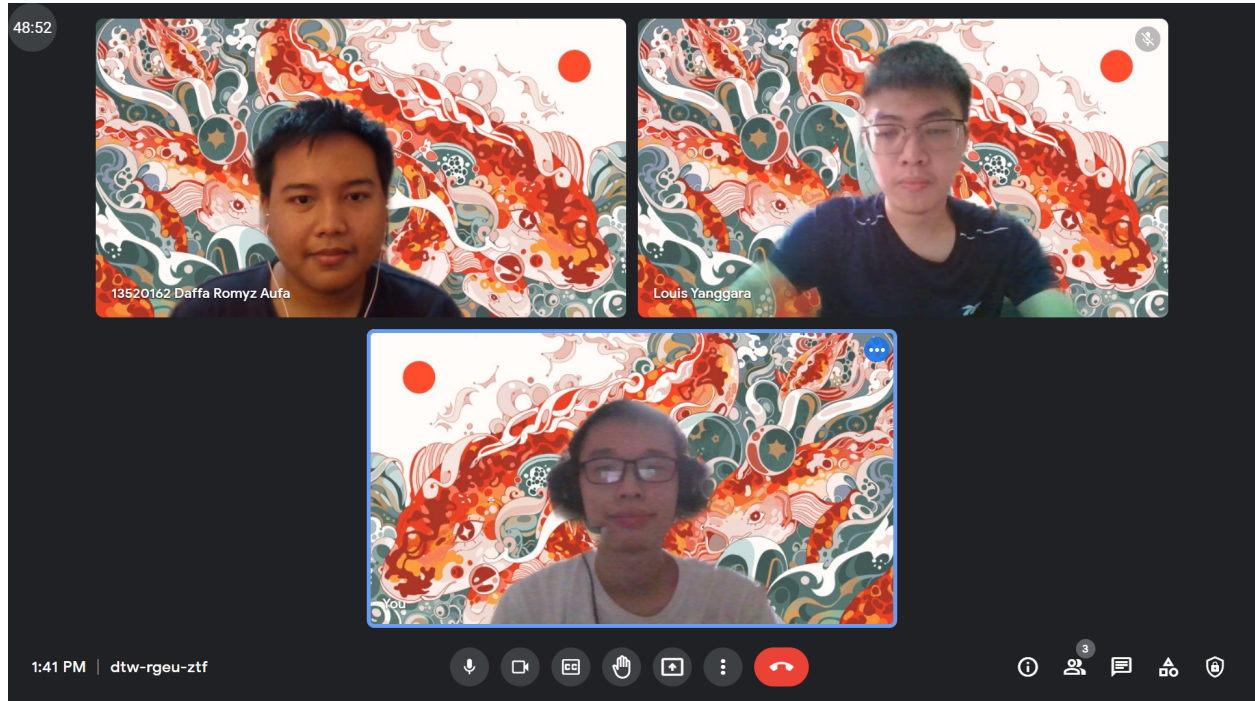


# LAPORAN TUGAS BESAR 3

## IF2211 Strategi Algoritma

### Penerapan String Matching dan Regular Expression dalam DNA Pattern Matching



Disusun oleh:

Kelompok 20

1. Louis Yanggara (13520063)
2. Johannes Winson Sukiatmodjo (13520123)
3. Daffa Romyz Aufa (13520162)

PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2022

# DAFTAR ISI

<b>DESKRIPSI TUGAS</b>	<b>4</b>
1.1 Latar belakang	4
1.2 Deskripsi tugas	5
1.3 Fitur-fitur aplikasi	5
1.4 Spesifikasi program	10
<b>LANDASAN TEORI</b>	<b>11</b>
2.1 Algoritma KMP, BM, dan Regex	11
2.1.1 Algoritma KMP	11
2.1.2 Algoritma BM	12
2.1.3 Regex	12
2.2 Gambaran umum aplikasi web	14
<b>ANALISIS PEMECAHAN MASALAH</b>	<b>15</b>
3.1 Langkah penyelesaian masalah	15
3.2 Fitur fungsional dan arsitektur aplikasi web	16
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>19</b>
4.1 Spesifikasi teknis program	19
4.2 Tata cara penggunaan program	19
4.3 Hasil pengujian beserta analisisnya	20
<b>PENUTUP</b>	<b>23</b>
5.1 Kesimpulan	23
5.2 Saran	23
5.3 Refleksi	23
<b>DAFTAR PUSTAKA</b>	<b>24</b>

## DAFTAR GAMBAR

**Gambar 1.** Ilustrasi Sekuens DNA

**Gambar 2.** Ilustrasi Input Penyakit

**Gambar 3.** Ilustrasi Prediksi

**Gambar 4.** Ilustrasi Interaksi 1

**Gambar 5.** Ilustrasi Interaksi 2

**Gambar 6.** Ilustrasi Interaksi 3

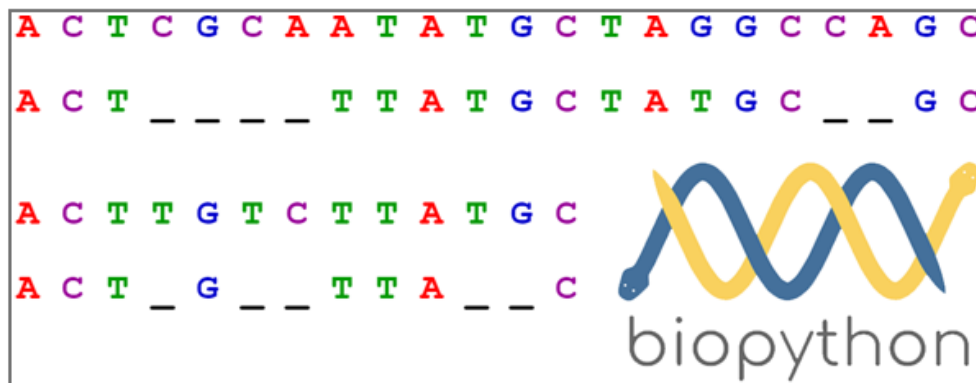
**Gambar 7.** Ilustrasi Bonus

# BAB I

## DESKRIPSI TUGAS

### 1.1 Latar belakang

Manusia umumnya memiliki 46 kromosom di dalam setiap selnya. Kromosom-kromosom tersebut tersusun dari DNA (*deoxyribonucleic acid*) atau asam deoksiribonukleat. DNA tersusun atas dua zat basa purin, yaitu Adenin (A) dan Guanin (G), serta dua zat basa pirimidin, yaitu sitosin (C) dan timin (T). Masing-masing purin akan berikatan dengan satu pirimidin. DNA merupakan materi genetik yang menentukan sifat dan karakteristik seseorang, seperti warna kulit, mata, rambut, dan bentuk wajah. Ketika seseorang memiliki kelainan genetik atau DNA, misalnya karena penyakit keturunan atau karena faktor lainnya, ia bisa mengalami penyakit tertentu. Oleh karena itu, tes DNA penting untuk dilakukan untuk mengetahui struktur genetik di dalam tubuh seseorang serta mendeteksi kelainan genetik. Ada berbagai jenis tes DNA yang dapat dilakukan, seperti uji pra implantasi, uji pra kelahiran, uji pembawa atau *carrier testing*, uji forensik, dan *DNA sequence analysis*.



**Gambar 1.** Ilustrasi Sekuens DNA

Sumber:

<https://towardsdatascience.com/pairwise-sequence-alignment-using-biopython-d1a9d0ba861f>

Salah satu jenis tes DNA yang sangat berkaitan dengan dunia bioinformatika adalah *DNA sequence analysis*. *DNA sequence analysis* adalah sebuah cara yang dapat digunakan untuk memprediksi berbagai macam penyakit yang tersimpan pada database berdasarkan urutan sekuens DNA-nya. Sebuah sekuens DNA adalah suatu representasi *string of nucleotides* yang disimpan pada suatu rantai DNA, sebagai contoh: ATTCGTAAGTAAAGTTA. Teknik *pattern matching* memegang peranan penting untuk dapat menganalisis sekuens DNA yang sangat panjang dalam waktu singkat. Oleh karena itu, mahasiswa Teknik Informatika berniat untuk membuat suatu aplikasi web berupa *DNA Sequence Matching* yang menerapkan algoritma String Matching dan Regular Expression untuk membantu penyedia jasa kesehatan

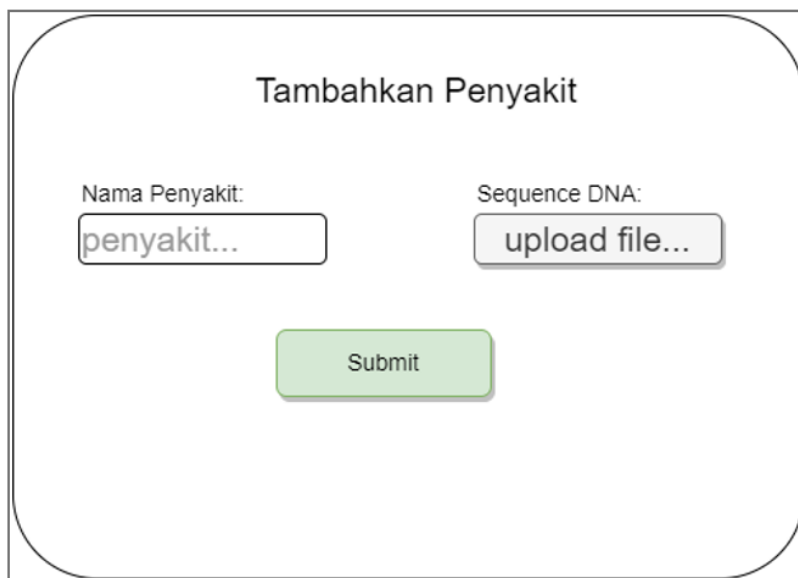
dalam memprediksi penyakit pasien. Hasil prediksi juga dapat ditampilkan dalam tabel dan dilengkapi dengan kolom pencarian untuk membantu admin dalam melakukan *filtering* dan pencarian.

## 1.2 Deskripsi tugas

Dalam tugas besar ini, anda diminta untuk membangun sebuah aplikasi *DNA Pattern Matching*. Dengan memanfaatkan algoritma *String Matching* dan *Regular Expression* yang telah anda pelajari di kelas IF2211 Strategi Algoritma, anda diharapkan dapat membangun sebuah aplikasi interaktif untuk mendeteksi apakah seorang pasien mempunyai penyakit genetik tertentu. Hasil prediksi tersebut dapat disimpan pada basis data untuk kemudian dapat ditampilkan berdasarkan query pencarian.

## 1.3 Fitur-fitur aplikasi

1. Aplikasi dapat menerima *input* penyakit baru berupa nama penyakit dan *sequence* DNA-nya (dan dimasukkan ke dalam *database*).
  - a. Implementasi *input sequence* DNA dalam bentuk *file*.
  - b. Dilakukan sanitasi *input* menggunakan **regex** untuk memastikan bahwa masukan merupakan *sequence* DNA yang valid (tidak boleh ada huruf kecil, tidak boleh ada huruf selain AGCT, dan tidak ada spasi).
  - c. Contoh *input* penyakit:



The image shows a web form titled "Tambahkan Penyakit" (Add Disease). It contains two input fields: "Nama Penyakit:" (Disease Name) with a placeholder text "penyakit..." and "Sequence DNA:" with a placeholder text "upload file...". Below these fields is a green "Submit" button.

**Gambar 2.** Ilustrasi Input Penyakit

2. Aplikasi dapat memprediksi seseorang menderita penyakit tertentu berdasarkan *sequence* DNA-nya.

- Tes DNA dilakukan dengan menerima input nama pengguna, *sequence* DNA pengguna, dan nama penyakit yang diuji. Asumsi *sequence* DNA pengguna > *sequence* DNA penyakit.
- Dilakukan sanitasi *input* menggunakan **regex** untuk memastikan bahwa masukan merupakan *sequence* DNA yang valid (tidak boleh ada huruf kecil, tidak boleh ada huruf selain AGCT, tidak ada spasi, dll).
- Pencocokan *sequence* DNA dilakukan dengan menggunakan algoritma **string matching**.
- Hasil dari tes DNA berupa tanggal tes, nama pengguna, nama penyakit yang diuji, dan status hasil tes. **Contoh: 1 April 2022 - Mhs IF - HIV - False**
- Semua komponen hasil tes ini dapat ditampilkan pada halaman web (*refer* ke poin 3 pada “Fitur-Fitur Aplikasi”) dan disimpan pada sebuah tabel *database*.
- Contoh tampilan web:

**Tes DNA**

Nama Pengguna:

Sequence DNA:

Prediksi Penyakit:

---

**Hasil Tes**

<Tanggal> - <pengguna> - <penyakit> - <True/False>

**Gambar 3. Ilustrasi Prediksi**

- Aplikasi memiliki halaman yang menampilkan urutan hasil prediksi dengan kolom pencarian di dalamnya. Kolom pencarian bekerja sebagai *filter* dalam menampilkan hasil.
  - Kolom pencarian dapat menerima masukan dengan struktur: <tanggal\_prediksi><spasi><nama\_penyakit>, contoh “13 April 2022 HIV”. **Format penanggalan dibebaskan**, jika bisa menerima >1 format lebih baik.
  - Kolom pencarian dapat menerima masukan hanya tanggal ataupun hanya nama penyakit. Fitur ini diimplementasikan menggunakan **regex**.
  - Contoh ilustrasi:
    - Masukan tanggal dan nama penyakit

13 April 2022 HIV

1. 13 April 2022 - Fulan - HIV - True.

2. 13 April 2022 - Kamal - HIV - False.

3. 13 April 2022 - Entah - HIV - False.

4. 13 April 2022 - Jamal - HIV - True.

5. 13 April 2022 - Yubai - HIV - True.

6. 13 April 2022 - Hika - HIV - False.

**Gambar 4.** *Ilustrasi Interaksi 1*

- ii. Masukan hanya tanggal

13 April 2022

1. 13 April 2022 - Fulan - Diabetes - True.

2. 13 April 2022 - Kamal - Sinusitis - False.

3. 13 April 2022 - Entah - Down Syndrome - False.

4. 13 April 2022 - Jamal - Polio - True.

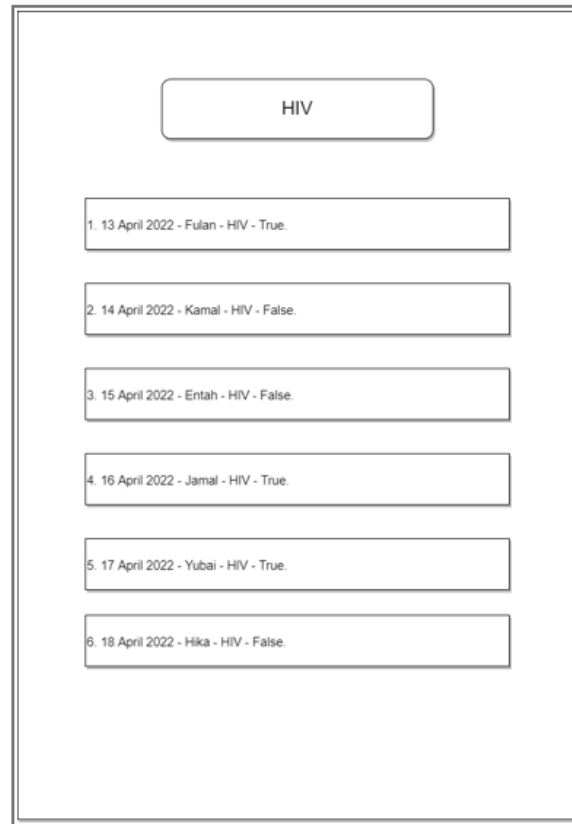
5. 13 April 2022 - Yubai - TBC - True.

6. 13 April 2022 - Hika - Hepatitis A - False.

**Gambar 5. Ilustrasi Interaksi 2**

- iii. Masukan hanya nama penyakit





**Gambar 6. Ilustrasi Interaksi 3**

4. **(Bonus)** Menghitung tingkat kemiripan DNA pengguna dengan DNA penyakit pada tes DNA
- Ketika melakukan tes DNA, terdapat persentase kemiripan DNA dalam hasil tes. **Contoh hasil tes: 1 April 2022 - Mhs IF - HIV - 75% - False**
  - Perhitungan tingkat kemiripan dapat dilakukan dengan menggunakan Hamming distance, Levenshtein distance, LCS, atau algoritma lainnya (dapat dijelaskan dalam laporan).
  - Tingkat kemiripan DNA dengan nilai lebih dari atau sama dengan 80% dikategorikan sebagai **True**. Perlu diperhatikan mengimplementasikan atau tidak mengimplementasikan bonus ini tetap dilakukan pengecekan *string matching* terlebih dahulu.
  - Contoh tampilan:

**Tes DNA**

Nama Pengguna:

Sequence DNA:

Prediksi Penyakit:

---

Hasil Tes

<Tanggal> - <pengguna> - <penyakit> - <similarity> - <True/False>

**Gambar 7. Ilustrasi Bonus**

## 1.4 Spesifikasi program

1. Aplikasi berbasis website dengan pembagian Frontend dan Backend yang jelas.
2. Implementasi Backend **wajib** menggunakan Node.js / Golang, sedangkan Frontend **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) **wajib** diimplementasikan pada sisi Backend aplikasi.
5. Informasi yang **wajib** disimpan pada basis data:
  - a. Jenis Penyakit:
    - i. Nama penyakit
    - ii. Rantai DNA penyusun.
  - b. Hasil Prediksi:
    - i. Tanggal prediksi
    - ii. Nama pasien
    - iii. Penyakit prediksi
    - iv. Status terprediksi.
6. Jika mengerjakan bonus tingkat kemiripan DNA, simpan hasil tingkat kemiripan tersebut pada basis data.

## BAB II

### LANDASAN TEORI

#### 2.1 Algoritma KMP, BM, dan Regex

##### 2.1.1 Algoritma 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, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977.

Jika kita melihat algoritma brute force bertambah mendalam, kita mengetahui bahwa dengan mengingat sebagian perbandingan yang dilakukan sebelumnya kita dapat meningkatkan akbar pergeseran yang dilakukan. Hal ini akan menghemat perbandingan, yang selanjutnya akan meningkatkan kecepatan pencarian.

Aturan penggeseran pada algoritma ini adalah sebagai berikut, bila terjadi ketidakcocokkan pada saat pattern sejajar dengan teks[ $i..i+n-1$ ], kita bisa mengasumsikan ketidakcocokan pertama terjadi di selang teks[ $i+j$ ] dan pattern[ $j$ ], dengan  $0 < j < n$ . Berarti, teks[ $i..i+j-1$ ] = pattern[ $0..j-1$ ] dan  $a = \text{teks}[i+j]$  selisih dengan  $b = \text{pattern}[j]$ . Saat kita menggeser, sangat beralasan bila mempunyai sebuah awalan  $v$  dari pattern akan sama dengan sebagian imbuhan belakang  $u$  dari sebagian teks. Sehingga kita bisa menggeser pattern supaya awalan  $v$  tersebut sejajar dengan imbuhan belakang dari  $u$ .

Dengan kata lain, pencocokan string akan berjalan secara efisien bila kita mempunyai tabel yang menentukan berapa panjang kita seharusnya menggeser seandainya terdeteksi ketidakcocokkan 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  $j - \text{next}[j]$  relatif terhadap teks.

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

- Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
- Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang berpadanan, hingga salah satu kondisi berikut dipenuhi:
  - Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
  - Seluruh karakter di pattern cocok. Belakang algoritma akan memberitahukan penemuan di posisi ini.
- Algoritma belakang menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 hingga pattern mempunyai di ujung teks.

Algoritma ini menemukan seluruh kemunculan dari pattern dengan panjang  $n$  di dalam teks dengan panjang  $m$  dengan kompleksitas waktu  $O(m+n)$ . Algoritma ini hanya membutuhkan  $O(n)$  ruang dari memori internal jika teks dibaca dari file eksternal. Seluruh besaran  $O$  tersebut tidak tergantung pada akbarnya ruang alfabet.

### 2.1.2 Algoritma BM

Algoritma Boyer-Moore yaitu 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 dibalik algoritma ini yaitu bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan banyak informasi yang didapat.

Misalnya mempunyai sebuah usaha pencocokan yang terjadi pada teks[ $i..i+n-1$ ], dan anggap ketidakcocokan pertama terjadi di selang teks[ $i+j$ ] dan pattern[ $j$ ], dengan  $0 < j < n$ . Berarti, teks[ $i+j+1..i+n-1$ ] = pattern[ $j+1..n-1$ ] dan  $a = \text{teks}[i+j]$  tidak sama dengan  $b = \text{pattern}[j]$ . Jika  $u$  yaitu imbuhan belakang dari pattern sebelum  $b$  dan  $v$  yaitu sebuah awalan dari pattern, maka penggeseran-penggeseran yang mungkin adalah:

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

Secara sistematis, langkah-langkah yang dimainkan algoritma Boyer-Moore pada masa mencocokkan string adalah:

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

Tabel bagi penggeseran bad-character dan good-suffix dapat dihitung dengan kompleksitas waktu dan ruang sebesar  $O(n + \sigma)$  dengan  $\sigma$  yaitu akbar ruang alfabet. Sedangkan pada fase pencarian, algoritma ini membutuhkan waktu sebesar  $O(mn)$ , pada kasus terburuk, algoritma ini akan memainkan  $3n$  pencocokan karakter, namun pada performa terbaiknya algoritma ini hanya akan memainkan  $O(m/n)$  pencocokan.

### 2.1.3 Regex

Text dapat diartikan sebagai rentetan dari karakter-karakter yang memiliki arti. Salah satu proses dalam memahami/mengolah isi text dalam sebuah dokumen adalah dengan memproses karakter-karakter tersebut. Contohnya adalah fitur spell-checker yang tersedia pada Microsoft Word dimana dapat membantu memberikan penanda untuk kata-kata yang salah ketik, atau tidak sesuai format, seperti tanda titik atau koma yang jika diberikan spasi maka

pembenaran yang dilakukan adalah dengan menghapus spasi tersebut. Pengecekan tersebut berada pada level karakter.

Demikian juga untuk pencarian sebuah kata/pola tertentu pada sebuah dokumen text dapat dilakukan dengan menggunakan Regular Expression. Regular Expression (RE) adalah sebuah notasi yang dapat digunakan untuk mendeskripsikan pola dari kata yang ingin dicari. Sebagai contoh jika RE yang dibuat adalah « nlp » maka kata yang akan cocok dengan pola ini hanya kata nlp (sama persis dengan yang ada pada RE). Namun demikian regular expression juga menyediakan beberapa special karakter yang dapat digunakan untuk mencocokkan karakter dengan pola-pola tertentu. Perlu diingat bahwa RE adalah case sensitive.

1. Wildcard

Simbol titik “.” disebut dengan wildcard dimana tanda ini dapat cocok dengan satu karakter apapun. Contohnya jika RE « n.p » maka kata yang akan cocok dengan RE tersebut bisa berupa nlp, nap, nup, n3p, n0p, dst.

2. Optionality

Dengan menggunakan symbol tanda tanya “?” untuk menandakan bahwa regular expression yang diberikan sebelum symbol tersebut bersifat optional. Contohnya adalah RE « colou?r » maka kata yang dapat cocok dengan pola tersebut adalah colour dan color (dimana u bersifat opsional). Sama juga dengan RE « e-?mail » dimana kata yang dapat cocok adalah e-mail dan email.

3. Repeatability

Ada dua jenis perulangan yang dapat digunakan, yaitu dengan menggunakan tanda “+” dan “\*”. Tanda “+” menandakan bahwa RE yang diberikan sebelum symbol tersebut dapat diulang. Sebagai contoh « coo+l » dapat cocok dengan kata cool, coool, coool, dst. Contoh lainnya adalah dalam RE « .+ed » sama dengan mencari kata yang berakhiran dengan kata “ed”. Sedangkan tanda “\*” menandakan bahwa RE yang diberikan sebelum symbol tersebut bersifat opsional atau dapat berulang. Jika menggunakan contoh RE diatas yaitu « coo\*l » maka kata yang dapat cocok adalah kata col, cool, coool, dst. Contoh lainnya adalah kata « .\* oo .\* » berarti mencari semua kata yang terdiri dari minimal 2 karakter o.

4. Choice

Dengan menggunakan symbol kurung siku buka dan tutup “[ ]” maka pola kata yang akan cocok akan terbatas pada RE yang akan diberikan di dalam symbol tersebut. Contohnya adalah « n[a,l,o]p » dimana berarti kata yang dapat cocok adalah nap, nlp, dan nop saja.

5. Range

Tanda “-” digunakan untuk menunjukkan suatu range, misalnya kata yang dapat cocok dari RE « n[a-z]p » adalah semua kombinasi dari huruf a-z (nap, nbp, ncp, dst.). Contoh RE untuk mencari kata dimana kata tersebut dimulai dengan huruf kapital adalah: « [A-Z][a-z]\* »

6. Complementation

Tanda “^” digunakan untuk membuat makna kebalikannya (negasi). Contohnya RE « [aiueo] » adalah RE untuk menghasilkan karakter vokal, tetapi jika kita menggunakan tanda tersebut menjadi « ^[aiueo] » maka karakter yang dihasilkan adalah karakter lain selain a, i, u, e, o, atau dengan kata lain yaitu karakter konsonan saja.

## 7. Common special symbol

Tanda “^” dan “\$” digunakan untuk mencocokkan awalan dan akhiran dari sebuah baris di dalam file. Tanda “^” memiliki dua arti. Bila tanda tersebut digunakan menjadi sebuah awalan pada class character seperti pada nomor 6 maka tanda tersebut berarti negasi, selain itu tanda tersebut berarti menandakan awal dari sebuah baris. Contohnya jika RE « ^ [A-Za-z]+ » berarti mencari di setiap awal baris kata yang terdiri dari satu atau lebih karakter A-Z (baik huruf besar ataupun huruf kecil).

## 8. Other Special character

Special Sequences	
\b	Word boundary (zero width)
\d	Any decimal digit (equivalent to [0-9])
\D	Any non-digit character (equivalent to [^0-9])
\s	Any whitespace character (equivalent to [ \t\n\r\f\v])
\S	Any non-whitespace character (equivalent to [^ \t\n\r\f\v])
\w	Any alphanumeric character (equivalent to [a-zA-Z0-9_])
\W	Any non-alphanumeric character (equivalent to [^a-zA-Z0-9_])

## 2.2 Gambaran umum aplikasi web

Pada tugas besar ini, kami menggunakan bantuan React untuk membuat *frontend*. Aplikasi nantinya akan memiliki 3 fitur yaitu menerima input penyakit baru ke basis data, memprediksi apakah seseorang memiliki penyakit tertentu, serta mencari hasil prediksi yang telah dilakukan.

## BAB III

### ANALISIS PEMECAHAN MASALAH

#### 3.1 Langkah penyelesaian masalah

##### A. Input Penyakit dan Sequence DNA Penyakit

Fitur ini merupakan awal dari aplikasi dimana pengguna akan memasukkan setiap penyakit yang akan diprediksi. Input yang diterima adalah nama dari penyakit dan file dari sequence DNA penyakit. Masukkan sequence DNA akan diperiksa dengan menggunakan regex. Jika input sudah sesuai, maka masukkan akan dimasukkan kedalam basis data.

##### B. Memprediksi Penyakit

Fitur untuk memprediksi apakah seseorang menderita penyakit tertentu. Langkah penyelesaiannya adalah:

- Pengguna memasukkan nama pengguna, sequence DNA pengguna yang disanitasi dengan menggunakan regex, serta nama penyakit yang akan diprediksi.
- Sequence DNA penyakit yang akan diprediksi diambil dari basis data kemudian dicocokkan dengan DNA pengguna dengan menggunakan algoritma BM maupun KMP(sesuai dengan pilihan pengguna)
- Hasil kemudian ditampilkan ke layar, jika sequence sesuai maka akan dihasilkan True, jika tidak maka akan dihasilkan False
- Hasil kemudian dimasukkan ke dalam basis data.

##### C. Searching Hasil Prediksi

Fitur untuk melakukan pencarian terhadap hasil prediksi yang telah dilakukan oleh pengguna. Terbagi menjadi 3 yaitu:

###### 1. Pencarian berdasarkan tanggal percobaan dan nama penyakit

Jika user memasukkan input dengan format YYYY-MM-DD<spasi>Nama Penyakit, maka pencarian akan dilakukan berdasarkan tanggal dan nama penyakit sehingga aplikasi akan menampilkan setiap percobaan sesuai dengan tanggal dan nama penyakit yang diinput.

###### 2. Pencarian berdasarkan tanggal percobaan saja

Jika user memasukkan input dengan format YYYY-MM-DD, maka pencarian akan dilakukan berdasarkan tanggal saja sehingga hasilnya adalah semua percobaan yang dilakukan pada tanggal tersebut terhadap penyakit apapun.

###### 3. Pencarian berdasarkan nama penyakit saja

Jika user memasukkan input nama saja, maka pencarian akan dilakukan berdasarkan nama saja sehingga hasilnya adalah semua percobaan sesuai dengan nama penyakit yang dimasukkan dengan tanggal percobaan kapanpun.

### 3.2 Fitur fungsional dan arsitektur aplikasi web

Aplikasi ini terdiri dari sebuah menu utama dan tombol untuk membuka menu lain.

#### A. Menu Utama

**Main Menu**

Tambah Penyakit

Tes DNA

Hasil Tes

Terdapat 3 menu yang bisa dibuka dari menu utama, yaitu menu tambah penyakit, menu test dna, dan menu hasil test. Menu-menu tersebut dapat dibuka dengan menekan tombol yang sesuai.

#### B. Tambah Penyakit

Back

**Tambah Penyakit**

Nama Penyakit :

Sequence DNA :  

Choose File No file chosen

Submit



Menu Tambah Penyakit memiliki beberapa fitur. Terdapat tombol back yang akan mengembalikan menu utama. Terdapat sebuah kolom “Nama Penyakit :” untuk memasukan nama penyakit. File sequence penyakit juga dapat dipilih pada kolom “Sequence DNA :” dengan menekan tombol Choose File. Setelah nama penyakit dan file sequence telah dimasukan terdapat tombol submit yang dapat ditekan untuk menambahkan penyakit baru pada database.

### C. Tes DNA

**Tes DNA**

Nama Pengguna :

Sequence DNA :  
 No file chosen

Prediksi Penyakit :

Menu Tes DNA memiliki beberapa fitur. Terdapat tombol back yang akan mengembalikan menu utama. Terdapat sebuah kolom “Nama Pengguna :” untuk memasukan nama pasien/pengguna. File sequence pasien/pengguna juga dapat dipilih pada kolom “Sequence DNA :” dengan menekan tombol Choose File. Nama penyakit dapat dimasukkan pada kolom “Prediksi Penyakit : ”. Setelah nama pasien/pengguna, file sequence, dan nama penyakit telah dimasukan terdapat tombol submit yang dapat ditekan untuk menambahkan hasil tes baru pada database.

### D. Hasil Tes

[Back](#)

## Hasil Tes

[Search](#)

Menu hasil test memiliki kolom untuk memasukan query dan tombol search untuk menampilkan hasil query.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Spesifikasi teknis program

1. BM.go

Dalam file ini terdapat dua method, yaitu method buildLast dan bmMatch. Method buildLast berfungsi untuk mengembalikan array indeks penyimpanan dari kemunculan terakhir setiap karakter ASCII dalam pola. Method bmMatch berfungsi untuk mengecek kesamaan antara text dengan pattern.

2. KMP.go

File ini memiliki fungsi kmpMatch dan borderFunction. Fungsi borderFunction adalah fungsi pinggiran untuk algoritma string matching KMP yang mengembalikan sebuah array berisi ukuran terbesar prefiks yang juga merupakan sufiks. Fungsi kmpMatch berfungsi untuk mengecek kesamaan antara text dengan pattern.

3. main.go

File ini berisi fungsi main cadangan untuk menguji setiap fitur yang ada berbasis CLI.

4. mains.go

File ini berfungsi untuk menghubungkan backend golang dengan frontend reactJS.

5. operation.go

File ini berisi fungsi yang diperlukan untuk setiap fitur yang ada di aplikasi.

- InsertPenyakit berfungsi untuk memasukkan penyakit baru ke dalam basis data
- checkString berfungsi untuk mengecek input sebuah string apakah sesuai dengan regex DNA
- checkSearchInput berfungsi untuk merealisasikan fitur 3 yaitu Search. Fungsi ini akan mengecek input dari user dan mencocokkan query pencarian dengan regex(tanggal saja, nama saja, atau tanggal dan nama)
- readFile berfungsi untuk membaca isi sebuah file dan mengembalikannya dalam bentuk string
- checkSequence menerima nama file dari sequence DNA dan mengembalikan true jika sequence sesuai dengan regex DNA
- InsertHasilTes berfungsi untuk memasukkan hasil tes ke dalam basis data

#### 4.2 Tata cara penggunaan program

Program ini terdiri dari 3 menu utama, yaitu menu Tambahkan Penyakit, menu Tes DNA, dan menu Pencarian Hasil Prediksi. Pada menu Tambahkan Penyakit, program ini menerima input nama penyakit dan sequence DNA-nya. Pada menu Tes DNA, program ini menerima input nama pengguna, sequence DNA-nya, dan prediksi penyakitnya serta mengeluarkan output

berupa hasil tesnya. Pada menu Pencarian Hasil Prediksi, program ini menerima input query pencarian serta mengeluarkan output berupa data-data yang sesuai dengan query tersebut.

### 4.3 Hasil pengujian beserta analisisnya

- Tambah penyakit baru
  - ❖ Berhasil

Back

## Tambah Penyakit

Nama Penyakit :

Sequence DNA :

Choose File Kolera.txt

Submit

```
MariaDB [dna]> select * from penyakit
-> ;
+-----+-----+-----+
| id | nama_penyakit | sequence |
+-----+-----+-----+
| 1 | Kolera        | CGCATGTGGA |
+-----+-----+-----+
1 row in set (0.000 sec)
```

- ❖ Gagal (Sequence tidak memenuhi regex)

[Back](#)

## Tambah Penyakit

**Nama Penyakit :**

**Sequence DNA :**

[Choose File](#) Cacar\_Salah.txt

[Submit](#)

```
MariaDB [dna]> select * from penyakit;
+----+-----+-----+
| id | nama_penyakit | sequence |
+----+-----+-----+
| 1  | Kolera        | CGCATGTGGA |
+----+-----+-----+
1 row in set (0.000 sec)
```

- Memprediksi penyakit seseorang

[Back](#)

## Tes DNA

**Nama Pengguna :**

**Sequence DNA :**

[Choose File](#) No file chosen

**Prediksi Penyakit :**

[Submit](#)

- Mencari hasil prediksi

# Hasil Tes

# BAB V

## PENUTUP

### 5.1 Kesimpulan

Melalui pengerjaan Tugas Besar 3 Strategi Algoritma, kami telah berhasil membuat sebuah aplikasi berbasis *website* dalam bahasa Golang sebagai backend, React sebagai *frontend*, serta MySQL sebagai basis data yang mengimplementasikan algoritma *string matching* (KMP dan Boyer-Moore) dan mengaplikasikan penggunaan regular expression untuk sanitasi input. Aplikasi ini terdiri dari 3 fitur yaitu menerima input penyakit, melakukan prediksi apakah seseorang menderita penyakit tertentu, dan mencari hasil prediksi yang telah dilakukan.

### 5.2 Saran

Tentunya program yang kami buat masih dapat dikembangkan menjadi lebih baik seperti pembuatan tampilan yang lebih menarik serta menyertai setiap fungsi dengan komentar yang sesuai agar lebih mudah dimengerti oleh orang awam.

### 5.3 Refleksi

Tugas Besar 3 ini cukup menantang karena memanfaatkan bahasa pemrograman yang belum pernah kami gunakan yaitu Golang serta penggunaan React. Melalui tugas besar ini kami belajar untuk menggunakan dan menyambungkan hal-hal baru dalam waktu yang singkat dan padat karena ditemani oleh tugas besar mata kuliah lain sehingga hasil yang diperoleh masih belum sempurna.

## DAFTAR PUSTAKA

1. <https://github.com/ritaly/README-cheatsheet>
2. <https://dasarpemrogramangolang.novalagung.com/>
3. <https://github.com/labstack/echo>
4. <https://docs.gofiber.io/>
5. <https://go.dev/doc/tutorial/web-service-gin>
6. <https://github.com/go-sql-driver/mysql>
7. <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>
8. <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf>
9. [http://p2k.unkris.ac.id/id3/1-3065-2962/Algoritma-Knuth-Morris-Pratt\\_27391\\_itkj\\_p2k-unkris.html](http://p2k.unkris.ac.id/id3/1-3065-2962/Algoritma-Knuth-Morris-Pratt_27391_itkj_p2k-unkris.html)
10. [http://p2k.unkris.ac.id/id1/1-3065-2962/Algoritma-Boyer-Moore\\_27387\\_p2k-unkris.html](http://p2k.unkris.ac.id/id1/1-3065-2962/Algoritma-Boyer-Moore_27387_p2k-unkris.html)
11. <https://socs.binus.ac.id/2018/11/26/regular-expression/>

Link repository → [https://github.com/johannes-ws/Tubes3\\_13520063](https://github.com/johannes-ws/Tubes3_13520063)