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

1st Isep Lutpi Nur   
*Faktultas Teknik*  
*Universitas Sangga Buana YPKP*Bandung, Indonesia  
iseplutpinur7@gmail.com

line 1: 4th Given Name Surname  
line 2: *dept. name of organization*  
*(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID 2nd Gunawansyah  
*Faktultas Teknik*  
*Universitas Sangga Buana YPKP*Bandung, Indonesia  
line 5: email address or ORCID

line 1: 5th Given Name Surname  
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCIDline 1: 3rd Given Name Surname  
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

line 1: 6th Given Name Surname  
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

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 Booyer More lebih cepat dibanding algoritma yang lain dengan selisih 161,8ms dengan SQL Query Like atau 45,8% lebih cepat berdasarkan pengujian performansi dengan sekenario 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.

Keywords—String Matching, Boyer-Moore, Naive String Matching, Knuth-Morris-Pratt, Rabin-Karp Algorithm

# Pendahuluan

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. 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 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.

# Metode

## 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.

## 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 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

## 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 pinggiran dimulai dari j=0, yaitu ketika k tidak dapat didefinisikan maka nilai fungsi pinggiran juga tidak dapat definisikan.

Selanjutnya fungsi pinggiran 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 pinggiran adalah 0 atau b(0) = 0.

Selanjutnya fungsi pinggiran 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 pinggiran adalah 1 atau b(1) = 1.

Selanjutnya fungsi pinggiran 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 pinggiran adalah 2 atau b(2) = 0.

Selanjutnya fungsi pinggiran 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 pinggiran adalah 4 atau b(4) = 1.

Selanjutnya fungsi pinggiran 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 pinggiran adalah 5 atau b(5) = 2.

Selanjutnya fungsi pinggiran 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 pinggiran 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

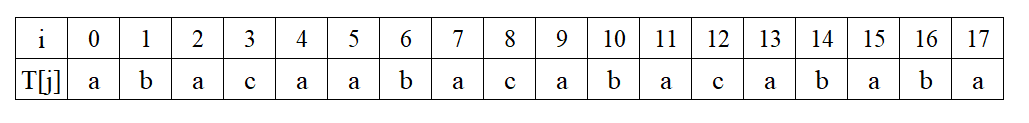
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| P[j] | a | c | a | b | a | c | a |
| K | - | 0 | 1 | 2 | 3 | 4 | 5 |
| b(k) | - | 0 | 1 | 0 | 1 | 2 | 3 |

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]!=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 TabelTeks "abacaabacabacababa" Algoritma KMP



* **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] = “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

## Algoritma Booyer 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

Gambar 3. 3 Variasi karakter Booyer moore

L(a) = 4 -> LO karakter a pada P ada di index 4

L(b) = 3 -> LO karakter b pada P ada di index 3

L(c) = 5 -> LO karakter c pada P ada di index 5

L(d) = -1 -> LO karakter d tidak muncul

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

A group of numbers in a grid

Description automatically generated

Gambar 3. 4 Tabel Last Occurence

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 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 diapat dari rumus tersebut adalah index 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 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.

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 index 8.

A screenshot of a computer

Description automatically generated

Gambar 3. 18 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 hasil nya adalah index 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 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.

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 index ke 10 sebagai hasil dari pencarian.

## 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” 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. 23 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. 24 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. 25 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. 26 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. 27 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. 28 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. 29 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. 30 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.

## 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 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%’**

# Implementasi dan Pengujian Sistem

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.

## Implementasi

Implementasi sistem pengujian string matching dengan kata kunci Aditya Degnan menggunakan 250.000 data.

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 graph

Description automatically generated with low confidence

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.

## Pengujian Sistem

Untuk pengujian algoritma string matching terdapat 2 skenario sebagai berikut:

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

Pada sekenario 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 sekenario 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 sekenario kasus rata-rata (Kecepatan)

Dalam gambar diagram garis diatas bisa diketahui bahwa algoritma Booyer Moore 64,6% lebih cepat disbanding algoritma Rabin Karp dengan selisih 349,6 Milidetik lebih cepat.

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

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

1. Pengujian Dengan Karakteristik Teks Khusus

Untuk pengujian dengan karakteristik teks khusus ini menggkunakan 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.

1. Pengujian Dengan Banyak Pengguna

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 sebelumny. Pengujian ini dilakukan 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 utuk 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.

1. 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 sebelumny. 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 mentode boyer moore adalah 854ms dan pada pengujian skenario sebelumhya dengan mengguakan 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.

# Kesimpulan dan Saran

## Kesimpulan

Berdasarkan hasil penelitian yang dilakukan maka dapat diambil kesimpulan bahwa algoritma Booyer More lebih cepat dibanding algoritma yang lain dengan selisih 104ms dengan SQL Query Like atau 39,25% lebih cepat berdasarkan pengujian performansi dengan sekenario 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.

## 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. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because, in an MSW document, this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.