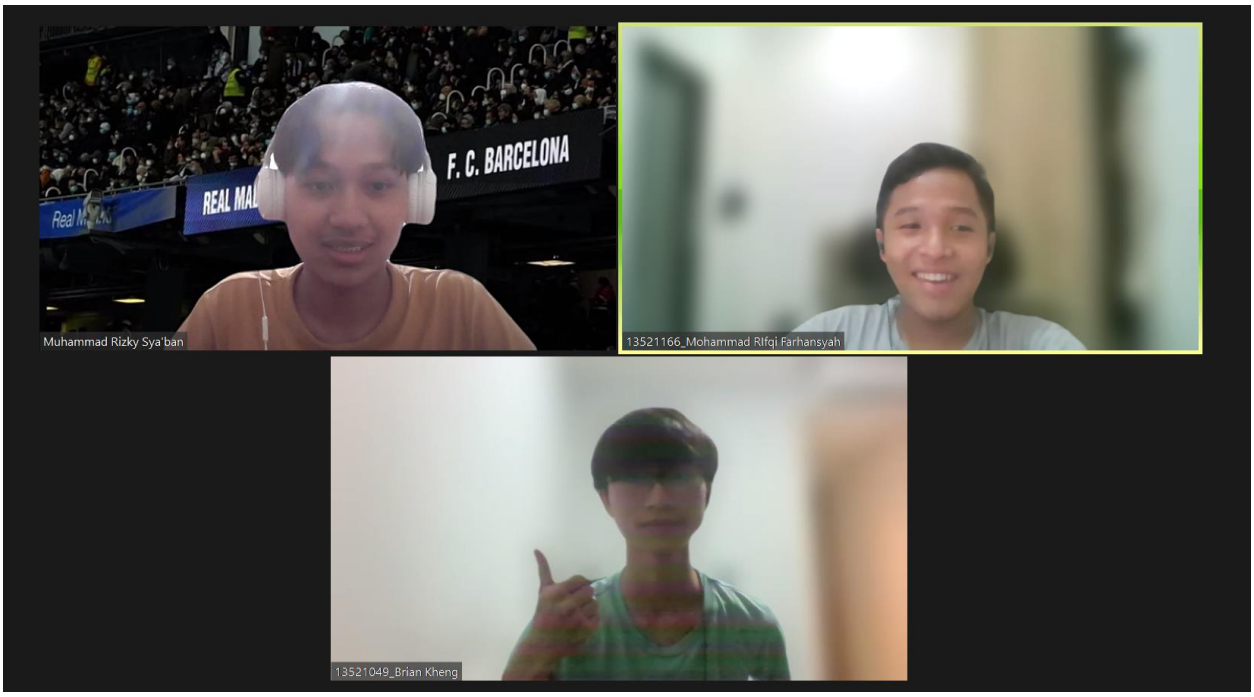
Laporan Tugas Besar 3 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Penerapan *String Matching* dan *Regular Expression* dalam Pembuatan *ChatGPT* Sederhana



Kelompok 51 - Kitzuiteyo, nee kitzuiteyo.. Honey, Baby you Baby you

Anggota Kelompok:

Brian Kheng	13521049
Muhammad Rizky Sya'ban	13521119
Mohammad Rifqi Farhansyah	13521166

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

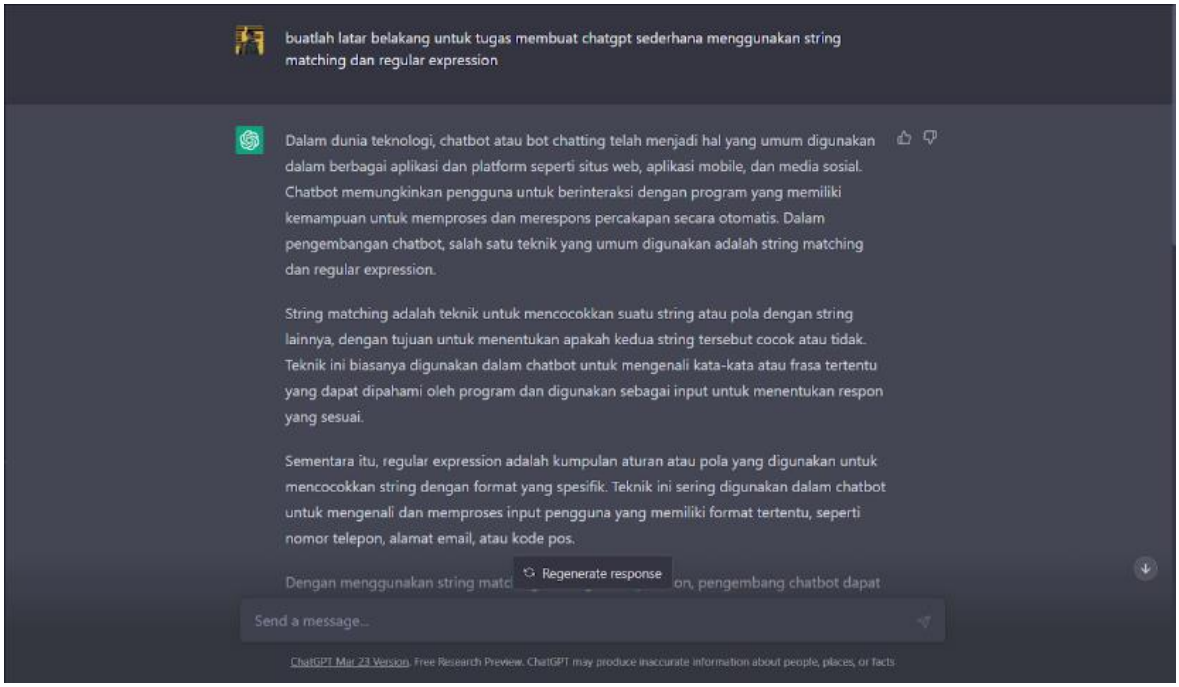
Daftar Isi

Daftar Gambar

BAB I

1.1. Latar Belakang

Dalam dunia teknologi, chatbot telah menjadi hal yang umum digunakan dalam berbagai aplikasi dan platform seperti situs web, aplikasi mobile, dan media sosial. Chatbot memungkinkan pengguna untuk berinteraksi dengan program yang memiliki kemampuan untuk memproses dan merespons percakapan secara otomatis. Salah satu contoh chatbot yang sedang booming saat ini adalah **ChatGPT**.



Gambar 1.1.1. *Ilustrasi Chatbot ChatGPT*
Sumber: https://docs.google.com/document/d/1d1eimK5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit

Pembangunan chatbot dapat dilakukan dengan menggunakan berbagai pendekatan dari bidang Question Answering (QA). Pendekatan QA yang paling sederhana adalah menyimpan sejumlah pasangan pertanyaan dan jawaban, menentukan pertanyaan yang paling mirip dengan pertanyaan yang diberikan pengguna, dan memberikan jawabannya kepada pengguna. Untuk mencocokkan input pengguna dengan pertanyaan yang disimpan pada database, kalian bisa menggunakan string matching.

String matching adalah teknik untuk mencocokkan suatu string atau pola dengan string lainnya, dengan tujuan untuk menentukan apakah kedua string tersebut cocok atau tidak. Teknik ini biasanya digunakan dalam chatbot untuk mengenali kata-kata atau frasa tertentu yang dapat dipahami oleh program dan digunakan sebagai input untuk menentukan respon yang sesuai. Sementara itu, regular expression adalah kumpulan aturan atau pola yang digunakan untuk pencocokan string dengan format yang spesifik. Teknik ini sering digunakan dalam chatbot untuk mengenali dan memproses input pengguna yang memiliki format tertentu, seperti nomor telepon, alamat email, atau kode pos.

1.2. Deskripsi Tugas

Dalam tugas besar ini, anda diminta untuk 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.

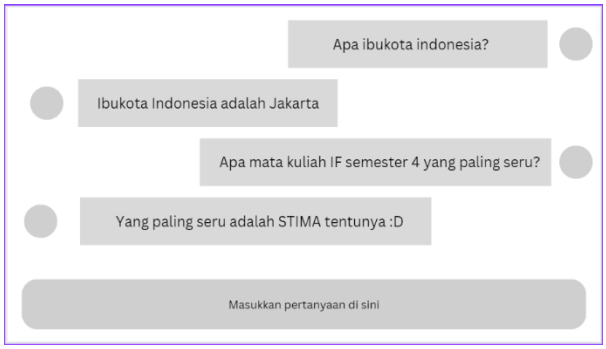
Perhitungan tingkat kemiripan dibebaskan kepada anda asalkan dijelaskan di laporan, namun disarankan menggunakan salah satu dari algoritma Hamming Distance, Levenshtein Distance, ataupun Longest Common Subsequence.

1.3. Fitur-Fitur Aplikasi

ChatGPT sederhana yang anda membuat wajib dapat melakukan beberapa fitur / klasifikasi query seperti berikut:

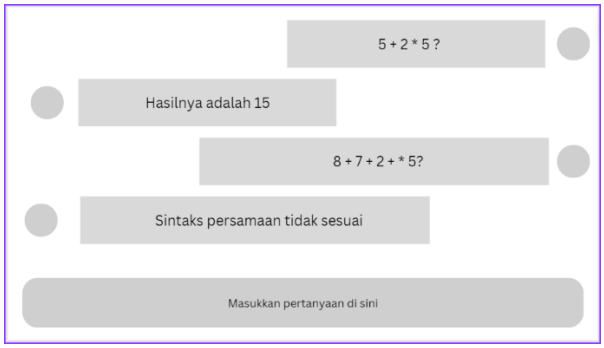
1. **Fitur pertanyaan teks (didapat dari database)**
   Mencocokkan pertanyaan dari input pengguna ke pertanyaan di database menggunakan algoritma **KMP atau BM**.
2. **Fitur kalkulator**
   Pengguna memasukkan input query berupa persamaan matematika. Contohnya adalah 2*5 atau 5+9*(2+4). Operasi cukup Tambah, kurang, kali, bagi, pangkat, kurung.
3. **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.
4. **Tambah pertanyaan dan jawaban ke database**
   Pengguna dapat menambahkan pertanyaan dan jawabannya sendiri ke database dengan query contoh “Tambahkan pertanyan xxx dengan jawaban yyy”. Menggunakan algoritma string matching untuk mencari tahu apakah pertanyaan sudah ada. Apabila sudah, maka jawaban akan diperbaharui.
5. **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** pada website bagi pengguna 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)

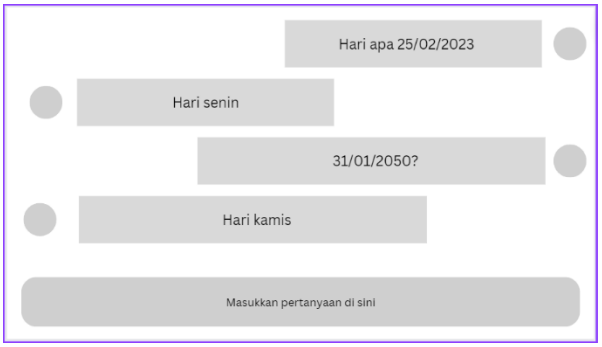


Gambar 1.3.1. *Ilustrasi Fitur Pertanyaan teks kasus exact*
Sumber: https://docs.google.com/document/d/1d1eim K5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit

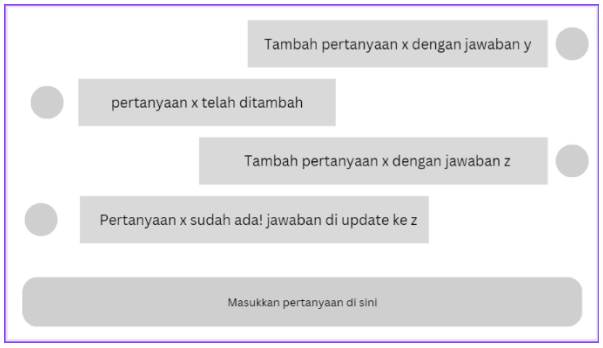
Gambar 1.3.2. *Ilustrasi Fitur Pertanyaan teks kasus tidak exact*
Sumber: https://docs.google.com/document/d/1d1eim K5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit



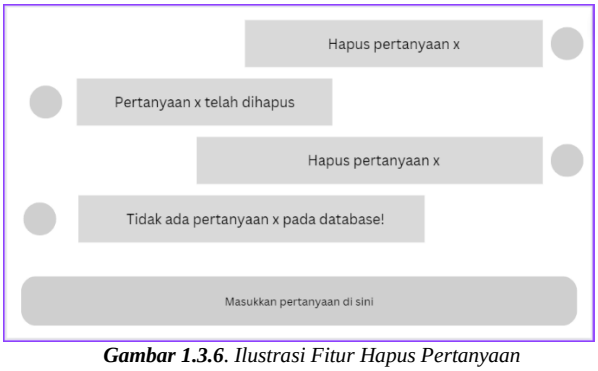
Gambar 1.3.3. *Ilustrasi Fitur Kalkulator*
Sumber: https://docs.google.com/document/d/1d1eim K5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit



Gambar 1.3.4. *Ilustrasi Fitur Tanggal*
Sumber: https://docs.google.com/document/d/1d1eim K5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit

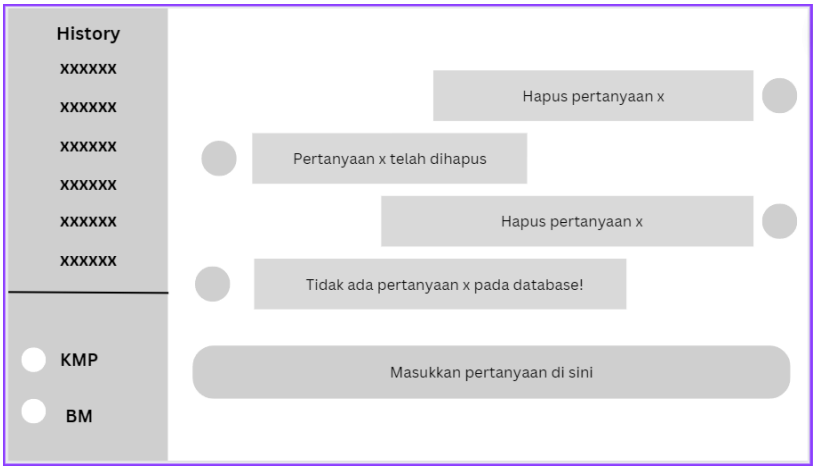


Gambar 1.3.5. *Ilustrasi Fitur Tambah Pertanyaan*
Sumber: https://docs.google.com/document/d/1d1eim K5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit



Gambar 1.3.6. *Ilustrasi Fitur Hapus Pertanyaan*
Sumber: https://docs.google.com/document/d/1d1eim K5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit

Layaknya **ChatGPT**, di sebelah kiri disediakan **history** dari hasil pertanyaan anda. Cukup tampilkan 5-10 pertanyaan terbaru di toolbar kiri. Perhatikan bahwa sistem history disini disamakan dengan chatGPT, sehingga satu history yang diklik menyimpan **seluruh pertanyaan pada sesi itu**. Apabila history diclick, maka akan merestore seluruh pertanyaan dan jawaban di halaman utama. Contoh ilustrasi keseluruhan:



***Gambar 1.3.7**. Ilustrasi Keseluruhan*
Sumber: https://docs.google.com/document/d/1d1eimK5XlJ7QYSb4iUE4ZmBBTsg6e7vF/edit

1.4. Spesifikasi Tugas

Berikut ini adalah spesifikasi program dan laporan dari tugas besar ini

1.4.1. Spesifikasi Program

1. Aplikasi berbasis website dengan pembagian Frontend dan Backend yang jelas.
2. Implementasi Backend **wajib** menggunakan Node.js / Golang, sedangkan Frontend dibebaskan tetapi **disarankan** untuk menggunakan React / Next.js / Vue / Angular. Lihat referensi untuk selengkapnya.
3. Penyimpanan data **wajib** menggunakan basis data (MySQL / PostgreSQL / MongoDB).
4. Algoritma pencocokan string (KMP dan Boyer-Moore) dan Regex **wajib** diimplementasikan pada sisi Backend aplikasi.
5. Informasi yang **wajib** 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”).

1.4.2. Spesifikasi Laporan

1. **Cover**: Cover laporan ada foto anggota kelompok (foto bertiga). Foto ini menggantikan logo “gajah” ganesha.
2. **Bab 1**: Deskripsi tugas (dapat menyalin spesifikasi tugas ini).
3. **Bab 2**: Landasan Teori
4. **Bab 3**: Analisis Pemecahan Masalah.
5. **Bab 4**: Implementasi dan pengujian.
6. **Bab 5**: Kesimpulan, saran, dan komentar/refleksi tentang tugas besar 3 ini.
7. Daftar Pustaka.

Spesifikasi detail dari Tugas Besar 3 - Strategi Algoritma 2022/2023 dapat diakses melalui pranala berikut ini.

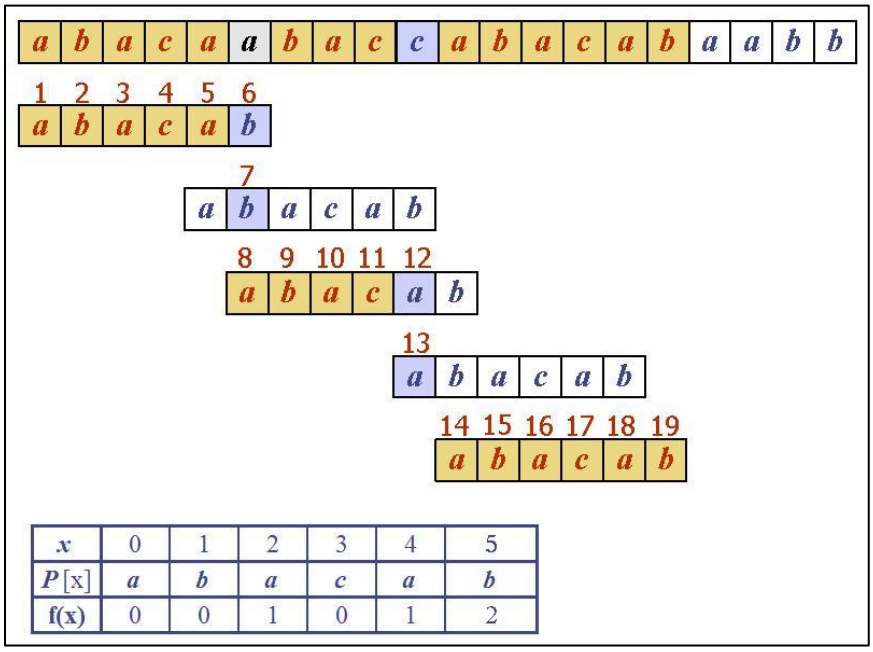
BAB II
Landasan Teori

2.1. Pattern Matching

Pattern matching adalah proses mencari dan mencocokkan pola atau serangkaian karakter dalam sebuah teks. Teknik ini digunakan untuk menemukan kemiripan atau pola tertentu dalam data atau teks yang kompleks dan besar. *Pattern matching* biasanya dilakukan dengan menggunakan algoritma dan metode yang telah dirancang khusus untuk menemukan pola yang diinginkan, seperti algoritma *Knuth-Morris-Pratt*, *algoritma Boyer-Moore*, dan algoritma *regular expression*. Contoh penggunaan *pattern matching* adalah dalam pencarian kata kunci di mesin pencari atau dalam analisis teks untuk menemukan pola tertentu, seperti nama orang, alamat email, nomor telepon, dan sebagainya. *Pattern matching* juga sering digunakan dalam pemrograman untuk memproses data, seperti dalam pengolahan citra dan pengolahan bahasa alami.

2.2. Algoritma Knuth-Morris-Pratt (KMP)

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, tetapi keduanya mempublikasikannya secara bersamaan pada tahun 1977. Jika kita melihat algoritma *brute force* lebih mendalam, kita mengetahui bahwa dengan mengingat beberapa perbandingan yang dilakukan sebelumnya kita dapat meningkatkan besar pergeseran yang dilakukan. Hal ini akan menghemat perbandingan, yang selanjutnya akan meningkatkan kecepatan pencarian.



x	0	1	2	3	4	5
P[x]	a	b	a	c	a	b
f(x)	0	0	1	0	1	2

Gambar 2.2.1. *Ilustrasi Algoritma Knuth-Morris-Pratt*
Sumber: https://th.bing.com/th/id/R.0b176964efa99370ca0ff7fb225db62d?rik=HLgtvxZnYk1uXw&riu=http%3a%2f%2f3.bp.blogspot.com%2f-VYM2tFCZAYY%2fVmmBlGy5WSI%2fAAAAAAAAAbs%2fRxuXJUcG1pw%2fs1600%2fkmpexample.jpg&ehk=y5BQDsA26q4RM35TVIr8lwoSFI995kaWGxBI8iSNAe8%3d&risl=&pid=ImgRaw&r=0

Perhitungan penggeseran pada algoritma ini adalah sebagai berikut, bila terjadi ketidakcocokkan pada saat pattern sejajar dengan $teks[i..i+n-1]$, kita bisa menganggap ketidakcocokan pertama terjadi di antara $teks[i+j]$ dan $pattern[j]$, dengan $0 < j < n$. Berarti, $teks[i..i+j-1] = pattern[0..j-1]$

dan $a = teks[i + j]$ tidak sama dengan $b = pattern[j]$. Ketika kita menggser, sangat beralasan bila ada sebuah awalan v dari pattern akan sama dengan sebagian akhiran u dari sebagian teks. Sehingga kita bisa menggeser pattern agar awalan v tersebut sejajar dengan akhiran dari u.

Dengan kata lain, pencocokan string akan berjalan secara efisien bila kita mempunyai tabel yang menentukan berapa panjang kita seharusnya menggeser seaindainya terdeteksi ketidakcocokan di karakter ke-j dari pattern. Tabel itu harus memuat $next[j]$ yang merupakan posisi karakter $pattern[j]$ setelah digeser, sehingga kita bisa menggeser pattern sebesar $j - next[j]$ relatif terhadap teks.

Secara sistematis, langkah langkah yang dilakukan algoritma Knuth-Morris Pratt pada saat mencocokkan string:

1. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
2. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   - Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
   - Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Secara umum, berikut adalah pseudocode algoritma KMP pada fase pra-pencarian:

```
procedure preKMP(input P: array[0…n-1] of char, input n: integer, input/output kmpNext: array[0..n] of integer)
{  I.S. Array P terdefinisi dan berisi n karakter, integer n terdefinisi sebagai panjang array P, array kmpNext terdefinisi dan berisi n+1 bilangan bulat, nilai i dan j terinisialisasi dengan nilai awal, proses pencocokan pola diawali.
   F.S. Array kmpNext berisi nilai-nilai hasil proses algoritma KMP yang merepresentasikan panjang prefix terpanjang dari pola yang cocok dengan suffix dari posisi tertentu dalam pola itu sendiri.}
KAMUS
  { Variabel }
  i, j : integer

ALGORITMA
  { inisialisasi nilai i }
  i:= 0;
  { inisialisasi nilai j }
  j:= kmpNext[0]:= -1;
  { proses looping }
  while (i < n) {
     while (j > -1 and not(P[i] = P[j]))
        j:= kmpNext[j];
     i:= i+1;
     j:= j+1;
     if (P[i] = P[j])
        kmpNext[i]:= kmpNext[j];
     else
        kmpNext[i]:= j;
     endif
  endwhile
```

Dan berikut adalah pseudocode algoritma KMP pada fase pencarian:

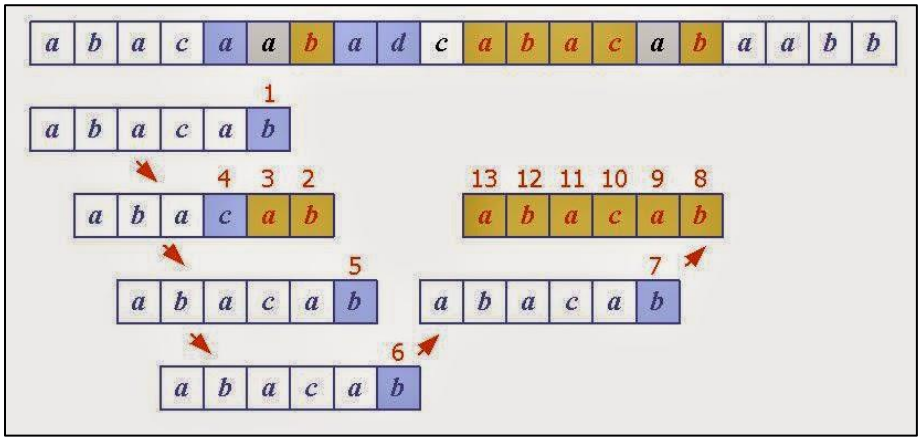
```
procedure KMPSearch(input m, n: integer, input P: array[0…n-1] of char, input T: array[0…m-1] of char, output ketemu: array[0..m-1] of boolean)
{  I.S. P dan T adalah array yang terdefinisi dan berisi karakter, n dan m adalah bilangan bulat yang terdefinisi, ketemu adalah array boolean yang terdefinisi dan berisi nilai false, kmpNext adalah array integer yang terdefinisi dan berisi nilai-nilai hasil proses algoritma KMP, dan proses pencocokan pola diawali.
```

```
   F.S. Array ketemu berisi nilai boolean yang menunjukkan apakah pola P ditemukan dalam teks T pada
posisi tertentu atau tidak, dan array kmpNext berisi nilai-nilai hasil proses algoritma KMP yang
merepresentasikan panjang prefix terpanjang dari pola yang cocok dengan suffix dari posisi tertentu dalam
pola itu sendiri. }
KAMUS
  { Variabel }
  i, j, next : integer
  kmpNext : array[0...n] of integer

ALGORITMA
  { pemanggilan fungsi preKMP }
  preKMP(n, P, kmpNext);
  { inisialisasi nilai j }
  i:=0
  { proses looping }
  while (i <= m-n) {
     j:=0
     while (j < n and T[i+j] = P[j]) do
         j:=j+1
     endwhile
     if(j >= n) then
         ketemu[i]:=true
     endif
     next:= j - kmpNext[j]
     i:= i+next
  endwhile
```

2.3. Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore adalah salah satu algoritma pencarian string, dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada tahun 1977. Algoritma ini dianggap sebagai algoritma yang paling efisien pada aplikasi umum. Tidak seperti algoritma pencarian string yang ditemukan sebelumnya, algoritma Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat.



Gambar 2.3.1. *Ilustrasi Algoritma Boyer-Moore*
Sumber: https://4.bp.blogspot.com/_ebteqMFaxAo/S8q2LZaaCaI/AAAAAAAADRE/LDa2wx554Q8/s1600/complete_example.JPG

Cara kerja dari algoritma Boyer-Moore adalah misalnya ada sebuah usaha pencocokan yang terjadi pada $teks[i..i+n-1]$, dan anggap ketidakcocokan pertama terjadi di antara $teks[i+j]$ dan $b = pattern[j]$. Jika u adalah akhiran dari pattern sebelum b dan v adalah sebuah awalan dari pattern, maka penggeseran-penggeseran yang mungkin adalah:

1. Penggeseran good-suffix yang terdiri dari menyejajarkan potongan $teks[i+j+1..i+n-1]$ dengan kemunculannya paling kanan di pattern yang didahului oleh karakter yang berbeda dengan $pattern[j]$. Jika tidak ada potongan seperti itu, maka algoritma akan menyejajarkan akhiran v dari $teks[i+j+1..i+n-1]$ dengan awalan dari pattern yang sama.
2. Penggeseran bad character yang terdiri dari menyejajarkan $teks[i+j]$ dengan kemunculan paling kanan karakter tersebut di pattern. Bila karakter tersebut tidak ada di pattern, maka pattern akan disejajarkan dengan $teks[i+n+1]$.

Secara sistematis, langkah-langkah yang dilakukan algoritma Boyer-Moore pada saat mencocokkan string adalah:

1. Algoritma Boyer-Moore mulai mencocokkan pattern pada awal teks.
2. Dari kanan ke kiri, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   - Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
   - Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern dengan memaksimalkan nilai penggeseran good-suffix dan penggeseran bad-character, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Secara umum, berikut adalah pseudocode algoritma Boyer-Moore pada fase pra-pencarian:

```
procedure preBmBc(input P: array[0…n-1] of char, input n: integer, input/output bmBc: array[0..n-1] of integer)
{  I.S. Array P terdefinisi dan berisi n karakter, integer n terdefinisi sebagai panjang array P dan array bmBc terdefinisi dan berisi n bilangan bulat dengan nilai default m, proses looping diawali.
   F.S. Array bmBc berisi nilai-nilai hasil proses algoritma Boyer-Moore yang merepresentasikan pergeseran yang dilakukan pada pencarian pola.}
KAMUS
  { Variabel }
  I : integer

ALGORITMA
  { proses looping }
  for (i:= 0 to ASIZE-1)
     bmBc[i]:= m
  endfor
  for (i:= 0 to m - 2)
     bmBc[P[i]]:= m - i - 1
  endfor
```

```
procedure preSuffixes(input P: array[0…n-1] of char, input n: integer, input/output suff: array[0..n-1] of integer)
{  I.S. Array P terdefinisi dan berisi n karakter, integer n terdefinisi sebagai panjang array P, array suff terdefinisi dan berisi n bilangan bulat dengan nilai sembarang.
   F.S. Array suff terisi dengan nilai-nilai hasil proses algoritma KMP yang merepresentasikan panjang suffix terpanjang dari pola yang cocok dengan prefix dari posisi tertentu dalam pola itu sendiri.}
KAMUS
  { Variabel }
  f, g, i : integer

ALGORITMA
  { inisialisasi nilai suff[n-1] }
  suff[n - 1]:= n
  { inisialisasi nilai g }
  g:= n - 1
  { proses looping }
  for (i:= n - 2 downto 0) {
     if (i > g and (suff[i + n - 1 - f] < i - g))
```

```
        suff[i]:= suff[i + n - 1 - f]
     else
        if (i < g)
           g:= i
        endif
        f:= i
        while (g >= 0 and P[g] = P[g + n - 1 - f])
           --g
        endwhile
        suff[i] = f - g
     endif
  endfor
```

```
procedure preBmGs(input P: array[0…n-1] of char, input n: integer, input/output bmBc: array[0..n-1] of integer)
{  I.S. Array P terdefinisi dan berisi n karakter, integer n terdefinisi sebagai panjang array P, array bmBc terdefinisi dan berisi n bilangan bulat. Proses looping akan dimulai.
   F.S. Array bmGs berisi nilai-nilai hasil proses algoritma Boyer-Moore yang merepresentasikan perpindahan pola yang dilakukan pada saat proses pencocokan pola berlangsung.}
KAMUS
  { Variabel }
  i, j : integer
  suff : array[0..RuangAlpabet] of integer

ALGORITMA
  { pemanggilan fungsi preSuffixes }
  preSuffixes(x, n, suff);
  { proses looping }
  for (i:= 0 to m-1)
     bmGs[i]:= n
  endfor
  { inisialisasi nilai j }
  j:= 0
  for (i:= n - 1 downto 0)
     if (suff[i] = i + 1)
        for (j:=j to n - 2 - i)
           if (bmGs[j] = n)
              bmGs[j]:= n - 1 - i
           endif
        endfor
     endif
  endfor
  for (i = 0 to n - 2)
     bmGs[n - 1 - suff[i]]:= n - 1 - i;
  endfor
```

Dan berikut adalah pseudocode algoritma Boyer-Moore pada fase pencarian:

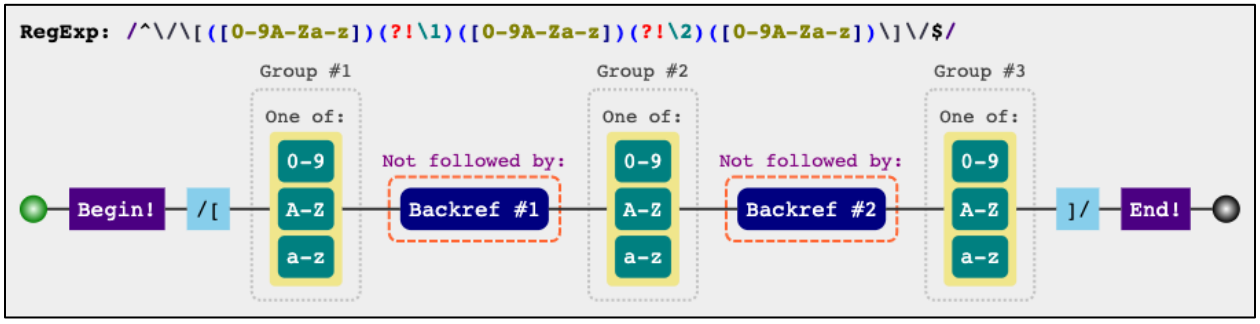
```
procedure BoyerMooreSearch(input P: array[0…n-1] of char, input P: array[0…m-1] of char, input m, n: integer, input/output ketemu: array[0..n-1] of integer)
{  I.S. Array P dan array T terdefinisi, dengan masing-masing berisi n dan m karakter, integer n dan m terdefinisi sebagai panjang array P dan T, array ketemu terdefinisi dan berisi n bilangan bulat 0, variabel i, j, shift, bmBcShift, dan bmGsShift terinisialisasi dengan nilai awal, proses pencarian pola diawali.
   F.S. Array ketemu berisi nilai true pada indeks yang sesuai dengan awal kemunculan pola dalam array T, jika ada, atau tetap 0 jika pola tidak ditemukan.}
KAMUS
  { Variabel }
  i, j, shift, bmBcShift, bmGsShift: integer
  BmBc: array[0..255] of integer
  BmGs: array[0..n-1] of integer

ALGORITMA
```

```
{ pemanggilan fungsi preBmBc }
preBmBc(n, P, BmBc)
{ pemanggilan fungsi preBmGs }
preBmGs(n, P, BmGs)
{ inisialisasi nilai i }
i:=0
while (i<= m-n) do
  j:=n-1
  while (j >=0 n and T[i+j] = P[j]) do
     j:=j-1
  endwhile
  if(j < 0) then
     ketemu[i]:=true;
  endif
  bmBcShift:= BmBc[chartoint(T[i+j])]-n+j+1
  bmGsShift:= BmGs[j]
  shift:= max(bmBcShift, bmGsShift)
  i:= i+shift
```

2.4. Regex

Ekspresi reguler (bahasa Inggris: regular expression, dipendekkan menjadi regex atau regexp; juga disebut ekspresi rasional) adalah serangkaian karakter yang mendefinisikan sebuah pola pencarian. Pola tersebut biasanya digunakan oleh algoritma pencarian string untuk melakukan operasi "cari" atau "cari dan ganti" pada string, atau untuk memeriksa string masukan. Ekspresi reguler merupakan teknik yang dikembangakan dalam bidang ilmu komputer teori dan teori bahasa formal.



***Gambar 2.4.1**. Ilustrasi Regex*
Sumber: https://i.stack.imgur.com/2I74p.png

Konsep ini muncul pada 1950-an ketika matematikawan Amerika Stephen Cole Kleene memformalkan deskripsi sebuah bahasa reguler. Konsep ini menjadi banyak digunakan untuk utilitas pengolahan teks Unix. Beberapa sintaks untuk menulis ekspresi reguler telah dibuat sejak 1980-an, salah satunya adalah standar POSIX dan yang satu lagi, yang sering digunakan, adalah sintaks Perl.

Ekspresi reguler digunakan dalam mesin pencari, dialog cari dan ganti dalam pengolah kata dan penyunting teks, dalam utilitas pengolahan teks seperti sed dan AWK dan dalam analisis leksikal. Kebanyakan bahasa pemrograman menyediakan pengolah ekspresi reguler baik secara bawaan atau melalui pustaka.

2.5. Penjelasan Singkat mengenai aplikasi Web yang dibangun

Pada tugas besar IF2211 Strategi Algoritma – 2022/2023 ini, kami membuat sebuah website ChatGPT-Lite yang menggunakan React untuk bagian front-end, Node JS digunakan untuk bagian back-end, serta

penanganan database menggunakan PostgreSQL. Kami juga memanfaatkan fitur autentikasi untuk dapat menyimpan chat-history dan room-chat tiap user. Dalam aplikasi kami, nantinya history-chat dan room-chat akan disimpan untuk tiap sesi, sehingga pengguna harus Login apabila ingin menggunakan web ChatGPT-Lite. Dengan penggunaan fitur autentikasi tersebut, akan terdapat beberapa tambahan fitur yang masih berkaitan, antara lain : penamaan akun yang akan menampilkan email dari pengguna serta foto profil pada Chat-Window-Page akan berisi foto dari akun Google pengguna. Untuk penjabaran selanjutnya akan dijelaskan pada Bab 3.2, sedangkan keseluruhan program dapat diakses pada pranala berikut ini.

BAB III
Analisis Pemecahan Masalah

3.1 Langkah-langkah Pemecahan Masalah

1. Mula-mula input user akan dipisahkan berdasarkan tanda koma.
2. Untuk setiap fitur yang diminta oleh user, akan dilakukan klasifikasi menggunakan regex dengan urutan prioritas: fitur tanggal, fitur kalkulator, fitur tambahkan pertanyaan, fitur hapus pertanyaan, dan fitur bertanya.
3. Jika klasfikasi = fitur tanggal, maka akan dilakukan pengecekan apakah tanggal tersebut valid? Jika valid, maka akan dikembalikan hari yang sesuai dari tanggal tersebut.
4. Jika klasifikasi = fitur kalkulator, maka akan dilakukan perhitungan matematis apabila persamaan yang diberikan valid.
5. Jika klasifikasi = fitur tambahkan pertanyaan, maka akan dilakukan pengecekan terlebih dahulu menggunakan “kmp” apakah pertanyaan sudah tersedia di database. Jika pertanyaan belum tersedia, maka akan memasukkan pertanyaan baru di jawaban yang dimasukkan user. Jika pertanyaan sudah tersedia, maka akan dilakukan update dengan jawaban baru yang dimasukkan oleh user.
6. Jika klasifikasi = fitur hapus pertanyaan, maka akan dilakukan pengecekan terlebih dahulu menggunakan “kmp” apakah pertanyaan sudah tersedia di database atau belum. Jika pertanyaan sudah tersedia di database, maka akan dilakukan penghapusan pertanyaan dari database. Jika belum ada maka akan mengembalikan pesan error.
7. Jika klasifikasi = fitur pertanyaan teks, maka akan dilakukan pengecekan terlebih dahulu berdasarkan algoritma yang dipilih oleh user yakni, antara kmp atau bm. Lalu juga akan dilakukan pengecekan similarity dari tiap pertanyaan yang ada di database dengan pertanyaan dari user. Jika yang exact-match dengan kmp/bm hanya 1 maka akan dikembalikan jawabannya ke user. Namun, jika tidak, maka akan dicek apakah terdapat pertanyaan dengan similarity > 90% dan dikembalikan ke user. Namun, jika tidak ada juga, maka akan dipilih 3 pertanyaan termirip yang terdapat di database dan dikembalikan ke user.
8. Pada KMP, langkah-langkahnya sebagai berikut: Fungsi borderFunction digunakan untuk membuat tabel border untuk pola yang diberikan. Tabel border digunakan untuk mengetahui pergeseran yang harus dilakukan saat tidak cocok dalam pencarian pola. Pencarian pola dimulai dengan mengecek karakter pertama pada teks dan pola. Jika sama, maka dilanjutkan pengecekan pada karakter selanjutnya pada teks dan pola. Jika karakter pada teks dan pola tidak sama, maka akan dilakukan pergeseran pada pola dengan menggunakan tabel border yang sudah dibuat di langkah pertama. Jika tidak ditemukan pola pada teks, maka pencarian akan berhenti dan mengembalikan nilai false. Jika ditemukan pola pada teks, maka pencarian akan berhenti dan mengembalikan nilai true.
9. Pada BM, langkah-langkahnya sebagai berikut: Buat sebuah fungsi untuk mencari indeks terakhir dari setiap karakter dalam pola yang dicari Inisialisasi indeks i dan j dengan nilai panjang pola - 1. Lakukan iterasi dari indeks i hingga akhir teks. Jika karakter pada indeks i sama dengan karakter pada indeks j dalam pola, kurangi nilai i dan j dengan 1. Jika karakter pada indeks i tidak sama dengan karakter pada indeks j dalam pola, periksa indeks terakhir karakter pada teks yang sama dengan karakter pada indeks i dalam pola. Pindahkan indeks i dengan jumlah langkah sejauh mana yang bisa dilakukan sehingga indeks j dalam pola selaras dengan karakter yang tidak cocok dalam

teks. Ulangi langkah tersebut terus sampai ditemukan kemunculan pola dalam teks atau teks sudah habis diperiksa.

3.2 Fitur Fungsional dan Arsitektur Aplikasi Web

3.2.1. Fitur Fungsional

Dalam tugas besar kali ini kami telah menerapkan beberapa fitur sesuai sesuai spek tugas dan beberapa fitur tambahan, antara lain:

1. Fitur Pertanyaan Teks
   Pertanyaan teks merupakan fitur sesuai spek yang diimplementasikan dengan cara mencocokkan pertanyaan dari input pengguna ke pertanyaan di database menggunakan algoritma KMP atau BM.
2. Fitur Kalkulator
   Kalkulator merupakan fitur sesuai spek yang diimplementasikan menggunakan regex. Pengguna memasukkan input query berupa persamaan matematika. Contohnya adalah 2*5 atau 5+9*(2+4). Operasi cukup Tambah, kurang, kali, bagi, pangkat, kurung.
3. Fitur Tanggal
   Tanggal merupakan fitur sesuai spek yang diimplementasikan dalam bentuk, misalnya: Pengguna memasukkan input berupa tanggal, lalu chatbot akan merespon dengan hari apa di tanggal tersebut (DD/MM/YY). Contohnya adalah 25/08/2023 maka chatbot akan menjawab dengan hari senin.
4. Fitur Tambah Pertanyaan
   Pengguna dapat menambahkan pertanyaan dan jawabannya sendiri ke database dengan query “Tambahkan pertanyan xxx dengan jawaban yyy”. Algoritma string matching digunakan untuk mencari tahu apakah pertanyaan sudah ada. Apabila sudah, maka jawaban akan diperbaharui.
5. Fitur Hapus Pertanyaan
   Pengguna dapat menghapus sebuah pertanyaan dari database dengan query contoh “Hapus pertanyaan xxx”. Algoritma string matching digunakan untuk mencari pertanyaan xxx tersebut pada database.
6. Fitur Tambah Room Chat
   Tambah Room Chat merupakan fitur tambahan diluar spek, dimana pengguna dapat menambahkan room chat baru dengan menekan tombol ‘New Chat’ pada sidebar.
7. Fitur Edit Title Room Chat
   Edit Title Room Chat merupakan fitur tambahan diluar spek, dimana pengguna dapat merubah title room chat dengan menekan ‘icon edit’ pada sidebar setelah memilih room chat tertentu.
8. Fitur Hapus Room Chat
   Hapus Room Chat merupakan fitur tambahan diluar spek, dimana pengguna dapat menghapus room chat dengan menekan ‘icon hapus’ pada sidebar setelah memilih room chat tertentu.
9. Fitur Ganti Algoritma
   Ganti algoritma merupakan fitur yang membuat pengguna dapat memilih algoritma yang akan digunakan untuk proses string matching (Boyer-Moore atau Knuth-Morris-Pratt).
10. Fitur Autentikasi
    Autentikasi merupakan fitur diluar spek yang seluruh proses akan tersimpan khusus untuk tiap akun pengguna tertentu. Fitur ini akan berdampak pada tersimpannya room chat, history chat, serta tampilan foto profil yang menyesuaikan dengan akun tiap pengguna.

Untuk contoh input valid dan tidak valid pada tiap fitur query dapat dilihat pada bab 4.4 Hasil Pengujian.

3.2.2. Arsitektur Aplikasi Web

Pada tugas besar kali ini, kelompok kami telah membuat sebuah website ChatGPT-Lite sebagai bentuk penerapan dari algoritma-algoritma string matching. Dalam pengerjaannya, kami menggunakan React untuk membangun tampilan depan (front-end) dari aplikasi ini, Node JS untuk menangani tampilan belakang (back-end), serta PostgreSQL digunakan untuk menangani database kami. Struktur proyek dari kelompok kami dapat dijelaskan di bawah ini:

```bash
├─────.next
├─────doc
├─────img
├─────node_modules
├─────prisma
├─────public
└─────src
    ├─────assets
    ├─────components
    │    ├─────chat-window
    │    │    ├─────chat-container
    │    │    ├─────chat-window-page
    │    │    ├─────input-box
    │    │    └─────send-button
    │    └─────sidebar
    │         ├─────account-select
    │         ├─────algorithm-select
    │         ├─────chat-history
    │         ├─────history-container
    │         ├─────new-chat-button
    │         └─────sidebar-page
    ├─────context
    ├─────hooks
    ├─────libs
    │    ├─────algorithms
    │    │    └─────string-matching
    │    └─────handler
    ├─────pages
    │    ├─────api
    │    │    ├─────auth
    │    │    ├─────chat
    │    │    └─────message
    │    ├─────chat
    │    └─────home
    └─────styles
```

```

- Folder .env, .eslintrc.json, .gitignore, global.d.ts, next-env.d.ts, next.config.js, package-lock.json, package.json, postcss.config.js, README.md, tailwind.config.js, dan tsconfig.json digunakan untuk menyimpan berbagai konfigurasi terkait pengembangan dan deployment website kami.
- Folder .next dan node_modules adalah folder yang dibuat oleh Next.js dan Node.js saat melakukan development dan instalasi package dependencies yang digunakan di aplikasi kami.
- Folder prisma berisi file schema.prisma yang digunakan untuk mengatur database kami dengan menggunakan ORM Prisma.
- Folder public berisi file-file yang akan diakses oleh public seperti file gambar logo dan ikon aplikasi.
- Folder src adalah folder utama yang digunakan untuk menyimpan semua source code aplikasi kami. Di dalamnya terdapat folder assets, components, context, hooks, libs, pages, dan styles. Folder assets berisi file-file yang digunakan sebagai asset di aplikasi kami. Folder components berisi semua komponen yang kami buat untuk membangun aplikasi, seperti chat window, sidebar, dan sebagainya. Folder context berisi file-file yang digunakan untuk membuat context di React, dalam hal ini context yang digunakan untuk mengatur state pada aplikasi chat kami. Folder hooks berisi file-file yang digunakan untuk membuat custom hook yang dapat digunakan kembali di seluruh aplikasi kami. Folder libs berisi file-file yang berisi fungsionalitas yang sering digunakan di seluruh aplikasi kami, seperti fetcher untuk melakukan fetch ke API, dan Prisma untuk mengatur database. Folder pages berisi file-file yang digunakan untuk membuat halaman di aplikasi kami, termasuk file untuk membuat API yang akan dipanggil oleh front-end. Folder styles berisi file CSS yang digunakan untuk styling seluruh aplikasi kami. Dengan struktur folder seperti ini, kami dapat mempermudah pengembangan dan pemeliharaan aplikasi kami.
```

BAB IV
Implementasi dan Pengujian

4.1 Implementasi Program

4.1.1. Implementasi Boyer Moore

```
Function bm(input text: string, input pattern: string) -> output boolean
{Mengembalikan true jika pattern ditemukan dalam text, dan false jika tidak}

KAMUS
 text, pattern: string

 {fungsi antara}
 procedure lastOccurrenceFunction(input pattern: string) -> array of integer
 {Mengembalikan array lastOccurrence, yaitu array yang berisi indeks terakhir dari setiap   karakter
pada pattern yang muncul dalam pattern}

ALGORITMA
 lastOccurrence <- lastOccurrenceFunction(pattern)
 if (pattern.length == 0)
    return true
 i <- pattern.length - 1
 j <- pattern.length - 1

 while (i < text.length) do
  if (text[i] == pattern[j]) then
   if (j == 0) then
    return true
   end if
   i <- i - 1
   j <- j - 1
  else
   i <- i + pattern.length - Math.min(j, 1 + lastOccurrence[text[i].charCodeAt(0)])
   j <- pattern.length - 1
  end if
 end while

 return false
```

4.1.2. Implementasi Knuth-Morris-Pratt

```
function kmp(text: string, pattern: string) -> boolean
   {melakukan pencarian pola pattern pada sebuah string text menggunakan      algoritma
Knuth-Morris-Pratt (KMP). Fungsi ini akan mengembalikan nilai true jika pattern ditemukan
dalam text dan false jika sebaliknya}

KAMUS
   text: string, teks yang akan dicari pola
   pattern: string, pola yang akan dicari dalam text
   borderTable: array of integer, menyimpan nilai border function dari pattern
   i: integer, indeks saat ini pada text
   j: integer, indeks saat ini pada pattern

ALGORITMA

   borderTable <- borderFunction(pattern)
   i <- 0
   j <- 0

   while (i < length(text)) do
       if (text[i] == pattern[j]) then
          if (j == length(pattern) - 1) then
           return true
          end if
```

```
          i <- i + 1
          j <- j + 1
       else if (j > 0) then
          j <- borderTable[j - 1]
       else
          i <- i + 1
       end if
   end while

   return false
```

4.1.3. Similarity Check

```
Function levenshteinDistance(str1: string, str2: string) -> number

  matrix : array of array of number

  {inisialisasi}
  for i from 0 to length of str2
    matrix[i][0] ← i
  end for

  for j from 0 to length of str1
    matrix[0][j] ← j
  end for


  for i from 1 to length of str2
    for j from 1 to length of str1
      if str2[i - 1] = str1[j - 1] then
        matrix[i][j] ← matrix[i - 1][j - 1]
      else
        matrix[i][j] ← minimum of matrix[i - 1][j - 1], matrix[i - 1][j], and
matrix[i][j - 1] + 1
      end if
    end for
  end for

  return matrix[length of str2][length of str1]

End Function

Function similarityCheck(input str1: string, input str2: string) -> number
    If (str1.length > str2.length)
        longer <- str1
    else
      longer <- str2

    if (longer.length == 0)
       return 1.0
    else

       return (longer.length - levenshteinDistance(str1, str2)) /  parseFloat(
longer.length.toString())
```

4.1.4. Calculator

```
Function calculator (str: string) -> string
  str = str.split(" ").join("");
   equation = str.match(/([\d+\-*/()^]+)/);
  try {
```

```
    res = eval(equation![0].split("^").join("**"));
    return `${equation![0]} = ${res}`;
  } catch (e) {
    return "Sintaks persamaan tidak sesuai!";
  }
```

4.1.5. Date

```
  Function date (text: string) -> string
    date = text.match(/\b\d{2}\/\d{2}\/\d{4}\b/g);
    splitDate = date![0].split("/");

   if (
     !isValidDate(
       parseInt(splitDate[2]),
       parseInt(splitDate[1]) - 1,
       parseInt(splitDate[0])
     )
   )
     return "Tanggal tidak valid!";

    day = new Date(
     parseInt(splitDate[2]),
     parseInt(splitDate[1]) - 1,
     parseInt(splitDate[0])
   ).toLocaleDateString("id-ID", { weekday: "long" })

   return `Tanggal ${date![0]} adalah hari ${day}`;

 Fucntion isValidDate (year: number, month: number, day: number) -> boolean
   var date = new Date(year, month, day)
   if (
     date.getFullYear() == year &&
     date.getMonth() == month &&
     date.getDate() == day
   ) {
     return true
   }
   return false
```

4.1.6. Regex

```
       Function regex = (str: string)-> string
        str = str.toLowerCase();

        if (
          str.match(/^.*(\d{2}\/\d{2}\/\d{4}).*$/) !== null) {
          return "date";
        } else if (
          str.match(/^.*[0-9]+\s*[\+\-\*\^\/\(\)]\s*[0-9\+\-\*\^\/\(\)\s]*.*$/)
      !== null) {
          return "calculator";
        } else if (

      str.match(/^.*(tambahkan|tambah)\s*pertanyaan\s*.+\s*dengan\s*jawaban\s*.+$
      /) !== null) {
          return "add";
        } else if (
          str.match(/^.*hapus\s*pertanyaan\s*.+$/) !== null) {
          return "delete";
        } else {
          return "ask";
```

```
        }
const regex = (str: string): string => {
  str = str.toLowerCase();

  if (
    str.match(/^.*(\d{2}\/\d{2}\/\d{4}).*$/) !== null) {
    return "date";
  } else if (
    str.match(/^.*[0-9]+\s*[\+\-\*\^\/\(\)]\s*[0-9\+\-\*\^\/\(\)\s]*.*$/) !== null) {
    return "calculator";
  } else if (
    str.match(/^.*(tambahkan|tambah)\s*pertanyaan\s*.+\s*dengan\s*jawaban\s*.+$/) !== null) {
    return "add";
  } else if (
    str.match(/^.*hapus\s*pertanyaan\s*.+$/) !== null) {
    return "delete";
  } else {
    return "ask";
  }
};
```

4.2 Penjelasan Struktur Data

Berikut merupakan struktur data beserta spesifikasi program yang digunakan dalam membuat program:

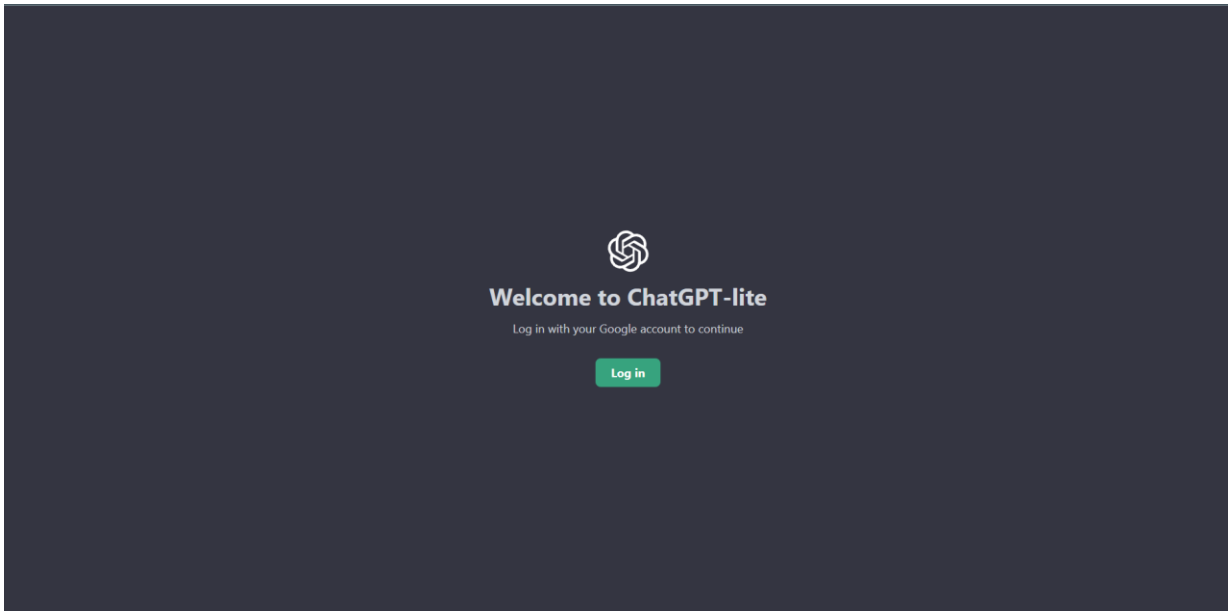
1. List
   Implementasi struktur data list digunakan dalam menyimpan semua percakapan pada setiap pengguna dan juga menyimpan semua pesan dalam setiap percakapan.
2. Matriks
   Struktur data matriks digunakan untuk menyimpan nilai pada file levenshteinDistance.tsx

4.3 Penjelasan Tata Cara Penggunaan Program

Secara garis besar program terdiri atas 2 halaman yaitu *homepage* dan juga *chatpage*.

4.3.1. Homepage

Pada halaman homepage akan menampilkan nama program dan juga tombol login. Pengguna dapat menekan tombol “Log In”. kemudian pengguna akan diarahkan untuk memilih akun apa yang ingin ia gunakan saat memakai chat-gpt-lite ini. Program ini hanya melayani penggunaan akun Google, sehingga jika pengguna tidak memiliki akun Goggle maka pengguna harus membuat akun Google terlebih dahulu. Setelah proses autentikasi berhasil, program akan menampilkan chatpage.



Gambar 4.3.1.1*. Homepage*
Sumber: Dokumen Pribadi Penulis

4.3.2. Chatpage

Ini merupakan fitur utama program. Pada halaman ini terdapat dua bagian besar yaitu sidebar dan juga chatwindow. Chat window merupakan bagian utama dari halaman ini, dibagian ini pesan dari pengguna dimuat sedangkan Sidebar merupakan bagian yang berisi fitur-fitur pendukung.

4.3.2.1. Window Page

Pengguna dapat mengirim pesan ke program pada bagian ini melalui input box yang berada pada bagian bawah. Pengguna dapat mengetikkan pesan berupa teks yang dapat dikirim menggunakan tombol enter ataupun tombol pesawat yang berada di sebelah kanan input box, sedangkan jika ingin membuat baris baru pengguna dapat menggunakan shift+enter.. Setelah pesan terkirim, pengguna harap menunggu pesan balasan dari program **(waktu balasan bisa jadi menjadi sedikit lama jika *server sedang lambat , oleh karena itu harap menunggu).*** Setelah pesan balasan dari program tampil di layar, pengguna dapat memberikan pertanyaan lain pada program.

4.3.2.2. Sidebar

Bagian ini berisi fitur-fitur yang mendukung program utama yaitu mengirim pesan yang dari atas ke bawah terdiri dari:

- Tombol “New Chat” untuk membuat pesan baru
- Daftar percakapan pengguna yang dapat ditekan salah satunya untuk melanjutkan percakapan. Pengguna juga dapat mengganti judul percakapan dengan menekan tombol “pena” ataupun menghapus percakapan dengan menekan tombol “tempat sampah”
- Tombol untuk mengganti algoritma yang ingin digunakan

- Tombol yang menampilkan akun yang sedang digunakan oleh pengguna saat itu dan jika ditekan, pengguna dapat memilih untuk keluar ataupun mengganti akun yang ingin digunakan.

Gambar 4.3.2.2.1. *Chatpage*
Sumber:Dokumen Pribadi Penulis

4.4 Hasil Pengujian

4.4.1. Query Pencocokan ke Database (KMP dan BM)

Pengujian pencocok string menggunakan kedua algoritma, pengujian meliputi berbagai kondisi yaitu jika pesan masukan sama persis dengan database, typo, ataupun beda jauh dengan database

Gambar 4.4.1.1. *Query Pencocokan ke Database dengan KMP*
Sumber:Dokumen Pribadi Penulis

Gambar 4.4.1.2. *Query Pencocokan ke Database dengan BM*
Sumber:Dokumen Pribadi Penulis

4.4.2. Query Kalkulator

***Gambar 4.4.2.1**. Query Kalkulator Valid*
Sumber:Dokumen Pribadi Penulis

***Gambar 4.4.2.2**. Query Kalkulator Tidak Valid*
Sumber:Dokumen Pribadi Penulis

4.4.3. Query Menanyakan hari apa pada tanggal

Gambar 4.4.3.1. *Query Tanggal*
Sumber:Dokumen Pribadi Penulis

4.4.4. Query Menambah pertanyaan

Gambar 4.4.4.1. *Query Menambah Pertanyaan Valid*
Sumber:Dokumen Pribadi Penulis

***Gambar 4.4.4.2**. Query Menambah Pertanyaan Tidak Valid*
Sumber:Dokumen Pribadi Penulis

4.4.5. Query Menghapus pertanyaan

***Gambar 4.4.5.1**. Query Menghapus Pertanyaan*
Sumber:Dokumen Pribadi Penulis

4.5 Analisis Hasil Pengujian

Algoritma KMP (Knuth-Morris-Pratt), BM (Boyer-Moore), dan Regex (Regular Expression) merupakan algoritma yang sering digunakan dalam string matching atau pencocokan pola dalam sebuah teks. Algoritma KMP bekerja dengan cara memanfaatkan sebuah tabel prefix, yaitu tabel yang menyimpan informasi terkait dengan prefix suatu string, sehingga kita dapat memanfaatkan informasi ini untuk menghindari pencocokan kembali pada bagian-bagian yang sebelumnya telah cocok. Keuntungan dari algoritma ini adalah waktu eksekusi yang relatif konstan, yaitu O(m+n) di mana m adalah panjang string pola dan n adalah panjang string teks.

Algoritma BM juga menggunakan strategi penghindaran pencocokan kembali pada bagian-bagian yang sudah dicocokkan sebelumnya, namun memanfaatkan tabel skip dan tabel good suffix. Keuntungan dari algoritma ini adalah waktu eksekusi yang efektif dan efisien dengan kondisi terbaik yang O(n/m) di mana n adalah panjang string teks dan m adalah panjang string pola. Sementara itu, Regex merupakan sebuah pola pencocokan dalam sebuah teks yang digunakan untuk mencari kata atau pola tertentu. Penggunaan Regex sangat populer dalam text editor, web scraping, dan manipulasi string. Keuntungan dari algoritma ini adalah fleksibilitasnya yang tinggi dan dapat menghandle berbagai pola pencocokan yang rumit.

Dalam penggunaan algoritma string matching, perbandingan tingkat efektivitas ketiga algoritma tergantung pada kasus penggunaan dan jenis data yang akan diolah. Jika data yang diolah memiliki pola pencocokan yang sederhana, maka penggunaan regex akan cukup efektif. Hal ini terlihat bahwa, regex dalam tugas besar ini diterapkan pada penanganan kasus operasi aritmatika yang tergolong cukup sederhana. Akan tetapi, jika data yang diolah memiliki pola pencocokan yang lebih kompleks dan panjang, maka penggunaan algoritma KMP atau BM akan lebih efektif. Oleh karena itu, dalam penyelesaian masalah pencocokan string pada masukan dan database digunakan kedua algoritma tersebut. Namun, pada umumnya algoritma BM dianggap lebih efektif dan memberikan hasil terbaik dalam hal kecepatan dan efisiensi dibandingkan dengan algoritma KMP dan regex. Oleh karena itu, algoritma BM menjadi pilihan yang lebih disukai dalam aplikasi yang memerlukan proses pencocokan pola pada data yang besar. Hal ini juga dapat terlihat dari beberapa ujicoba yang kami lakukan pada program.

BAB V
Kesimpulan Saran dan Refleksi

5.1 Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma semester 2 2022/2023 berjudul "Penerapan String Matching dan Regular Expression dalam Pembuatan ChatGPT Sederhana", kami berhasil membuat Aplikasi berbasis Web yang dapat melakukan sebuah interaksi selayaknya ChatGPT dengan memanfaatkan konsep string matching dengan algoritma KMP dan BM. Kami juga menambahkan berhasil mengaplikasiskan beberapa fitur lain seperti penentuan jenis hari, kalkulator, serta penambahan dan penghapusan pertanyaan dari database.

Kami juga telah berhasil memanfaatkan berbagai macam TechStack yang tersedia untuk membangun aplikasi berbasis web, seperti : React, Node JS, serta PostGreSQL. Kami menggunakan React untuk bagian frontend serta Node JS untuk bagian backend-nya.

Kesimpulan yang kami dapatkan dari pengerjaan tugas besar ini, antara lain:

- Algoritma KMP dan BM dapat digunakan sebagai strategi string matching untuk beberapa kasus penerapan tertentu. Sebagai contoh adalah pembuatan ChatGPT sederhana, seperti yang telah kami lakukan pada tugas kali ini.

5.2 Saran

Pengembangan sebuah aplikasi berbasis web memerlukan pengerjaan intens serta pertimbangan yang mendalam. Kami harus melakukan eksplorasi pada dokumentasi untuk tiap TechStack yang kami gunakan, misalnya React untuk bagian Frontend dan Node JS untuk bagian Backend. Selain itu, kami juga dituntut mempelajari bahasa pemrograman baru (dalam hal ini adalah typescript) serta melakukan penerapan terhadap algoritma KMP dan BM dalam pemecahan permasalahan string matching. Dengan mempertimbangkan banyaknya hal yang harus dieksplorasi, sebaiknya deadline pengerjaan tugas besar yang serupa untuk kedepannya dapat diperpanjang. Walaupun deadline tugas besar kali ini diberikan rentang waktu 3 minggu, tetapi itu sudah termasuk masa libur lebaran selama 1 minggu, sehingga secara efektif tugas ini hanya diberikan rentang waktu pengerjaan kurang dari 2 minggu. Oleh karena itu, kami menyarankan untuk mempertimbangkan hal-hal tersebut untuk tugas besar serupa kedepannya.

5.3 Refleksi

Refleksi dari kelompok kami mungkin terkait komunikasi dan sistem pengerjaan yang dilakukan selama rentang waktu tugas besar ini. Dari segi komunikasi, mungkin seharusnya dilakukan lebih sering baik secara offline maupun online, agar komunikasi dapat terjalin dengan lebih baik lagi.

5.4 Tanggapan

Tugas Besar 3 IF2211 – Strategi Algoritma 2022/2023 cukup menantang. Hal ini disebabkan oleh kami yang dipacu untuk mempelajari bahasa pemrograman typescript, membaca dokumentasi beberapa TechStack baru, serta melakukan eksplorasi terhadap algoritma string matching yang sebelumnya belum pernah dipelajari secara mendalam. Selain itu, kami juga didorong untuk melakukan pengerjaan tugas besar ini dalam rentang waktu yang tergolong cukup pendek.

Referensi

Munir, Rinaldi, Nur Ulfa Mauladevi. 2023, “Diktat Pencocokan String (String/Pattern Matching)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf, diakses pada 2 Mei 2023.

Munir, Rinaldi, Nur Ulfa Mauladevi. 2023, “Diktat String Matching dengan Regular Expression”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf, diakses pada 2 Mei 2023.

React. “React Documentation”, https://react.dev/learn/, diakses pada 2 Mei 2023.

NodeJS. “Node JS Documentation”, https://nodejs.org/en/docs, diakses pada 2 Mei 2023.

PostgreSQL. “PostgreSQL Documentation”, https://www.postgresql.org/docs/, diakses pada 2 Mei 2023.

Tailwind CSS. “Tailwind CSS Documentation”, https://tailwindcss.com/docs/installation, diakses pada 2 Mei 2023.

GeekForGeeks. “KMP Algorithm for Pattern Searching”, https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/, diakses pada 2 Mei 2023.

GeekForGeeks. “Boyer Moore Algorithm for Pattern Searching”, https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/, diakses pada 2 Mei 2023.

GeekForGeeks. “How to write Regular Expressions”, https://www.geeksforgeeks.org/write-regular-expressions/, diakses pada 2 Mei 2023.

OpenAI. “ChatGPT Website”, https://chat.openai.com/, diakses pada 2 Mei 2023.

Munir, Rinaldi. 2023, “Tugas besar 3: Penerapan String Matching dan Regular Expression dalam Pembuatan ChatGPT Sederhana”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes3-Stima-2023.pdf, diakses pada 2 Mei 2023.

Pranala Terkait

Link Repository: https://github.com/briankheng/Tubes3_13521049/tree/frontend
Link Video : bit.ly/VideoDemoTubes3StimaBaby
Link Deploy : https://chatgpt-lite-alpha.vercel.app/