

**LAPORAN TUGAS BESAR III**  
**IF2211 STRATEGI ALGORITMA**  
**PENERAPAN STRING MATCHING DAN REGULAR EXPRESSION**  
**DALAM PEMBUATAN CHATGPT SEDERHANA**



Disusun oleh:

13521043	Nigel Sahl
13521086	Ariel Jovananda
13521149	Rava Maulana

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**BANDUNG 2023**

## **Daftar Isi**

<b>Daftar Isi</b>	<b>2</b>
<b>BAB I</b>	<b>3</b>
<b>DESKRIPSI TUGAS</b>	<b>3</b>
1.1. Deskripsi Tugas	3
1.2. Fitur-Fitur Aplikasi	3
1.3. Spesifikasi Program	4
<b>BAB II</b>	<b>5</b>
<b>LANDASAN TEORI</b>	<b>5</b>
2.1. Algoritma Knuth-Morris-Pratt (KMP)	5
2.2. Algoritma Boyer-Moore (BM)	7
2.3. Regular Expression	9
2.4. Aplikasi Web	9
<b>BAB III</b>	<b>10</b>
<b>APLIKASI ALGORITMA</b>	<b>10</b>
3.1. Pemecahan Masalah	10
3.2. Arsitektur Aplikasi Web yang Dibangun	10
3.3. Fitur Fungsional	10
<b>BAB IV</b>	<b>12</b>
<b>Analisis Pemecahan Masalah</b>	<b>12</b>
4.1. Spesifikasi Teknis Program	12
4.2. Tata Cara Penggunaan Program	19
4.3. Hasil Pengujian dan Analisis Hasil Pengujian	19
<b>BAB V</b>	<b>22</b>
<b>KESIMPULAN DAN SARAN</b>	<b>22</b>
5.1. Kesimpulan	22
5.2. Saran	22
5.3. Komentar dan Refleksi	22
<b>Referensi</b>	<b>23</b>
<b>Lampiran</b>	<b>24</b>

# **BAB I**

## **DESKRIPSI TUGAS**

### **1.1. Deskripsi Tugas**

Dalam tugas besar ini, kami membangun sebuah aplikasi ChatGPT sederhana dengan mengaplikasikan pendekatan QA yang paling sederhana tersebut. Pencarian pertanyaan yang paling mirip dengan pertanyaan yang diberikan pengguna dilakukan dengan algoritma pencocokan string Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Regex digunakan untuk menentukan format dari pertanyaan (akan dijelaskan lebih lanjut pada bagian fitur aplikasi). Jika tidak ada satupun pertanyaan pada database yang exact match dengan pertanyaan pengguna melalui algoritma KMP ataupun BM, maka gunakan pertanyaan termirip dengan kesamaan setidaknya 90% Apabila tidak ada pertanyaan yang kemiripannya di atas 90%, maka chatbot akan memberikan maksimum 3 pilihan pertanyaan yang paling mirip untuk dipilih oleh pengguna.

### **1.2. Fitur-Fitur Aplikasi**

ChatGPT sederhana yang kami buat dapat melakukan beberapa fitur / klasifikasi query seperti berikut:

a. Fitur pertanyaan teks (didapat dari database)

Mencocokkan pertanyaan dari input pengguna ke pertanyaan di database menggunakan algoritma KMP atau BM.

b. Fitur kalkulator

Pengguna memasukkan input query berupa persamaan matematika. Contohnya adalah  $2*5$  atau  $5+9*(2+4)$ . Operasi cukup Tambah, kurang, kali, bagi, pangkat, kurung.

c. Fitur tanggal

Pengguna memasukkan input berupa tanggal, lalu chatbot akan merespon dengan hari apa di tanggal tersebut. Contohnya adalah 25/08/2023 maka chatbot akan menjawab dengan hari senin.

d. Tambah pertanyaan dan jawaban ke database

Pengguna dapat menambahkan pertanyaan dan jawabannya sendiri ke database dengan query contoh “Tambahkan pertanyaan xxx dengan jawaban yyy”. Menggunakan algoritma string matching untuk mencari tahu apakah pertanyaan sudah ada. Apabila sudah, maka jawaban akan diperbaharui.

- e. Hapus pertanyaan dari database

Pengguna dapat menghapus sebuah pertanyaan dari database dengan query contoh “Hapus pertanyaan xxx”. Menggunakan string algoritma string matching untuk mencari pertanyaan xxx tersebut pada database.

Klasifikasi dilakukan menggunakan regex dan terklasifikasi layaknya bahasa sehari - hari. Algoritma string matching KMP dan BM digunakan untuk klasifikasi query teks. Tersedia toggle untuk memilih algoritma KMP atau BM. Semua pemrosesan respons dilakukan pada sisi backend. Jika ada pertanyaan yang sesuai dengan fitur, maka tampilkan saja “Pertanyaan tidak dapat diproses”. Berikut adalah beberapa contoh ilustrasi sederhana untuk tiap pertanyaannya. (Note: Tidak wajib mengikuti ilustrasi ini, tampilan disamakan dengan chatGPT juga boleh)

Layaknya ChatGPT, di sebelah kiri disediakan history dari hasil pertanyaan. History menampilkan 5-10 pertanyaan terbaru di toolbar kiri. Perhatikan bahwa sistem history disini disamakan dengan chatGPT, sehingga satu history yang diklik menyimpan seluruh IF2211 Strategi Algoritma - Tugas Besar 3 7 pertanyaan pada sesi itu. Apabila history diclick, maka akan merestore seluruh pertanyaan dan jawaban di halaman utama.

### **1.3. Spesifikasi Program**

1. Aplikasi berbasis website dengan pembagian Frontend dan Backend yang jelas.
2. Implementasi Backend wajib menggunakan Node.js / Golang, sedangkan Frontend menggunakan Next.js.
3. Penyimpanan data wajib menggunakan basis data MongoDB.
4. Algoritma pencocokan string (KMP dan Boyer-Moore) dan Regex diimplementasikan pada sisi Backend aplikasi.
5. Informasi yang disimpan pada basis data:
  - a. Tabel pasangan pertanyaan dan Jawaban
  - b. Tabel history
6. Skema basis data dibebaskan asalkan mencakup setidaknya kedua informasi di atas.
7. Proses string matching pada tugas ini Tidak case sensitive.
8. Pencocokan yang dilakukan adalah dalam satu kesatuan string pertanyaan utuh (misal “Apa ibukota Filipina?”), bukan kata per kata (“apa”, “ibukota”, “Filipina”).

## BAB II

### LANDASAN TEORI

Pencocokan *string* atau *pattern matching* adalah teknik untuk mencocokkan pola atau mencari pola tertentu dalam suatu teks atau data. Teknik ini umumnya digunakan dalam pemrograman dan pengolahan bahasa alami. Dalam pemrograman, *pattern matching* dapat digunakan untuk melakukan pencarian dan penggantian teks dalam sebuah file atau database. Dalam pengolahan bahasa alami, *pattern matching* dapat digunakan untuk mencari kata-kata yang mirip atau memiliki arti yang sama dalam suatu teks.

Pattern matching juga merupakan teknik yang digunakan dalam algoritma pencocokan pola seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Kedua algoritma ini sangat efisien dalam melakukan pencocokan pola karena mereka menghindari pengecekan ulang pada teks yang sudah diproses.

KMP adalah sebuah algoritma pencocokan pola yang merujuk pada penggunaan tabel pembangunan pola (pattern building table) untuk mengurangi jumlah pengecekan ulang pada teks yang sudah diolah. Algoritma ini bekerja dengan membandingkan pola yang dicari dengan teks dan mencari kata yang cocok. Algoritma KMP sangat efisien dalam mengurangi waktu yang dibutuhkan untuk mencari pola dalam teks.

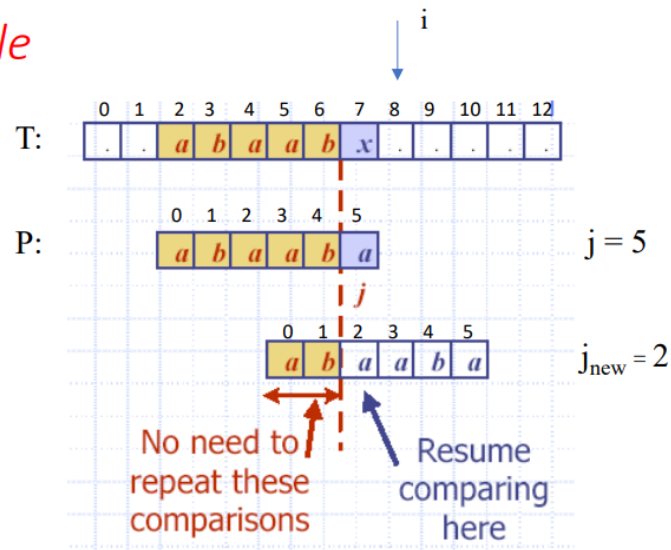
Sementara itu, BM adalah algoritma pencocokan pola yang mencari pola dengan cara menggeser pola ke arah kanan dan membandingkannya dengan teks. Algoritma ini sangat efektif dalam mencari pola dalam teks karena meminimalkan jumlah pengecekan ulang pada teks yang sudah diproses.

Kedua algoritma ini digunakan secara luas dalam aplikasi pencocokan pola yang memerlukan kecepatan dan efisiensi dalam pencarian pola dalam teks.

#### 2.1. Algoritma Knuth-Morris-Pratt (KMP)

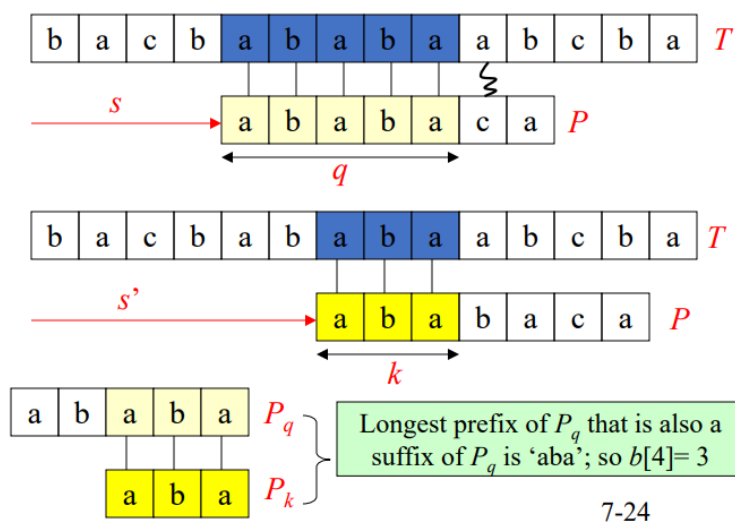
Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma efisien untuk pencocokan pola pada sebuah teks. Algoritma ini bekerja dengan melakukan pencocokan pola dari kiri ke kanan pada teks sambil menghindari pengecekan ulang secara berulang-ulang pada teks yang sudah diproses. Algoritma ini efisien dalam melakukan pencocokan pola karena menghindari pengecekan ulang pada teks yang sudah diproses. Algoritma Knuth-Morris-Pratt (KMP) mencari pola dalam teks secara berurutan dari kiri ke kanan (seperti algoritma brute force). Namun, algoritma ini menggeser pola dengan lebih cerdas daripada algoritma brute force.

## Example



Gambar 2.1 Contoh Penerapan Algoritma KMP

Cara kerja algoritma KMP adalah dengan membangun sebuah tabel pembangunan pola (pattern building table) yang digunakan untuk menentukan jumlah karakter yang harus dilewati saat melakukan pencocokan ulang pada teks jika terjadi kegagalan pencocokan pada karakter tertentu. Dengan cara ini, algoritma KMP dapat mencocokkan sebuah pola pada sebuah teks dengan waktu yang lebih efisien dibandingkan dengan algoritma pencocokan pola lainnya.



Gambar 2.2 Contoh Pergeseran dalam KMP

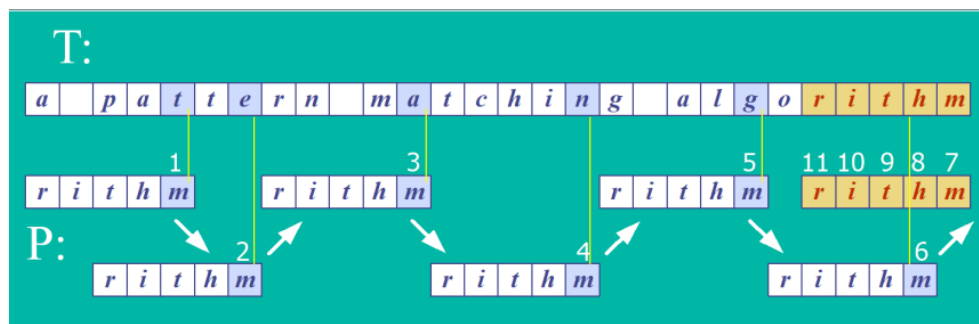
Tabel pembangunan pola pada algoritma KMP dibentuk dengan menghitung prefix fungsi dari pola yang akan dicocokkan. Prefix fungsi merupakan sebuah fungsi yang mengembalikan nilai yang menunjukkan panjang dari prefiks terpanjang dari pola yang juga merupakan suffiks dari pola tersebut. Dengan menggunakan nilai prefix fungsi, tabel pembangunan pola dapat dibentuk dengan menghitung nilai-nilai prefix fungsi untuk setiap karakter pada pola.

## 2.2. Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan pola yang digunakan untuk mencari pola tertentu dalam sebuah teks. Algoritma ini merupakan salah satu algoritma pencocokan pola yang paling efisien dalam hal waktu eksekusi.

Cara kerja algoritma BM adalah dengan mencocokkan pola yang dicari dengan teks dari arah kanan ke kiri. Algoritma ini menerapkan dua heuristik untuk menghindari pengecekan ulang pada teks yang sudah diproses, yaitu heuristik karakter yang buruk (bad character heuristic) dan heuristik sufiks yang baik (good suffix heuristic).

### *Boyer-Moore Example (1)*



Heuristik karakter yang buruk memungkinkan algoritma BM untuk menghindari pengecekan ulang pada teks dengan menentukan karakter pada pola yang tidak cocok dengan karakter pada teks saat pencocokan dilakukan. Heuristik sufiks yang baik memungkinkan algoritma BM untuk menghindari pengecekan ulang pada teks dengan menggunakan sufiks pada pola yang sudah cocok dengan teks.

Algoritma BM sangat efisien dalam melakukan pencocokan pola pada teks karena menghindari pengecekan ulang pada teks yang sudah diproses dan menggunakan dua heuristik untuk mengoptimalkan waktu eksekusi.

The Boyer-Moore pattern matching algorithm is based on two techniques.

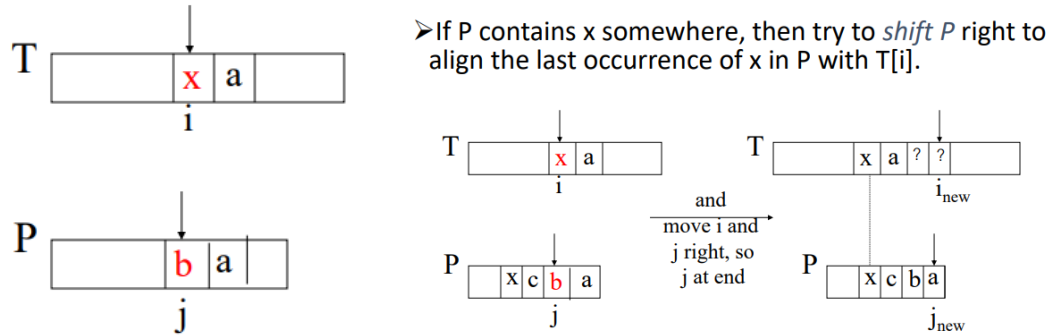
1. The looking-glass technique

find P in T by moving backwards through P, starting at its end

## 2. The character-jump technique

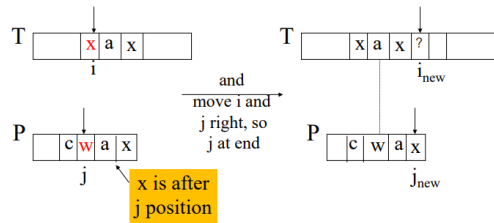
- when a mismatch occurs at  $T[i] == x$
- the character in pattern  $P[j]$  is not the same as  $T[i]$

### Case 1



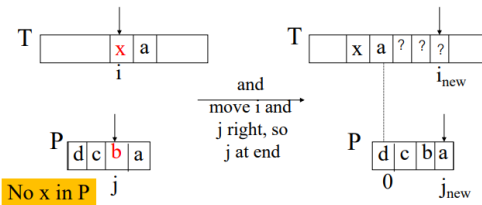
### Case 2

- If  $P$  contains  $x$  somewhere, but a shift right to the last occurrence is *not* possible, then *shift  $P$*  right by 1 character to  $T[i+1]$ .



### Case 3

- If cases 1 and 2 do not apply, then *shift  $P$*  to align  $P[0]$  with  $T[i+1]$ .



Gambar 2.2 Tiga Kemungkinan Kasus dalam BM



### **2.3. Regular Expression**

Regular expression (regex) adalah sebuah pola atau aturan yang digunakan untuk mencari atau mengganti teks tertentu pada sebuah string. Dalam regex, kita dapat menggunakan simbol atau karakter khusus untuk mempermudah proses pencarian, seperti karakter wildcard (\*), tanda kurung (), tanda tanya (?), tanda tambah (+), dan lain sebagainya. Regex sangat berguna dalam pengolahan teks massal, seperti pada aplikasi web, pemrosesan data, atau scripting. Dengan regex, kita dapat mengekstraksi data yang spesifik dari sebuah teks dan melakukan manipulasi atau transformasi teks dengan cepat dan efisien. Regex ini adalah yang nanti akan digunakan untuk mengklasifikasi tipe pertanyaan apa yang sesuai dengan kalimat utuh yang diberikan.

### **2.4. Aplikasi Web**

Aplikasi berbasis web adalah program komputer yang dirancang untuk diakses melalui internet menggunakan browser web. Aplikasi ini dibangun dengan teknologi web seperti HTML, CSS, dan JavaScript, serta menggunakan server web dan database untuk menyimpan dan mengelola data. Contoh dari aplikasi berbasis web termasuk media sosial, e-commerce, aplikasi email, dan banyak lagi.

Aplikasi berbasis web memungkinkan pengguna untuk mengakses informasi dan berinteraksi dengan layanan melalui internet, tanpa harus mengunduh atau menginstal program pada perangkat mereka. Hal ini membuat aplikasi web menjadi pilihan yang populer untuk bisnis dan organisasi yang ingin menyediakan layanan online yang mudah diakses oleh pengguna dari berbagai tempat. Namun, keamanan dan skalabilitas adalah masalah yang perlu diperhatikan dalam pengembangan aplikasi berbasis web.

## BAB III

### APLIKASI ALGORITMA

#### 3.1. Pemecahan Masalah

Permasalahan yang ada yaitu pengecekan pertanyaan yang diberikan pengguna dalam sebuah database berisi pasangan pertanyaan jawaban. Pertama, kami membuat algoritma pattern matching seperti yang dijelaskan dalam perkuliahan yaitu pencocokan sebuah substring dengan string. Selanjutnya, desain sebuah database dengan MongoDB. Database yang kami buat berisi sebuah array objek dengan elemen pertanyaan dan jawaban.

Proses pencarian pertanyaan dalam database dilakukan dengan

Regex digunakan untuk mengklasifikasi masukkan yang diberikan. Karena dalam satu masukkan bisa ada beberapa tipe *query*, regex dibutuhkan agar masukan tersebut dapat dicacah dan mengelompokkan substring-substring menjadi tipe-tipe *query* yang berbeda

#### 3.2. Arsitektur Aplikasi Web yang Dibangun

Aplikasi web yang dibangun memiliki tiga komponen utama yaitu *frontend*, *backend*, dan basis data. Komponen *frontend* dibangun menggunakan bahasa pemrograman Typescript dengan *framework* Next.JS. Komponen *backend* dibangun menggunakan bahasa pemrograman Node.JS dengan *framework* Express. Komponen basis data yang digunakan adalah MongoDB yang merupakan jenis basis data non-relasional.

Komponen *frontend* merupakan komponen yang digunakan untuk menampilkan aplikasi pada perangkat *web browser* yang dimiliki oleh *client*. Komponen *backend* merupakan sebuah *server* yang digunakan untuk melakukan operasi algoritma dan menghubungkan *client* dengan basis data. Komunikasi antara *frontend* dan *backend* menggunakan protokol HTTP dengan *frontend* sebagai pengirim *request* dan *backend* sebagai pemberi *response*.

#### 3.3. Fitur Fungsional

Terdapat tiga fitur fungsional dari aplikasi yaitu fitur pemilihan algoritma, fitur riwayat percakapan, dan fitur pemrosesan pertanyaan. Fitur pemilihan algoritma berupa sebuah *toggle button* yang bertuliskan “Knuth-Morris-Pratt” dan “Boyer-Moore” yang dapat digunakan untuk memilih algoritma *string matching* yang akan digunakan untuk memproses pertanyaan. Fitur riwayat percakapan berupa tampilan daftar lima riwayat percakapan terbaru yang dapat ditekan untuk menampilkan sekaligus mengakses riwayat percakapan sebelumnya. Fitur pemrosesan pertanyaan merupakan fitur utama aplikasi yang dapat mengklasifikasikan pertanyaan menjadi lima kategori yaitu penambahan pertanyaan,

penghapusan pertanyaan, memproses tanggal, memproses perhitungan matematis, dan melakukan *string matching* pada basis data pertanyaan.

## BAB IV

### Analisis Pemecahan Masalah

#### 4.1. Spesifikasi Teknis Program

##### 4.1.1. Basis Data

Program menggunakan basis data non-relasional yang terdiri dari dua *collection* dengan skema sebagai berikut

##### 1. Questions

```
question : String
answer   : String
```

##### 2. History

```
chatList : { question: String, response: String } []
dateCreated : Date
```

##### 4.1.2. Algoritma

Dalam algoritma yang kami buat, terdapat beberapa fungsi untuk menyelesaikan persoalan yaitu

##### 1. addSimilarity

Fungsi untuk menambahkan index dan distance (ukuran jarak ketidaksamaan antara pola string dengan elemen dalam database) ke ListOfSimilarity dengan penambahan yang terurut dari distance terkecil ke terbesar dan terbatas dengan 3 elemen list. Fungsi ini yaitu jika ListOfSimilarity kosong, maka akan langsung ditambahkan index dan dist ke ListOfSimilarity. Jika ListOfSimilarity tidak kosong dan belum penuh, maka akan ditambahkan index dan dist ke ListOfSimilarity dengan urutan ascending

##### 2. computeBorder

Fungsi untuk menghitung border dari pattern yang digunakan untuk algoritma KMP. Border adalah panjang suffix terpanjang dari prefix pattern yang merupakan prefix pattern juga.

- a. Inisiasi array border,  $\text{border}[0] = 0$ ,  $i = 1$ ,  $j = 0$
- b. Iterasi  $i$  dari 1 sampai  $n-1$  ( $n$  adalah panjang pattern)
- c. Jika  $\text{pattern}[i] = \text{pattern}[j]$  maka  $\text{border}[i] = j+1$ ,  $i++$ ,  $j++$
- d. Jika  $\text{pattern}[i] \neq \text{pattern}[j]$  dan  $j > 0$  maka  $j = \text{border}[j-1]$

- e. Jika `pattern[i] != pattern[j]` dan `j = 0` maka `border[i] = 0`, `i++`
- f. Kembalikan `border`

### 3. kmpMatch

Fungsi untuk menerapkan algoritma KMP dengan memproses terlebih dahulu array `border` untuk `pattern` dengan fungsi `computeBorder` untuk mempercepat pencarian.

- a. Inisialisasi `n` dan `m` dengan panjang `source` dan `pattern`, `border` dengan `border` function dari `pattern`, `i` dengan 0, dan `j` dengan 0.
- b. Iterasi `i` dari 0 sampai `n-1`.
- c. Jika `pattern[j]` sama dengan `source[i]`, maka:
- d. Jika `j` sama dengan `m-1`, maka kembalikan `i-m+1`. Jika tidak, maka tambahkan 1 ke `i` dan `j`.
- e. Jika `j > 0`, maka `j` sama dengan `border[j-1]`. Jika tidak, maka tambahkan 1 ke `i`.
- f. Jika iterasi sudah selesai, maka kembalikan -1 yang berarti tidak ada `pattern` dalam `source`.

### 4. buildLast

Fungsi untuk membuat *last occurence* dari setiap karakter dalam `pattern` yang digunakan untuk algoritma Boyer-Moore. Last occurence adalah index terakhir dari suatu karakter dalam `pattern`.

- a. Inisialisasi array `last` dengan panjang 256 dengan nilai default -1.
- b. Iterasi `i` dari 0 sampai panjang `pattern`.
- c. Assign ASCII value dari karakter `pattern[i]` sebagai index dari array `last` dan assign `i` sebagai value dari index tersebut.
- d. Kembalikan array `last`.

### 5. bmMatch

Fungsi untuk mencari `pattern` dalam `source` dengan algoritma Boyer-Moore. Algoritma ini menggunakan last occurence dari setiap karakter dalam `pattern` untuk mempercepat pencarian `pattern` dalam `source`.

- a. Inisialisasi `last` dengan `buildLast(pattern)`, `n` dengan panjang `source`, `m` dengan panjang `pattern`, `i` dengan `m-1`, dan `j` dengan `m-1`.
- b. Iterasi `i` dari `m-1` sampai `n-1`.
- c. Jika `pattern[j]` sama dengan `source[i]`, maka:
- d. Jika `j` sama dengan 0, maka kembalikan `i`. Jika tidak, maka kurangi 1 dari `i` dan `j`.
- e. Jika `pattern[j]` tidak sama dengan `source[i]`, maka `i` sama dengan `i + m - min(j, 1 + last[source[i]])`, `j` sama dengan `m - 1`, dan `dist` sama dengan `dist + 1`.

- f. Jika iterasi sudah selesai, maka kembalikan -1 yang berarti tidak ada pattern dalam source.

#### 6. hammingDistance

Fungsi untuk menghitung hamming distance dari dua string. Hamming distance adalah jumlah karakter yang berbeda dari dua string yang memiliki panjang yang sama.

- a. Inisialisasi n dengan panjang s1 dan m dengan panjang s2.
- b. Jika n tidak sama dengan m, maka kembalikan -1 yang berarti tidak ada hamming distance dari dua string tersebut.
- c. Iterasi i dari 0 sampai n-1.
- d. Jika s1[i] tidak sama dengan s2[i], maka tambahkan 1 ke distance.
- e. Jika iterasi sudah selesai, maka kembalikan distance.

#### 7. Distance

Fungsi untuk mencari jarak terdekat dari pattern dengan source. Jarak terdekat adalah hamming distance terkecil dari pattern dengan substring dari source yang memiliki panjang yang sama dengan pattern.

- a. Inisialisasi ns dengan panjang source dan np dengan panjang pattern, distance dengan np, s1 dengan string kosong, dan s2 dengan pattern.
- b. Iterasi i dari 0 sampai ns-np.
- c. Masukkan substring dari source dengan panjang np yang dimulai dari i sebagai s1.
- d. Hitung hamming distance dari s1 dengan s2 dan simpan sebagai distTemp.
- e. Jika distTemp lebih kecil dari distance, maka distance sama dengan distTemp.
- f. Setelah selesai iterasi, jika ns sama dengan np, maka kembalikan hamming distance dari source dengan pattern.
- g. Kembalikan distance.

#### 8. calculate

Fungsi untuk menghitung hasil dari ekspresi matematika yang diberikan oleh user. Ekspresi matematika yang diberikan oleh user berupa string yang berisi angka, operator, dan tanda kurung. Operator yang dapat digunakan adalah +, -, \*, /, dan ^. Tanda kurung yang dapat digunakan adalah ( dan ). Fungsi ini menggunakan algoritma Shunting Yard untuk mengubah ekspresi matematika dari infix menjadi postfix dan algoritma Reverse Polish Notation untuk menghitung hasil dari ekspresi matematika yang sudah berupa postfix.

#### 9. getDayName

Fungsi untuk menghitung hari apa di tanggal yang diberikan oleh user. Fungsi ini menerima input berupa string yang berisi tanggal dengan format dd/mm/yyyy.

#### 10. getIdResponse

Fungsi untuk mencari jawaban dari pertanyaan yang diberikan oleh user dengan menggunakan algoritma yang dipilih oleh user. Pertama iterasi dengan algoritma yang dipilih. Kedua, jika tidak ditemukan jawaban, maka iterasi dengan fungsi Distance untuk mencari pertanyaan yang mirip dengan pertanyaan yang diberikan oleh user. Jika ditemukan pertanyaan yang mirip, maka jawab dengan jawaban dari pertanyaan yang mirip tersebut dengan 90% kemiripan. Jika tidak, maka kembalikan list pertanyaan yang mirip dengan pertanyaan yang diberikan oleh user.

- a. question berisi string yang berisi pertanyaan dari user, jadikan question menjadi lowercase
- b. Iterasi idx dari 0 sampai panjang database
- c. source berisi string yang berisi pertanyaan dari database, pattern berisi string yang berisi pertanyaan dari user, temp berisi string kosong, jadikan source menjadi lowercase
- d. Jika panjang source kurang dari panjang pattern, maka tukar source dengan pattern
- e. answer berisi hasil dari algoritma yang dipilih oleh user dengan source dan pattern sebagai parameter
- f. Jika answer tidak sama dengan -1, maka kembalikan jawaban true dan index dari database
- g. Inisialisasi ListOfSim dengan array 3x2, dist berisi integer yang berisi jarak terdekat dari pattern dengan substring dari source yang memiliki panjang yang sama dengan pattern
- h. Iterasi idx dari 0 sampai panjang database
- i. source berisi string yang berisi pertanyaan dari database, pattern berisi string yang berisi pertanyaan dari user, temp berisi string kosong, jadikan source menjadi lowercase
- j. Jika panjang source kurang dari panjang pattern, maka tukar source dengan pattern
- k. dist berisi jarak terdekat dari pattern dengan substring dari source yang memiliki panjang yang sama dengan pattern, tambahkan index dan dist ke ListOfSim dengan menggunakan fungsi addSimilarity
- l. Jika ListOfSim[0][0] sama dengan null, maka ListOfSim[0][0] sama dengan index dan ListOfSim[0][1] sama dengan dist
- m. Jika tidak, maka iterasi i dari 0 sampai 3:
- n. Jika ListOfSim[i][0] sama dengan null, maka ListOfSim[i][0] sama dengan index dan ListOfSim[i][1] sama dengan dist

- o. Jika ListOfSim[i][1] lebih besar dari dist, maka Iterasi j dari 2 sampai i
- p. ListOfSim[j][0] sama dengan ListOfSim[j-1][0], ListOfSim[j][1] sama dengan ListOfSim[j-1][1], dan ListOfSim[i][0] sama dengan index dan ListOfSim[i][1] sama dengan dist
- q. Jika similarity\_question lebih besar dari maxSimilarity, maka kembalikan jawaban true dan ListOfSim[0][0]
- r. Jika iterasi sudah selesai, maka kembalikan jawaban false dan ListOfSim (ListOfSim berisi 3 jawaban terdekat dari pertanyaan yang diberikan oleh user)

#### 11. isThereQuestion

Fungsi untuk mengecek apakah pertanyaan yang diberikan oleh user sudah ada di database atau belum dengan menggunakan algoritma hammingDistance.

#### 4.1.3. Regular Expression

Regular expression digunakan untuk mengklasifikasi tipe-tipe pertanyaan yang ada dalam suatu kalimat utuh. Regular expression berada dalam file *classification.js* Berikut adalah cuplikan kode untuk menunjukkan regular expression yang telah kami buat.

```
const equationRegex =
/(\((.*?)\)?|-?\d+(\.\d+)?)(\s*[-+*/^]\s*(\((.*?)\)|\d+(\.\d+)?))+(\))*/g

const dateRegex = /\d{2,4}[-/]\d{1,2}[-/]\d{2,4}/g;
const addQueryRegex = /(?!<=tambah pertanyaan).*(?=\.\s|$)/gi
const eraseQueryRegex = /(?!<=hapus pertanyaan).*(?=\.\s|$)/gi
const questionWithAndRegex = /.+?([.?!]\s*$)/g;
const questionRegex = /.+?(?:([.?!]\s*)|(?<:dan\s+)|$)/g;
```

equationRegex:

```
/(\((.*?)\)?|-?\d+(\.\d+)?)(\s*[-+*/^]\s*(\((.*?)\)|\d+(\.\d+)?))+(\))*/g
```

Regex ini digunakan untuk memvalidasi persamaan matematika. Ini terdiri dari beberapa bagian:

$(\((.*?)\)?|-?\d+(\.\d+)?)$  : Mengidentifikasi angka atau tanda kurung sebagai karakter pertama di persamaan. Ini mencocokkan angka dengan satu atau dua tanda negatif atau desimal.



`(\s*[-+*/^]\s*(\((.*?\)|\d+(\.\d+)?) )+ :` Mengidentifikasi operator matematika (seperti +, -, \*, /, dan ^) diikuti oleh angka atau tanda kurung. Ini dapat berulang beberapa kali.

`(\))*/g :` Mengidentifikasi tanda kurung tutup sebagai karakter terakhir dalam persamaan. Ini dapat berulang beberapa kali.

`dateRegex: /\d{2,4}[-/]\d{1,2}[-/]\d{2,4}/g;`

Regex ini digunakan untuk memvalidasi tanggal dalam format tertentu. Ini mencocokkan tanggal dengan format DD/MM/YYYY atau DD-MM-YYYY, di mana DD adalah 2 digit untuk hari, MM adalah 1 atau 2 digit untuk bulan, dan YYYY adalah 2 atau 4 digit untuk tahun.

`\d{2,4} :` Mencocokkan dua, tiga, atau empat digit berturut-turut. Dalam konteks ini, ini digunakan untuk mencocokkan dua digit untuk tanggal dan empat digit untuk tahun.

`[-/] :` Mencocokkan tanda hubung (-) atau garis miring (/) yang digunakan sebagai pemisah antara tanggal, bulan, dan tahun.

`\d{1,2} :` Mencocokkan satu atau dua digit berturut-turut untuk bulan.

`\d{2,4} :` Mencocokkan dua, tiga, atau empat digit berturut-turut untuk tahun.

`/g :` Merupakan opsi global untuk mencocokkan semua kemunculan dari regex tersebut dalam suatu string.

`addQueryRegex: /(?!<=tambah pertanyaan).*?(?=\.\s|$)/gi`

Regex ini digunakan untuk mengekstrak pertanyaan dari sebuah teks. Ini mencocokkan semua teks yang dimulai dengan "tambah pertanyaan" dan diakhiri dengan titik atau spasi diikuti oleh titik.

`(?!<=tambah pertanyaan)` adalah positive lookbehind assertion, yang berarti substring yang dicari harus diawali dengan "tambah pertanyaan".

`. *?` adalah lazy quantifier, yang berarti mencari nol atau lebih karakter (termasuk spasi, titik, tanda tanya, dll.) secara non-greedy (paling sedikit mungkin) untuk membentuk substring yang diinginkan.

`(?=\.\s|$)` adalah positive lookahead assertion, yang berarti substring yang dicari harus diakhiri dengan titik "." dan spasi atau akhir string.

`/gi` digunakan untuk mengaktifkan pencarian global (g) dan case-insensitive (i).

```
eraseQueryRegex:/(?<=hapus pertanyaan).*?(?=\.\s|$)/gi
```

Regex ini digunakan untuk menghapus pertanyaan dari sebuah teks. Ini mencocokkan semua teks yang dimulai dengan "hapus pertanyaan" dan diakhiri dengan titik atau spasi diikuti oleh titik. Cara berkerjanya sama seperti `addQueryRegex` hanya bda di tambah pertanyaan di ubah menjadi hapus pertanyaan.

```
questionWithAndRegex: /.+?([.?!]\s*|$)/g;
```

Regex ini digunakan untuk mengekstrak kalimat dari sebuah teks. Ini mencocokkan semua teks yang diakhiri oleh tanda baca seperti titik, tanda tanya, atau seru. Ini juga dapat mencocokkan tanda spasi setelah tanda baca atau mencocokkan langsung diakhir teks.

`.+?` : Mencocokkan satu atau lebih karakter apa pun (+) yang muncul minimal sekali dan maksimal sebanyak mungkin, namun bersifat non-greedy (?), artinya akan mencoba untuk mencocokkan sebanyak mungkin sampai bertemu dengan pola berikutnya.

`([.?!]\s*|$)` : Grup pengapit yang mencocokkan tanda baca (. atau ? atau !) yang diikuti dengan satu atau lebih spasi (\s\*), atau (|) yang berarti alternatif, end of string (\$), artinya mencocokkan dengan tanda baca yang diikuti dengan spasi atau end of string.

```
questionRegex: /.+?(?:(?:[.?!]\s*)|(?:dan\s+)|$)/g;
```

Regex ini digunakan untuk mengekstrak pertanyaan dari sebuah teks. Ini mencocokkan semua teks yang dimulai dengan kalimat dan diakhiri dengan tanda tanya. Ini juga dapat mencocokkan tanda spasi setelah tanda baca atau kata "dan" setelah kalimat, atau mencocokkan langsung diakhir teks.

`.+?:` Mencocokkan satu atau lebih karakter apapun (.) sebanyak mungkin (+?), tetapi sedikit mungkin sehingga tidak mengambil tanda baca yang ada setelahnya.

`(?:` : Memulai sebuah grup yang tidak perlu direkam.

`(?:[.?!]\s*)` : Mencocokkan satu tanda baca (. atau ? atau !) dan beberapa spasi (\s\*). Tanda baca ini menandakan akhir kalimat dan pertanyaan yang mungkin berada di dalamnya.

`|` : Atau.

`(?:dan\s+)` : Mencocokkan kata "dan" diikuti oleh beberapa spasi (\s+). Hal ini memungkinkan mencari pertanyaan yang mengandung kata "dan" di dalamnya.

\$) : Mencocokkan akhir string. Ini memastikan bahwa pertanyaan yang tidak diakhiri oleh tanda baca tetap tercakup.

#### 4.2. Tata Cara Penggunaan Program

Cara input:

Untuk melakukan perhitungan, cukup masukkan ekspresi matematika yang ingin anda hitungkan, operator yang bisa di gunakan adalah +,-,\*,/,^ apabila sesuai maka angka yang dimasukkan dan operator yang digunakan akan menghasilkan hasil dari perhitungan. Selain itu apabila ingin memasukkan dua ekspresi matematika bisa dilakukan sebagai berikut:

-1+1 5+2

-1+1 (5+2)

(-1+1) 5+2

(-1+1) (5+2)

Jadi pastikan tidak ada operator diantara angka agar program bisa mendapat dua ekspresi matematika



Untuk menentukan hari pastikan mengikuti format DD-MM-YYYY atau DD/MM/YYYY.













Untuk menambahkan pertanyaan kepada database, harus mengikuti format “tambah pertanyaan {pertanyaan} dengan jawaban {jawaban}.” kalimat awal harus tambah pertanyaan, dan harus diakhiri oleh titik dan harus ada pertanyaan dan jawaban.



Untuk menghapus pertanyaan kepada database, harus mengikuti format “hapus pertanyaan {pertanyaan}.” kalimat awal harus pertanyaan, dan harus diakhiri oleh titik.

Untuk pertanyaan, bisa ketik apa saja asal tidak berisi komponen-komponen dari keempat masukkan diatas, dan kalimat pertanyaan harus diakhiri dengan tanda tanya atau titik agar regex tahu bahwa ada lebih dari satu pertanyaan.

#### 4.3. Hasil Pengujian dan Analisis Hasil Pengujian

Klasifikasi Query	Hasil Pengujian	Analisis
Penambahan pertanyaan	 tambah pertanyaan apakah mayonaise itu? dengan jawaban sejenis instrumen.  Pertanyaan "apakah mayonaise itu?" berhasil ditambah	Program berhasil mengidentifikasi bahwa <i>query</i> merupakan sebuah penambahan pertanyaan. Program juga berhasil mengambil <i>string</i> pertanyaan dan jawaban dari input pengguna lalu menambahkannya pada basis data.

Penghapusan pertanyaan	 hapus pertanyaan apakah mayonaise itu?  Pertanyaan "apakah mayonaise itu?" telah dihapus	Program berhasil mengidentifikasi bahwa <i>query</i> merupakan sebuah penghapusan pertanyaan. Program juga berhasil menghapus pertanyaan dari basis data.
Pencarian pertanyaan	 apakah mayonaise itu?  Jawaban untuk "apakah mayonaise itu?" adalah "sejenis instrumen"	Program berhasil melakukan <i>string matching</i> untuk menentukan <i>exact match</i> dari pertanyaan yang di input pengguna.
Pencarian pertanyaan (kasus 90% kemiripan)	 1234567899  Jawaban untuk "1234567899" adalah "itu adalah angka-angka"	Program berhasil melakukan <i>string matching</i> untuk menentukan kemiripan terdekat dari pertanyaan yang di input pengguna. Dari gambar disamping, terlihat bahwa tingkat kemiripan antara pertanyaan dalam database dengan pertanyaan pengguna mirip 90% yaitu hanya berbeda pada angka terakhir.
Pencarian pertanyaan (kasus tidak ada yang mirip)	 apakah IF gampang?  Pertanyaan apakah IF gampang? tidak ditemukan di database Apakah maksud anda: 1. apakah IF susah? 2. apakah ITb susah? 3. apa itu kuda?	Program berhasil melakukan <i>string matching</i> untuk menentukan kemiripan terdekat dari pertanyaan yang di input pengguna. Dari kode disamping, terlihat bahwa tingkat kemiripan antara pertanyaan dalam database dengan pertanyaan pengguna tidak ada yang lebih besar atau sama dengan 90% sehingga program memberikan kemungkinan pertanyaan yang dimaksud yaitu list 3 pertanyaan termirip yang dihasilkan dari fungsi Distance yang sudah dijelaskan di atas. Dari gambar disamping, program berhasil menampilkan 3 pertanyaan termirip.
Pemrosesan tanggal	 01/12/3030  Hari untuk tanggal 01/12/3030 adalah Rabu	Program berhasil mengidentifikasi bahwa <i>query</i> merupakan sebuah tanggal dan berhasil menentukan hari dari tanggal tersebut.
Kalkulasi operasi matematika	 $10 * 10 * 10 + 2 * 5 / 10$  Jawaban untuk persamaan matematika $10 * 10 * 10 + 2 * 5 / 10$ adalah 1001	Program berhasil mengidentifikasi bahwa <i>query</i> merupakan operasi matematis dan berhasil menentukan jawabannya dengan benar.

<p>Beberapa query sekaligus</p>	<div> <div>  10/12/2023. <math>10 + 10</math>. apakah IF susah? </div> <div>  Hari untuk tanggal 10/12/2023 adalah Minggu  Jawaban untuk persamaan matematika <math>10 + 10</math> adalah 20  Jawaban untuk "apakah IF susah?" adalah "enggga kok gampang banget" </div> </div>	<p>Program berhasil mengidentifikasi semua query dengan benar dan memberikan jawaban yang sesuai untuk setiap query. Program memisahkan antar query disamping dengan tanda titik.</p>
---------------------------------	---	---

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1. Kesimpulan**

Algoritma pencocokan kalimat dapat digunakan untuk menemukan sebuah upa-kalimat pada kalimat dalam sebuah database. Algoritma yang digunakan dalam tugas besar ini adalah algoritma Knuth-Morris-Pratt dan Boyer-Moore untuk pencocokan exact match. Konsep exact match di sini adalah terdapat upa-kalimat pada kalimat pertanyaan dalam database. Jika tidak ada yang sama maka akan digunakan algoritma Hamming Distance yaitu algoritma yang menghitung berapa banyak karakter berbeda dari dua kalimat yang disebut sebagai distance. Penerapan regular expression pada tugas besar ini adalah dalam metode klasifikasi masukan dari pengguna. Klasifikasi tersebut menjadi tanggal, ekspresi matematika, menambah dan menghapus pertanyaan ke dalam database, dan menemukan jawaban dari pertanyaan yang diberikan dengan algoritma pencocokan kalimat.

#### **5.2. Saran**

Pada pengerjaan tugas besar ini, terdapat kendala dalam proses pengerjaan yang tidak dilakukan segera dikarenakan banyaknya tugas besar mata kuliah lain dan libur Hari Raya Idul Fitri.

#### **5.3. Komentar dan Refleksi**

Pada tugas besar kali ini, seharusnya kami dapat memberikan yang lebih baik dalam implementasi tugas. Kami menyadari bahwa manajemen waktu, proses pemecahan masalah, dan pengerjaan tugas merupakan faktor penting dalam pengerjaan tugas besar ini. Komentar yang dapat kami berikan yaitu, spesifikasi dapat lebih diperjelas, kami merasa spesifikasi masih kurang menjelaskan dengan detail terutama terhadap implementasi algoritma.

## Referensi

Boyer, R. S., & Moore, J. S. (1977). A fast string searching algorithm. *Communications of the ACM*, 20(10), 762-772.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press.

Galil, Z. (1981). On improving the worst-case running time of the Knuth-Morris-Pratt algorithm. *Journal of the ACM (JACM)*, 28(4), 633-637.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>

<https://www.ibm.com/cloud/learn/web-applications>

<https://www.techopedia.com/definition/1502/web-application>

Knuth, D. E., Morris, J. H., & Pratt, V. R. (1977). Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2), 323-350.

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

## Lampiran

### Website

<https://tubes3-chat-gps-fe.vercel.app/>

### Repository

[NerbFox/Tubes3\\_ChatsGPS\\_BE \(github.com\)](#)

[NerbFox/Tubes3\\_ChatsGPS\\_FE \(github.com\)](#)

### Link video:

<https://youtu.be/nbdmPv4NEBY>

[https://itb-ac-id.zoom.us/rec/play/Xx7sVL3YDIpXIBekQPfNCWPvvhgKLV4I2qmGjgLP9z8C71yv0gL81L-Eh-yUC\\_zyEV4ZB0sv6hEnlwKJ.JScTI9LcBsTjgCOx?autoplay=true&startTime=1683294394000](https://itb-ac-id.zoom.us/rec/play/Xx7sVL3YDIpXIBekQPfNCWPvvhgKLV4I2qmGjgLP9z8C71yv0gL81L-Eh-yUC_zyEV4ZB0sv6hEnlwKJ.JScTI9LcBsTjgCOx?autoplay=true&startTime=1683294394000) (backup apabila link youtube tidak bisa, passcode: PHvB5+NK)