah sistem tersebut sanggup menuntaskan masalah yang ada serta cocok dengan tujuan dari penelitian ini.

**BAB V: PENUTUP**

Berisikan tentang kesimpulan dari pelaksanaan tugas akhir ini. Penutup juga berisikan tentang saran, di mana saran tersebut bisa bermanfaat untuk pembaharuan sistem pada waktu ke depannya.

# **BAB II LANDASAN TEORI**

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 Prilaku.

1. **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.

1. **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].

1. **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.

1. **Algoritma *Knuth Morris Pratt***

Algoritma *Knuth-Morris-Pratt* adalah salah satu algoritma pencarian string, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977.

Algoritma Knuth-Morris-Pratt (KMP) merupakan salah satu algoritma pencocokan pola. Metode pencarian KMP bekerja dengan melewatkan perbandingan-perbandingan yang tidak diperlukan untuk menghindari besarnya jumlah perbandingan, dengan demikian mencapai waktu berjalan O(n+m) yang optimal dalam kasus terburuk (worst case) pencocokan pola algoritma harus memeriksa semua karakter teks dan semua karakter dari pola setidaknya sekali. Ide utama algoritma KMP adalah untuk preprocess string pola P sehingga untuk menghitung fungsi kegagalan f yang menunjukan pergesaran P yang tepat sehingga kita dapat menggunakan kembali perbandingan yang dilakukan sebelumnya [7].

1. **Algoritma *Boyer Moore***

Algoritma Boyer-Moore diperkenalkan oleh Bob Boyer dan J.S. Moore pada tahun 1977. Pada algoritma ini pencocokan kata dimulai dari karakter terakhir kata kunci menuju karakter awalnya. Jika terjadi perbedaan antara karakter terakhir kata kunci dengan kata yang dicocokkan maka karakter-karakter dalam potongan kata yang dicocokkan tadi akan diperiksa satu per satu. Hal ini dimaksudkan untuk mendeteksi apakah ada karakter dalam potongan kata tersebut yang sama dengan karakter yang ada pada kata kunci. Apabila terdapat kesamaan, maka kata kunci akan digeser sedemikian rupa sehingga posisi karakter yang sama terletak sejajar, dan kemudian dilakukan kembali pencocokan karakter terakhir dari kata kunci. Sebaliknya jika tidak terdapat kesamaan karakter, maka seluruh karakter kata kunci akan bergeser ke kanan sebanyak m karakter, di mana m adalah panjang karakter dari kata kunci [8].

### **Mekanisme umum alogitma Boyer Moore**

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

**The looking-glass technique**

Looking glass technique digunakan untuk memeriksa kecocokan pattern P dengan teks T, dimulai dari indeks terakhir pada P. Pemeriksaan terhadap T tetap dimulai dari awal, dalam hal ini indeks I akan dimulai pada nilai m – 1.

Contoh :

A green and blue squares with white text

Description automatically generated

Gambar 2. 1 The looking-glass technique

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

**The character-jump technique**

Character jump technique merupakan Teknik Ketika terjadi mismatch antara (P[j] != 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:

1. **Kasus 1**

Pada kasus ini mismatch terjadi pada T[i] dan P[j] dan karakter apada T[i] adalah x, kemudian jika terdapat karakter x di P dengan indeks yang lebih kecil daripada j, maka geser P ke kanan agar posisi x di T[i] sejajar dengan posisi kemunculan terakhir/Last Occurrence(LO) x di P.

A diagram of a diagram

Description automatically generated

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

Pada pemeriksaan berikutnya, indeks j selalu dimulai pada indeks terakhir (m-1) yang di geser adalah posisi indeks 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.

1. **Kasus 2**

Pada 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 dengan indeks yang lebih besar daripada j. geser P satu karakter ke kanan agar posisi indeks terakhir P sejajar dengan posisi akhir T sebelumnya + 1.

A diagram of a computer

Description automatically generated

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

Pemeriksaan indeks 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.

1. **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 indeks pertama P sejajar dengan indeks i + 1.

A screenshot of a computer

Description automatically generated

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

Pemeriksaan indeks berikutnya tetap dimulai dari indeks 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.

### **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}

A group of black letters

Description automatically generated

L(a) = 4 -> Kemunculan terakhir karakter a pada P ada di indeks 4

L(b) = 3 -> Kemunculan terakhir karakter b pada P ada di indeks 3

L(c) = 5 -> Kemunculan terakhir karakter c pada P ada di indeks 5

L(d) = -1 -> Kemunculan terakhir karakter d tidak muncul

A group of numbers in a grid

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

1. **Algoritma *Rabin Karp***

Algoritma Rabin-Karp adalah algoritma pencocokan string yang menggunakan fungsi hash sebagai pembanding antara string yang dicari (m) dengan substring pada teks (n). Apabila hash value keduanya sama maka akan dilakukan perbandingan sekali lagi terhadap karakter-karakternya. Apabila hasil keduanya tidak sama, maka substring akan bergeser ke kanan. Pergeseran dilakukan sebanyak (n-m) kali. Perhitungan nilai hash yang efisien pada saat pergeseran akan mempengaruhi performa dari algoritma ini [9].

1. **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".

1. **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:

### ***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

### ***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. 5 Contoh Activity Diagram

### ***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. 6 Contoh Sequence Diagram

### ***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. 7 Contoh Class Diagram

1. **PHP (PHP: Hypertext Preprocessor)**

PHP adalah sebuah bahasa pemrograman ***server side scripting*** yang bersifat ***open source***. Sebagai sebuah scripting language, PHP menjalankan instruksi pemrograman saat proses runtime. Hasil dari instruksi tentu akan berbeda tergantung data yang diproses.PHP merupakan bahasa pemrograman server-side, maka script dari PHP nantinya akan diproses di server. Jenis server yang sering digunakan bersama dengan PHP antara lain ***Apache, Nginx***, dan [*LiteSpeed*](https://www.niagahoster.co.id/blog/apa-itu-litespeed/)**.**Selain itu, PHP juga merupakan bahasa pemrograman yang bersifat open source. Pengguna bebas memodifikasi dan mengembangkan sesuai dengan kebutuhan mereka.

1. **Basis Data (MYSQL)**

Database atau basis data adalah kumpulan informadi yang disimpan di dalam komputer secara sistematik sehingga dapat diperiksa menggunakan suatu program komputer untuk memperoleh informasi dari basis data tersebut. Kegunaan utama sistem basis data adalah agar pemakai mampu menyusun suatu pandangan (view) abstraksi data.

Database juga bisa diartikan sebagai sebuah sistem yang berfungsi mengumpulkan data, arsip, atau tabel yang disimpan dan terhubung ke media elektronik, seperti aplikasi atau situs web. Database membuat penyimpanan dan pengelolaan data lebih efisien.

MySQL dibaca MY-ES-KYOO-EL [maɪˌɛsˌkjuːˈɛl]. Beberapa orang bahkan membaca MySQL sebagai “my sequel”. Kegunaan atau fungsi MySQL adalah untuk data warehousing (gudang data), yaitu pengumpulan data dari berbagai sumber, untuk e-commerce, maupun aplikasi logging.

Pengembang pertama MySQL adalah MySQL AB, sebuah perusahaan asal Swedia, yang memulai perjalanannya di tahun 1994. Hak kepemilikan MySQL kemudian diambil secara menyeluruh oleh perusahaan teknologi Amerika Serikat, Sun Microsystems, ketika mereka membeli MySQL AB pada tahun 2008.

Di tahun 2010, Oracle yang adalah salah satu perusahaan teknologi terbesar di Amerika Serikat mengakuisisi Sun Microsystems. Sejak saat itu, MySQL sepenuhnya dimiliki oleh Oracle.

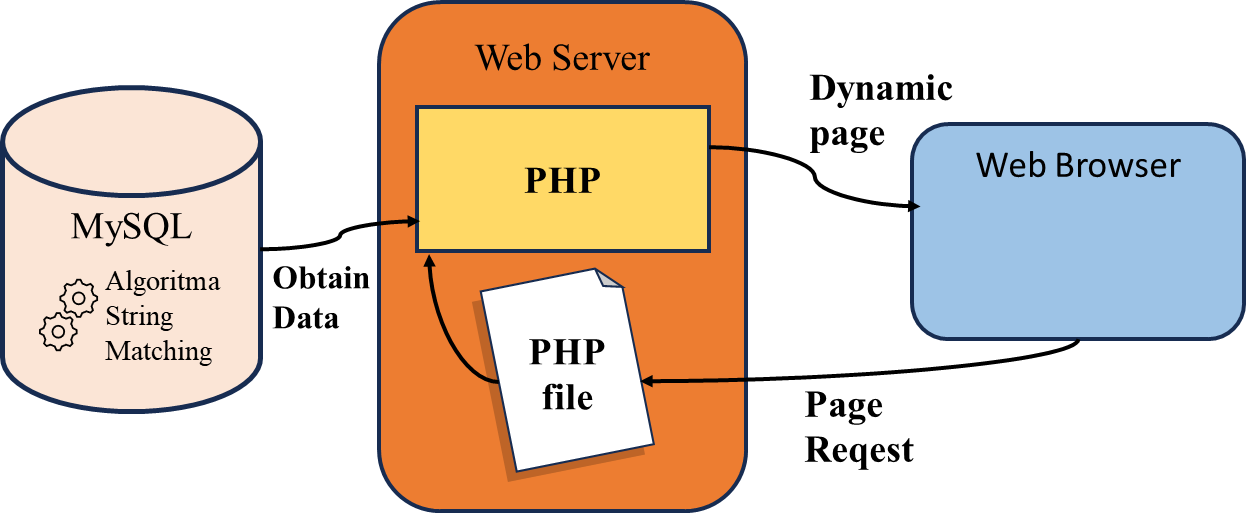
# **BAB III ANALISIS DAN PERANCANGAN SISTEM**

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, Knuth Morris Pratt, Boyer-Moore* dan *Rabin-Karp*.

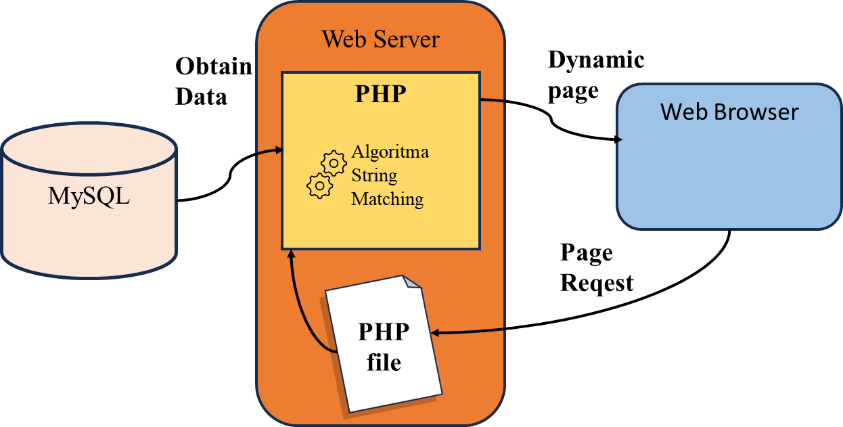
Dalam penelitian ini akan dibuat sistem yang digunakan untuk melakukan pengujian perbandingan string matching dengan menggunakan algoritma *Naive String Matching, Knuth Morris Pratt, Boyer-Moore, Rabin-Karp* dan *SQL Query Like* untuk melihat performa dari algoritma tersebut.



Gambar 3. 1 Pencarian Data Menggunakan MySQL Dengan PHP

1. **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, Knuth Morris Pratt, Boyer-Moore, 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.

* + 1. **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 Port Daveland, ID 31389-6584 | 1.831E+10 | female |
| … | … | … | … | … | … |
| 250  000 | Courtney Brakus | hahn.bernardo249999@example.com | 63453 Luz Motorway O'Reillyland, SC 79560 | 1-559-352-4733 | female |

* + 1. **Data preproceing**

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

* **Algoritma Knut Morris Pratt**: menggunakan data preprocessing yatiu menghitung terlebih dahulu *Border Function / Failure Function*. Fungsi pinggiran ini melakukan pengecekan pada *pattern* untuk menemukan kecocokan *prefix* dari *pattern* dengan *pattern* itu sendiri.
* **Algoritma Boyer More**: menggunakan Fungsi last occurrence sebagai data preprocessing untuk menentukan posisi kemunculan terakhir semua karakter pada Teks (T) di dalam Pattern (P).
* **Algoritma Rabin Karp**: menggunakan fungsi hash untuk pattern terlebih dahulu sebelum membandingkannya dengan beberapa karakter di Teks.

**3.2.3 Metode penelitian**

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

* + - 1. **Algoritma Naive String Matching**

Algoritma naive string matching memiliki kompleksitas waktu sebesar O(m\*(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 = PLANINGANDANALYASIS

Pattern = AND

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

A picture containing text, screenshot, font, line

Description automatically generated

Gambar 3. 3 Langkah 1 contoh algoritma naive string matching

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

A screenshot of a computer

Description automatically generated with low confidence

Gambar 3. 4 Langkah 2 contoh algoritma naive string matching

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

A picture containing text, font, screenshot, line

Description automatically generated

Gambar 3. 5 Langkah 3 contoh algoritma naive string matching

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

A picture containing text, font, screenshot, number

Description automatically generated

Gambar 3. 6 Langkah 4 contoh algoritma naive string matching

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

A picture containing text, font, screenshot, line

Description automatically generated

Gambar 3. 7 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.

A screenshot of a computer

Description automatically generated with low confidence

Gambar 3. 8 Langkah 6 contoh algoritma naive string matching

* + - 1. **Algoritma Knuth Morris Pratt**

Dalam algoritma ini, terdapat fungsi pinggiran KMP atau yang biasa disebut KMP *Border Function / Failure Function*. Fungsi pinggiran ini melakukan pengecekan pada *pattern* untuk menemukan kecocokan *prefix* dari *pattern* dengan *pattern* itu sendiri. b(k) merupakan fungsi pinggiran dimana ukuran terbesar dari suatu *prefix pattern* P[0..k] yang juga *suffix* dari *pattern* P[1..k], dimana k adalah posisi sebelum terjadi ketidakcocokan yaitu j-1, dimana j merupakan posisi terjadi ketidakcocokan.

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 |

Untuk menghitung fungsi pembatas, maka dimulai dari j=0, yaitu k tidak dapat didefinisikan sehingga nilai fungsi pembatas juga tidak dapat definisikan.

Fungsi pembatas selanjutnya pada j=1, sehingga 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 pembatas adalah 0 atau b(0) = 0.

Fungsi pembatas selanjutnya pada j=2, sehingga 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 pembatas adalah 1 atau b(1) = 1.

Fungsi pembatas selanjutnya pada j=3, sehingga 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 pembatas adalah 2 atau b(2) = 0.

Fungsi pembatas selanjutnya pada j=4, sehingga 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 pembatas adalah 4 atau b(4) = 1.

Fungsi pembatas selanjutnya pada j=5, sehingga 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 pembatas adalah 5 atau b(5) = 2.

Fungsi pembatas selanjutnya pada j=6, sehingga 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 pembatas 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 |

Pergeseran yang dilakukan pada algoritma Knuth-Morris-Pratt adalah berdasarkan fungsi pembatas yang sudah didefinisikan dari pattern. Persyaratannya adalah sebagai berikut:

1. Apabila terjadi ketidakcocokan karakter pada pattern P[j] yaitu P[j]!=T[i], dan k=j-1, maka nilai j menjadi b(k)
2. Apabila terjadi kecocokan karakter pada P[j] yaitu P[j]=T[i], maka nilai i menjadi i+1, dan nilai j menjadi j+1

Berikut merupakan contoh pergeseran dengan algoritma Knuth-Morris-Pratt dengan pattern P “acabaca” dan teks T “abacaabacabacababa”

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**: Pengecekan 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] = “b” dan P[j] = “c”. Karena terjadi ketidakcocokan maka pattern harus di geser sesuai dengan tabel fungsi pembatas ketika j=1 yaitu 0, sehingga mengulangi pengecekan pada i=1 dan j=0.

A picture containing text, font, line, number

Description automatically generated

Gambar 3. 9 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.

A picture containing text, font, line, number

Description automatically generated

Gambar 3. 10 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.

A picture containing text, font, line, number

Description automatically generated

Gambar 3. 11 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.

A picture containing text, line, font, screenshot

Description automatically generated

Gambar 3. 12 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.

A screenshot of a computer

Description automatically generated with low confidence

Gambar 3. 13 Langkah 5 contoh algoritma KMP

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

A picture containing text, font, line, number

Description automatically generated

Gambar 3. 14 Langkah 6 contoh algoritma KMP

* + - 1. **Algoritma Boyer Moore**

Dalam algoritma ini terdapat fungsi last occurrence untuk menentukan kemunculan terakhir Teks di Pattern, fungsi ini merupakan pre-procesing dari alogritma 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}

A group of black letters

Description automatically generated

L(a) = 4 -> Kemunculan terakhir karakter a pada P ada di indeks 4

L(b) = 3 -> Kemunculan terakhir karakter b pada P ada di indeks 3

L(c) = 5 -> Kemunculan terakhir karakter c pada P ada di indeks 5

L(d) = -1 -> Kemunculan terakhir karakter d tidak muncul

A group of numbers in a grid

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

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

A white background with black and white text

Description automatically generated

Gambar 3. 15 Langkah 1 Contoh Algoritma Boyer Moore

Pada pemeriksaan pertama terjadi mismatch yaitu di Teks indeks 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 indeks i. kemudian geser indeks i dengan rumus i=i+m-(lo+1), hasil yang diapat dari rumus tersebut adalah indeks 6.

A screenshot of a computer

Description automatically generated

Gambar 3. 16 Langkah 2 Contoh Algoritma Boyer Moore

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

A screenshot of a math test

Description automatically generated

Gambar 3. 17 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 indeks 8.

A screenshot of a computer

Description automatically generated

Gambar 3. 18 Langkah 4 Contoh Algoritma Boyer Moore

Pada pemeriksaan selanjutnya terjadi mismatch di indeks 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 hasil nya adalah indeks ke 14.

A black and white text

Description automatically generated

Gambar 3. 19 Langkah 5 Contoh Algoritma Boyer Moore

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

A screenshot of a computer

Description automatically generated

Gambar 3. 20 Langkah 6 Contoh Algoritma Boyer Moore

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

* + - 1. **Algoritma Rabin Karp**

Algoritma ini menggunakan fungsi hash sebagai pembanding antara string yang dicari (m) dengan substring pada teks (n). Apabila hash value keduanya sama maka akan dilakukan perbandingan sekali lagi terhadap karakter-karakternya. Apabila hasil keduanya tidak sama, maka substring akan bergeser ke kanan. Pergeseran dilakukan sebanyak (n-m) kali. Perhitungan nilai hash yang efisien pada saat pergeseran akan mempengaruhi performa dari 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” dimasukan kedalam fungsi hash dan menghasilkan nilai “a8”. Berikut merupakan Langkah-langkah untuk melakukan algoritma String Matching Rabin Karp.

A screenshot of a computer

Description automatically generated

Gambar 3. 21 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.

A screenshot of a computer

Description automatically generated

Gambar 3. 22 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.

A screenshot of a computer

Description automatically generated

Gambar 3. 23 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.

A screenshot of a computer

Description automatically generated

Gambar 3. 24 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.

A screenshot of a computer

Description automatically generated

Gambar 3. 25 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 “a4” kemudian hash pattern yang dicari adalah “a8” maka terjadi ketidak cocokan yang kemudian akan menggeser pencocokan teks 1 langkah kedepan.

A screenshot of a computer

Description automatically generated

Gambar 3. 26 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.

A screenshot of a computer

Description automatically generated

Gambar 3. 27 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.

A screenshot of a computer

Description automatically generated

Gambar 3. 28 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.

* + - 1. **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. 2 Tabel Customer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 Port Daveland, ID 31389-6584 | 1.831E+10 | female |
| … | … | … | … | … | … |
| 250  000 | 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

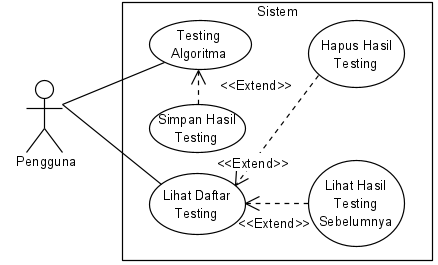
**select \* from customer where ‘%sandf%’**

1. **Perancangan UML**

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

* + 1. **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. 29 Rancangan UML Use Case Diagram

* + - 1. **Sekenario Use Case Testing Algoritma**

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

Tabel 3. 3 Sekenario use case mengelola data penyakit

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

* + - 1. **Sekenario Use Case Simpan Hasil Testing**

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

Tabel 3. 4 Sekenario Usecase Lihat Daftar Hasil Testing

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

* + - 1. **Sekenario Use Case Lihat Daftar Hasil Testing**

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

Tabel 3. 5 Sekenario use case Simpan hasil testing

|  |  |  |
| --- | --- | --- |
| **Nama** | Simpan Hasil Testing | |
| **Aktor** | Pengguna | |
| **Deskripsi** | Merupakan tombol untuk menyimpan hasil testing | |
| **Skenario** | | |
| **Aksi Aktor** | | **Reaksi Sistem** |
| Melakukan testing di halaman Testing | | Setelah Menampilkan hasil dari testing dan perbadingan algoritma *string matching.* |
| Klik tombol simpan | | Menyimpan data hasil testing sebelumnya. |
| Klik tombol Test | | Menampilkan Peringatan bahwa data berhasil disimpan. |

* + - 1. **Sekenario Use Case Hapus Hasil Testing**

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

Tabel 3. 6 Sekenario Use Case Hapus Hasil Testing

|  |  |  |
| --- | --- | --- |
| **Nama** | Hapus Hasil Testing | |
| **Aktor** | Pengguna | |
| **Deskripsi** | Merupakan tombol untuk menghapus data testing yang sudah tersimpan sebelumnya. | |
| **Skenario** | | |
| **Aksi Aktor** | | **Reaksi Sistem** |
| 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. |

* + - 1. **Sekenario Use Case Detail Hasil Testing Sebelumnya**

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

Tabel 3. 7 Sekenario Use Case Hapus Hasil Testing

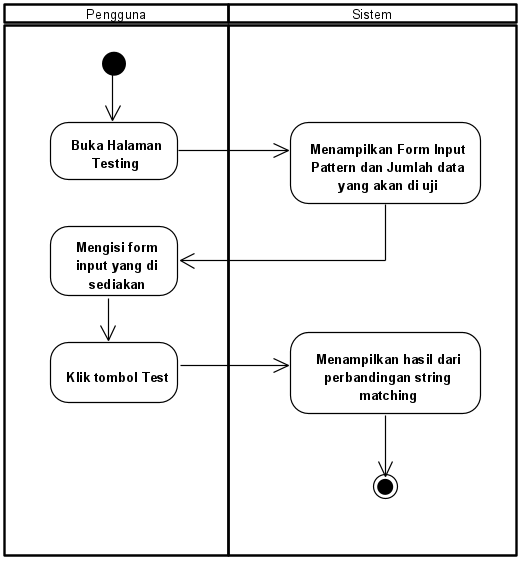
|  |  |  |
| --- | --- | --- |
| **Nama** | Detail Hasil Testing Sebelumnya | |
| **Aktor** | Pengguna | |
| **Deskripsi** | Merupakan halaman untuk melihat detail testing yang sudah di simpan sebelumnya. | |
| **Skenario** | | |
| **Aksi Aktor** | | **Reaksi Sistem** |
| Klik Menu Daftar Hasil Testing | | Muncul data daftar hasil testing sebelumnya. |
| Klik nama salasatu data yang sudah di simpan. | | Menampilkan halaman detail data hasil testing sebelumnya. |

* + 1. **Activity Diagram**

Activity Diagram menunjukan aktivitas yang dilaksanakan oleh penggua terhadap sistem perbandingan *algoritma string matching*.

1. **Activity Diagram Testing Algoritma**

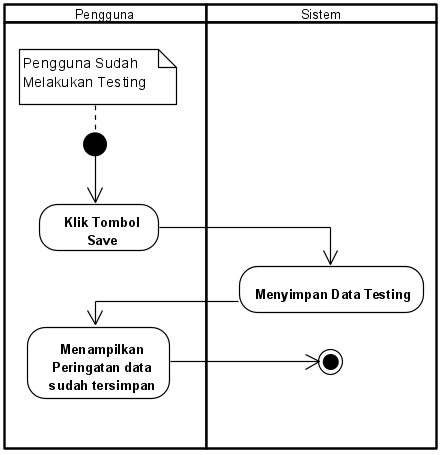
Diagram Aktivitas untuk melakukan testing algoritma yang dilaksanakan oleh pengguna pada sistem.



Gambar 3. 30 Activity Diagram Testing Algoritma

1. **Activity Diagram Simpan Hasil Testing**

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



Gambar 3. 31 Activity Diagram Simpan Hasil Testing

1. **Activity Diagram Lihat Daftar Hasil Testing**

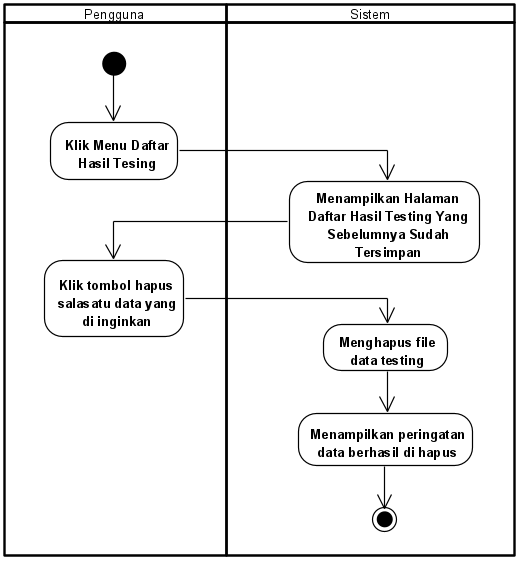
Diagram Aktivitas untuk melakukan Lihat Daftar Hasil Testing yang dilaksanakan oleh pengguna pada sistem.



Gambar 3. 32 Activity Diagram Lihat Daftar Hasil Testing

1. **Activity Diagram Hapus Hasil Testing**

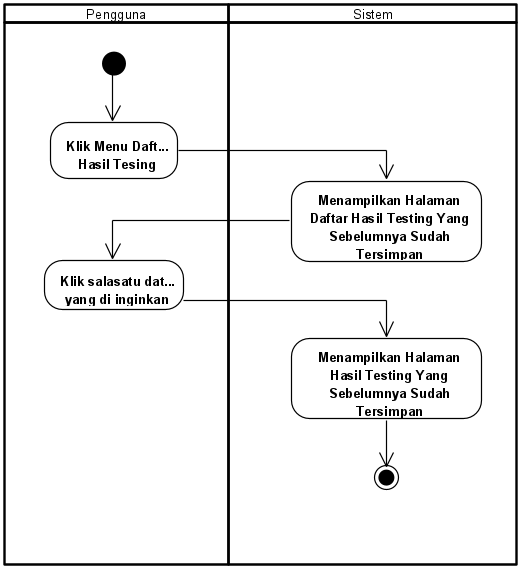
Diagram Aktivitas untuk melakukan Hapus Hasil Testing yang dilaksanakan oleh pengguna pada sistem.



Gambar 3. 33 Activity Diagram Hapus Hasil Testing

1. **Activity Diagram Detail Hasil Testing Sebelumnya**

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



Gambar 3. 34 Activity Diagram Detail Hasil Testing Sebelumnya

1. **Perancangan Antarmuka**

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

* + 1. **Rancangan Tampilan Halaman Testing**

Perancangan antar muka untuk halaman testing bisa dilihat pada gambar berikut.

A screenshot of a computer

Description automatically generated

Gambar 3. 35 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.

* + 1. **Rancangan Tampilan Halaman Daftar Hasil Testing**

Perancangan antar muka untuk halaman daftar hasil testing bisa dilihat pada gambar berikut.

A screenshot of a computer

Description automatically generated

Gambar 3. 36 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.

* + 1. **Rancangan Tampilan Halaman Detail Hasil Testing Sebelumnya**

Perancangan antar muka untuk halaman detail hasil testing Sebelumnya bisa dilihat pada gambar berikut.

A screenshot of a computer screen

Description automatically generated

Gambar 3. 37 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.

1. **Deskripsi Rencana Pengujian**

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

* + - 1. **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, Booyer 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.
  + - 1. **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.
  + - 1. **Skenario Pengujian**

Untuk pengujian algoritma string matching terdapat 3 sekenario yang akan dijalankan sebagai berikut:

* **Uji Coba Hukum Terburuk (Worst Case)**: Pengujian ini menguji performansi algoritma dengan input yang dirancang untuk skenario terburuk mereka.
* **Uji Coba 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 teks yang sangat panjang atau teks dengan berbagai pola berulang.

# **BAB IV IMPLEMENTASI DAN PENGUJIAN SISTEM**

* 1. **Batasan Implementasi**

Dalam implementasi perangkat lunak dibutuhkan pendukung perangkat lunak dan perangkat keras yang baik agar senantiasa mendukung saat penulisan program. Perangkat penyusun pada saat melakukan implementasi adalah sebagai berikut:

* + 1. **Komputer perangkat keras**

Adapun komputer yang digunakan adalah dengan spesifikasi 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 |

* + 1. **Perangkat lunak**

Dalam menerapkan rancangan yang telah dibuat, dibutuhkan beberapa *software* untuk menggunakan program aplikasi yaitu :

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 |

* 1. **Implementasi Antarmuka**

Tahap implementasi sistem merupakan tahap perancangan berdasarkan hasil analisis ke dalam suatu Bahasa pemograman tertentu serta penerapan perangkat lunak yang dibangun pada lingkungan yang sebenarnya. Sistem Perbandingan String Matching ini menggunakan sistem Berbasis Web.

1. **Halaman Testing**

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

A screenshot of a computer

Description automatically generated with medium confidence

Gambar 4. 1 Halaman Testing Tabel 10.000-100.000 Data

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

A screenshot of a computer

Description automatically generated with low confidence

Gambar 4. 2 Halaman Testing Tabel 100.000-250.000 Data

A screenshot of a graph

Description automatically generated with low confidence

Gambar 4. 3 Halaman Testing Diagram Garis Kecepatan Waktu dan Penggunaan Memory

* + 1. **Halaman Daftar Hasil Testing**

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

**A screenshot of a computer

Description automatically generated with medium confidence**

Gambar 4. 4 Halaman Daftar Hasil Testing

* + 1. **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.

A screenshot of a computer

Description automatically generated with medium confidence

Gambar 4. 5 Halaman Detail Testing Sebelmunya - Tabel 10.000-100.000 Data

A picture containing text, screenshot, number, design

Description automatically generated

Gambar 4. 6 Halaman Detail Testing Sebelmunya - Tabel 100.000-250.000 Data

A screenshot of a graph

Description automatically generated with low confidence

Gambar 4. 7 Halaman Testing Sebelumnya - Diagram Garis Kecepatan Waktu dan Penggunaan Memory

* 1. **Hasil Penelitian**

…….

* 1. **Pengujian Sistem**

Tahapan pengujian adalah tahapan selanjutnya setelah tahap implementasi selesai. Tahapan ini fungsinya untuk mengetahui apakah sistem/aplikasi yang dibangun sudah sesuai dengan tujuan perancangan. Pengujian sistem adalah rangkaian percobaan pada setiap komponen dalam suatu sistem/aplikasi. Tujuan dari diujinya suatu aplikasi adalah agar ditemukannya kesalahan maupun kekurangan yang ada pada aplikasi. Metode pengujian *blackbox* dan pengujian beta merupakan pengujian yang dilakukan pada penelitian ini.

1. **Pengujian Blackbox**

Pengujian *blackbox* adalah metode uji yang fokusnya kepada persyaratan fungsi dari suatu aplikasi. Pengujian dilakukan dengan melakukan serangkaian proses input dan kemudian melihat hasil yang didapatkan dari proses input tersebut.

Hasil pengujian *blackbox* dari Sistem Perbandingan String Matching ini dapat dilihat sebagai berikut.

* + 1. Halaman Testing

Berdasarkan pengujian dengan metode *blackbox* yang dilakukan terhadap halaman testing, maka diperoleh hasil 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 |

* + 1. Halaman Daftar Testing

Berdasarkan pengujian dengan metode *blackbox* yang dilakukan terhadap halaman daftar testing, maka diperoleh hasil 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 |

* + 1. Halaman Detail Testing Sebelumnya

Berdasarkan pengujian dengan metode *blackbox* yang dilakukan terhadap halaman daftar testing, maka diperoleh hasil sebagai berikut.

Tabel 4. 5 Halaman Detail Testing Sebelumnya

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

1. **Pengujian Beta**

Pengujian beta yang dilaksanakan dalam penelitian ini adalah pengguna mencoba aplikasi, setelah itu pengguna mengisi kuesioner yang telah disediakan. Kuesioner ini dilakukan untuk mengelola data yang sudah diperoleh untuk selanjutnya dijadikannya hasil penelitian dan bisa diambil kesimpulan.

# **BAB V PENUTUP**

1. **Kesimpulan**

Berdasarkan pengujian yang dilakukan, berikut adalah urutan dari algoritma string matching dari yang paling efisien hingga yang paling kurang efisien:

1. **Algoritma Boyer Moore**: Algoritma ini memperlihatkan performansi yang paling unggul. Boyer Moore menunjukkan waktu komputasi yang paling cepat dan efisiensi memori yang baik. Hal ini mungkin disebabkan oleh cara kerja algoritma tersebut, yang tidak perlu memeriksa setiap karakter dalam teks jika ada kecocokan sebagian yang gagal.
2. **Algoritma Naive String Matching**: Meski merupakan pendekatan yang paling dasar, algoritma ini menunjukkan performansi yang mengesankan, menjadi yang kedua tercepat dalam pengujian. Ini menunjukkan bahwa untuk kasus-kasus tertentu dengan input tertentu, pendekatan yang lebih sederhana mungkin cukup efisien.
3. **Algoritma Knuth Morrin Pratt (KMP)**: Algoritma KMP menempati posisi ketiga dalam performansi. Meskipun algoritma ini memiliki keuntungan dalam beberapa kasus edge-case karena kemampuan lompatannya, ia tidak melampaui Boyer Moore atau Naive dalam pengujian ini.
4. **SQL Query Like**: SQL Query Like, meskipun sangat berguna dalam pencarian berbasis database, tidak seefisien algoritma lain dalam konteks pencocokan string murni dan menempati posisi ke empat.
5. **Algoritma Rabin Karp**: Meskipun Rabin Karp merupakan algoritma yang efisien dengan pendekatan hashing, ia menunjukkan performa paling lambat dalam pengujian ini, membuatnya berada di posisi terakhir.

Dengan demikian, berdasarkan hasil pengujian, untuk keperluan pencocokan string murni, Algoritma Boyer Moore menjadi pilihan optimal dengan Algoritma Naive sebagai alternatif yang memadai. Namun, perlu diingat bahwa keefektifan algoritma sangat bergantung pada karakteristik input dan persyaratan kasus penggunaan tertentu.

1. **Saran**

Berdasarkan hasil pengujian yang telah diterapkan, berikut merupakan saran yang dapat diberikan:

1. **Menganalisis Karakteristik Data**: Dalam rangka menentukan algoritma string matching yang paling efektif, krusial untuk mendalami karakteristik data yang akan diproses. Sebagai contoh, Algoritma Boyer Moore mendemonstrasikan efisiensi tinggi pada teks dengan panjang yang signifikan, sementara Algoritma Naive dapat menjadi solusi efisien untuk teks dengan panjang yang relatif lebih pendek.
2. **Mempertimbangkan Konteks Penggunaan**: Dalam konteks yang melibatkan manajemen database SQL dan hanya membutuhkan operasi seperti pencarian wildcard, SQL Query Like dapat diprioritaskan sebagai alternatif yang lebih sesuai dibandingkan dengan implementasi algoritma string matching lainnya.
3. **Penerapan Benchmarking**: Dalam setiap situasi, selalu dianjurkan untuk melaksanakan pengujian dan benchmarking independen. Algoritma yang tampak "terbaik" dalam suatu konteks dapat berbeda efektivitasnya pada konteks lain, berdasarkan spesifikasi dan kebutuhan yang unik pada setiap kasusnya.
4. **Memanfaatkan Struktur Algoritma**: Beberapa algoritma, seperti Knuth-Morris-Pratt dan Rabin-Karp, memiliki struktur yang unik dan telah dirancang untuk mengatasi beberapa skenario terburuk. Memahami struktur dan fitur ini dapat menjadi faktor penentu dalam memilih algoritma yang paling cocok.
5. **Evaluasi antara Kecepatan dan Akurasi**: Dalam setiap kasus, penyeimbangan antara kecepatan dan akurasi memegang peranan penting. Dalam beberapa situasi, memilih algoritma yang sedikit lebih lambat dapat menjadi keputusan yang lebih baik jika hal tersebut dapat memberikan peningkatan signifikan dalam hal akurasi atau hasil yang konsisten.

Secara umum, tidak ada algoritma yang "terbaik" dalam semua skenario. Pemilihan algoritma yang efisien sering kali bergantung pada berbagai faktor, dan algoritma tersebut harus dianalisis dan direview secara periodik untuk memastikan bahwa mereka tetap optimal dalam menghadapi perkembangan dan perubahan data dan kebutuhan.

Abstrak belum jadi

Biasanya, dalam pencocokan string, metode yang umum digunakan adalah SQL Query Like[2%5E]. Namun, terdapat banyak algoritma pencocokan string lainnya yang mungkin memiliki performa dan efisiensi yang berbeda tergantung pada konteks dan kebutuhan spesifiknya. Dalam Penelitian ini dibuat untuk melihat performansi lima algoritma string matching, termasuk Algoritma Boyer Moore, Algoritma Naive String Matching, Algoritma Knuth Morrin Pratt (KMP), SQL Query Like, dan Algoritma Rabin Karp. Melalui rangkaian pengujian yang komprehensif, Boyer Moore menunjukkan efisiensi tinggi dan konsisten, menjadikannya pilihan utama untuk kegiatan string matching. Di sisi lain, Algoritma Naive String Matching, sederhana namun efisien, menempati posisi kedua dalam urutan, menegaskan asumsi bahwa pendekatan sederhana mungkin cukup dalam sejumlah kasus dengan input tertentu. Ketiga adalah Algoritma KMP, walaupun dengan kelebihannya dalam beberapa kasus edge-case, ia tidak mampu melampaui dua algoritma di atasnya. SQL Query Like, walaupun bermanfaat dalam kasus pemilihan spesifik, menunjukkan efisiensi yang lebih rendah dalam konteks pencocokan string. Pada posisi terakhir ada Algoritma Rabin Karp, yang meski memiliki pendekatan unik menggunakan hashing, tetap menunjukkan performa terendah dalam pengujian ini. Penelitian ini memberikan pandangan berharga bagi penentuan algoritma string matching yang tepat berdasarkan efisiensi dan konteks penggunaannya.

# **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 SIMALIRITY 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).