

LAPORAN TUGAS BESAR 3

Penerapan String Matching dan Regular Expression dalam DNA Pattern Matching

Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2211 Strategi Algoritma
pada Semester II Tahun Akademik 2021/2022

Disusun oleh:

Ahmad Alfani Handoyo (K2)	13520023
Saul Sayers (K1)	13520094
Rizky Ramadhana Putra Kusnaryanto (K1)	13520151



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

2022

DAFTAR ISI

DAFTAR ISI.....	1
BAB I DESKRIPSI TUGAS	2
BAB II LANDASAN TEORI	5
1) Algoritma Knuth-Morris-Pratt (KMP).....	5
2) Algoritma Boyer-Moore (BM).....	6
3) Regular Expression (Regex)	7
4) Longest Common Subsequence (LCS)	9
5) Gestalt Pattern Matching.....	9
6) Aplikasi Web yang Dibangun	10
BAB III ANALISIS PEMECAHAN MASALAH	11
1. Langkah Penyelesaian Masalah Tiap Fitur	11
2. Fitur Fungsional dan Arsitektur Aplikasi Web	14
BAB IV IMPLEMENTASI DAN PENGUJIAN	16
1. Spesifikasi Teknis Program	16
2. Penjelasan Tata Cara Penggunaan Program.....	18
3. Hasil Pengujian	19
4. Analisis Hasil Pengujian	23
BAB V KESIMPULAN, SARAN, DAN REFLEKSI.....	24
1. Kesimpulan	24
2. Saran	24
3. Refleksi	25
LINK PENTING	26
DAFTAR PUSTAKA.....	26

BAB I

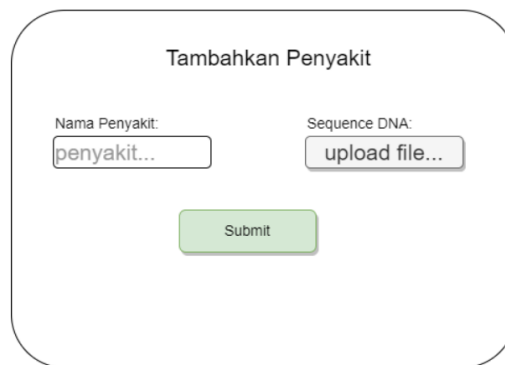
DESKRIPSI TUGAS

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.

Fitur-Fitur Aplikasi:

1. Aplikasi dapat menerima *input* penyakit baru berupa nama penyakit dan sequence DNA-nya (dan dimasukkan ke dalam *database*).

- Implementasi *input sequence* DNA dalam bentuk file.
- 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).
- Contoh *input* penyakit:



Gambar 1.1 Contoh 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:

Gambar 1.2 Ilustrasi Prediksi

3. Aplikasi memiliki halaman yang menampilkan urutan hasil prediksi dengan kolom pencarian di dalamnya. Kolom pencarian bekerja sebagai *filter* dalam menampilkan hasil.

a. 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.

b. Kolom pencarian dapat menerima masukan hanya tanggal ataupun hanya nama penyakit. Fitur ini diimplementasikan menggunakan **regex**.

c. Contoh ilustrasi:

- Masukan tanggal dan nama penyakit
- Masukan hanya tanggal
- Masukan hanya nama penyakit

Search Input	Result 1	Result 2	Result 3	Result 4	Result 5	Result 6
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 - Yuba - HIV - True	6. 13 April 2022 - Hika - HIV - False
13 April 2022	1. 13 April 2022 - Fulan - Diabetes - True	2. 13 April 2022 - Kamal - Simulasi - False	3. 13 April 2022 - Dinda - Covid Syndrome - False	4. 13 April 2022 - Jamal - False - True	5. 13 April 2022 - Yuba - TBC - True	6. 13 April 2022 - Hika - Hepatitis A - False
HIV	1. 13 April 2022 - Fulan - HIV - True	2. 14 April 2022 - Kamal - HIV - False	3. 15 April 2022 - Entah - HIV - False	4. 16 April 2022 - Jamal - HIV - True	5. 17 April 2022 - Yuba - HIV - True	6. 18 April 2022 - Hika - HIV - False

Gambar 1.3 Contoh Interaksi Program

4. (Bonus) Menghitung tingkat kemiripan DNA pengguna dengan DNA penyakit pada tes DNA

a. Ketika melakukan tes DNA, terdapat persentase kemiripan DNA dalam hasil tes.

Contoh hasil tes: 1 April 2022 - Mhs IF - HIV - 75% - False

b. Perhitungan tingkat kemiripan dapat dilakukan dengan menggunakan Hamming distance, Levenshtein distance, LCS, atau algoritma lainnya (dapat dijelaskan dalam laporan).

c. 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.

d. Contoh tampilan:

Tes DNA

Nama Pengguna:

Sequence DNA:

Prediksi Penyakit:

Hasil Tes

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

Gambar 1.4 Ilustrasi Kemiripan

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:
 - Nama penyakit
 - Rantai DNA penyusun.
 - b. Hasil Prediksi:
 - Tanggal prediksi
 - Nama pasien
 - Penyakit prediksi
 - Status terprediksi.
6. Jika mengerjakan bonus tingkat kemiripan DNA, simpan hasil tingkat kemiripan tersebut pada basis data.

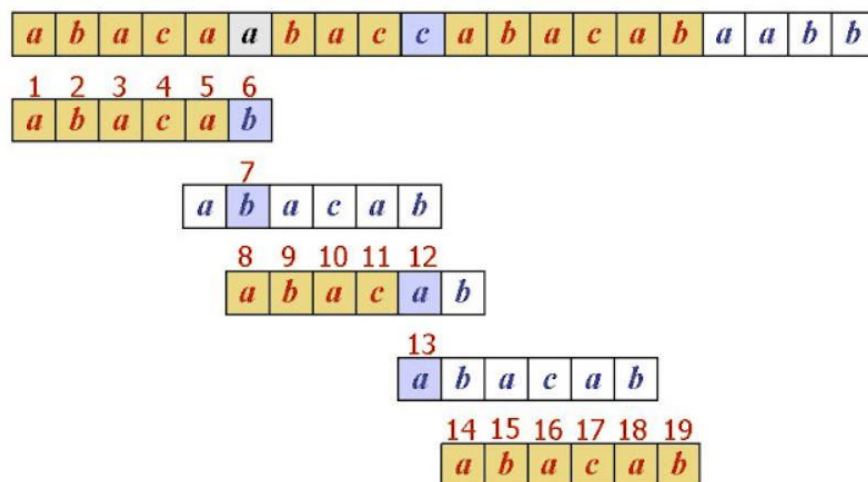
BAB II

LANDASAN TEORI

1) Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang ditemukan oleh D.E. Knuth bersama J.H. Morris dan V.R. Pratt. Algoritma ini merupakan pengembangan dari algoritma pencarian string dengan pendekatan secara *Brute Force*. Algoritma KMP akan melakukan pencarian pola dalam sebuah string secara urut dari kiri ke kanan seperti algoritma *Brute Force*, namun memiliki pola pencarian yang lebih cerdas sehingga meningkatkan efisiensi dari pencarian dengan mengurangi banyaknya perbandingan huruf yang diperiksa.

Algoritma ini memanfaatkan kecocokan prefix dan suffix dari sebuah string. Apabila saat pencarian ditemukan ketidak-cocokan sebuah huruf antara string S pada indeks ke i dengan pola P pada indeks ke j, maka indeks i pada S dapat digeser sebanyak prefix terbesar dari $P[0..j-1]$ yang merupakan suffix dari $P[1..j-1]$ untuk mengurangi pemborosan pengecekan dengan menghindari pemeriksaan huruf yang tidak perlu. Berikut adalah ilustrasi pergeserannya :



Gambar 2.1.1 Ilustrasi Pergeseran Prefix dengan KMP

Untuk mempermudah mengetahui prefix terbesar yang cocok dengan suffix apabila ditemukan ketidak-cocokan huruf pada suatu indeks x , algoritma KMP menggunakan suatu *border function* atau dengan nama lain *failure function* yang menyimpan nilai dari panjang prefix terbesar yang sesuai. Hal tersebut dimaksudkan agar KMP dapat melakukan pergeseran berdasarkan informasi yang disimpan sehingga membuat waktu pencarian panjang prefix terbesar menurun drastis. Berikut adalah contoh *failure function* untuk ilustrasi sebelumnya :

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

Gambar 2.1.2 Contoh *failure function* untuk Ilustrasi Sebelumnya

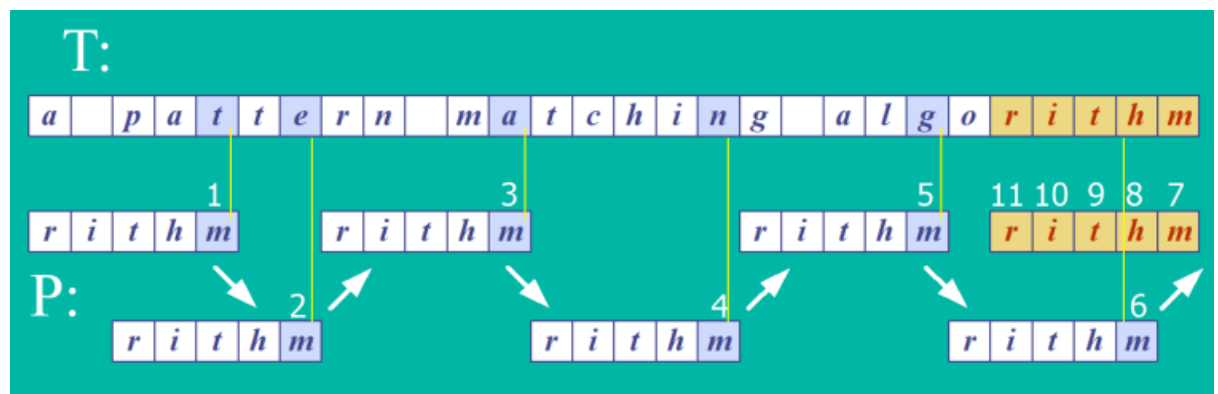
2) Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan string yang ditemukan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Berbeda dengan algoritma KMP dan *Brute Force* yang melakukan pencocokan pola urut dari kiri ke kanan, algoritma Boyer-Moore justru melakukan pencocokan pola dari kanan ke kiri. Ide dibalik algoritma ini adalah untuk memulai pencocokan dari kanan sehingga lebih banyak informasi yang didapatkan. Pemeriksaan terhadap suatu string S dimulai dari awal, namun terhadap suatu pola P dimulai dari indeks terakhirnya yakni panjang pola $- 1$.

Algoritma Boyer-Moore memanfaatkan dua teknik, yakni *the looking-glass technique* dan *character-jump technique*. *The looking-glass technique* merupakan bagaimana kita memeriksa suatu pola P dalam string S dengan bergerak mundur terhadap P , dimulai dari akhir. Apabila huruf yang diperiksa benar, maka akan lanjut bergeser ke kiri hingga semua huruf sudah diperiksa benar atau ditemukan huruf yang tidak cocok. Apabila huruf yang diperiksa salah, akan memanfaatkan *character-jump technique*. *Character-jump technique* merupakan pergeseran indeks i pada S apabila ditemukan huruf yang tidak cocok, dan dikembalikan ke indeks terakhir pada P . Terdapat 3 kasus dalam pergeseran indeks i untuk ketidakcocokan huruf x di S , yaitu :

1. P mengandung x , maka kita menggeser P agar kemunculan huruf x terakhir pada P sejajar dengan $S[i]$. Dengan kata lain, kita mengembalikan j menjadi panjang $P - 1$ dan menambah i hingga sejajar dengan kemunculan huruf x terakhir pada P .
2. P mengandung x tetapi tidak memungkinkan untuk menggeser P hingga kemunculan terakhir x dalam P sejajar dengan x dalam S . Dengan demikian, kita menggeser P sebanyak 1 karakter di kanan $S[i+1]$.
3. Kasus selain kasus 1 dan kasus 2. Apabila kasus ini terjadi, maka kita geser P agar $P[0]$ sejajar dengan $T[i+1]$.

Berikut adalah ilustrasi pergeseran dengan algoritma Boyer-Moore :



Gambar 2.2.1 Ilustrasi Pergeseran dengan BM

Untuk mempermudah mengetahui kemunculan terakhir x dalam suatu pola P , algoritma BM memanfaatkan sebuah *Last Occurrence Function* yang menyimpan indeks terakhir kemunculan huruf x dalam P . Hal tersebut dimaksudkan agar BM dapat melakukan pergeseran berdasarkan informasi yang disimpan sehingga membuat waktu pencarian index kemunculan terakhir menurun drastis. Berikut adalah contoh *Last Occurrence Function* untuk ilustrasi sebelumnya :

P	a	b	a	c	a	b
	0	1	2	3	4	5

<i>x</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>L(x)</i>	4	5	3	-1

Gambar 2.2.1 Contoh *Last Occurrence Function*

3) Regular Expression (Regex)

Regular Expression (Regex) merupakan sebuah konsep yang ditemukan oleh seorang ilmuwan matematika bernama Stephen Cole Kleene saat sedang memformulasikan definisi terkait bahasa formal pada tahun 1951. Regex merupakan sebuah teks atau string yang mendefinisikan pola pencarian agar dapat melakukan pencocokan string (*matching*), pencarian string (*locate*), dan manipulasi string. Konsep ini kemudian diterapkan kepada beberapa bahasa pemrograman menjadi suatu library pemroses string.

String matching dengan Regex berbeda dengan algoritma *Brute Force*, KMP, dan BM yang melakukan *exact matching*. Regex melakukan string matching berdasarkan notasi dan pola yang dimilikinya. Perlu diperhatikan bahwa pola dan notasi dalam Regex termasuk case-sensitive. Simbol dalam pola Regex dapat diklasifikasikan menjadi hal – hal berikut :

1. Wildcard : Simbol titik “.” yang melambangkan cocok dengan huruf apapun selain newline
2. Optionality : simbol tanda tanya “?” yang menandakan bahwa simbol yang diberikan sebelum simbol tanda tanya merupakan optional
3. Repeatability : Simbol tanda plus “+” yang melambangkan bahwa Regex dapat diulang lebih dari sekali namun minimal diulang sekali, dan tanda asterisk “*” yang melambangkan boleh diulang namun boleh tidak disertakan sama sekali
4. Choice : Simbol kurung siku buka dan tutup “[]” yang melambangkan pola kata yang cocok terbatas berdasarkan Regex yang diberikan dalam choice tersebut.
5. Range : simbol kurang “-” yang menunjukkan range, misalkan A-Z yang berarti dari huruf A kapital hingga Z kapital dan a-z yang berarti dari huruf a hingga z
6. Complementation : Simbol panah keatas “^” yang berarti kebalikan atau negasinya, sehingga simbol yang berada setelah simbol ^ tidak boleh dipakai
7. Special Symbol : Simbol panah keatas “^” juga dapat digunakan untuk mencocokkan awalan dari sebuah baris apabila diletakkan di awal regex, dan simbol dollar “\$” dapat digunakan untuk mencocokkan akhiran suatu baris.
8. Alternation : Simbol garis tegak “|” yang menjadi semacam if-else
9. Token : Beberapa simbol khusus yang memiliki makna tertentu. Berikut adalah daftar token yang ada dalam regex :

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_])

Gambar 2.3.1 Token dalam regex

Dengan demikian, simbol – simbol sebelumnya dapat digunakan dalam regex untuk mendapatkan notasi – notasi umum dengan contoh sebagai berikut :

.	Any character except newline.
\.	A period (and so on for *, \ (, \\, etc.)
^	The start of the string.
\$	The end of the string.
\d, \w, \s	A digit, word character [A-Za-z0-9_], or whitespace.
\D, \W, \S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	a through z.
[^abc]	Any character except a, b, or c.
aa bb	Either aa or bb.
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{n}	Exactly n of the preceding element.
{n, }	n or more of the preceding element.
{m, n}	Between m and n of the preceding element.
??, *?, +?, {n}?, etc.	Same as above, but as few as possible.
(expr)	Capture expr for use with \1, etc.
(?:expr)	Non-capturing group.
(?=expr)	Followed by expr.
(?!expr)	Not followed by expr.

Gambar 2.3.2 Notasi Umum dalam Regex

Notasi regex tersebut dapat dimanfaatkan dalam bahasa pemrograman untuk melakukan string matching. Mayoritas bahasa pemrograman memiliki library-nya sendiri untuk pengecekan Regex. Fungsi string matching akan menerima parameter string yang akan diperiksa serta pola yang akan diperiksa dalam string tersebut dan mengembalikan boolean True or False berdasarkan ada atau tidaknya pola ditemukan atau mengembalikan indeks ditemukannya pola tersebut, bergantung fungsi yang digunakan. Berikut adalah contoh string matching menggunakan pola dalam Regex :

<pre>/[bcr]at/g</pre> <p>Test String</p> <p>bat rat cat</p>	<pre>/^[bcr]at/g</pre> <p>Test String</p> <p>bat rat cat hat</p>	<pre>/(ade)/g</pre> <p>Test String</p> <p>aderay bade</p>	<pre>/[ade]/g</pre> <p>Test String</p> <p>aderay bade</p>
<pre>/ke-[1-3]/g</pre> <p>Test String</p> <p>Peringkat ke-1 dan ke-5</p>	<pre>/^[a-z]/g</pre> <p>Test String</p> <p>huruf besar saja</p>		

Gambar 2.3.3 Contoh String Matching Menggunakan Regex

4) Longest Common Subsequence (LCS)

Longest Common Subsequence adalah permasalahan mencari subsequence (uparangkaian) terpanjang dari dua buah string atau teks. Perlu diperhatikan bahwa subsequence berbeda dengan substring. Substring merupakan upa-rangkaian dalam sebuah string yang harus *consecutive* (harus urut) berdasarkan string semula, sementara subsequence memperbolehkan untuk tidak *consecutive* tetapi tetap urut relatif dari kemunculannya. Sebagai contoh, dua buah string ABCDEFG dengan BCEZ akan memiliki substring terpanjang BC dengan panjang 2, namun memiliki subsequence terpanjang sebagai BCE dengan panjang 3. Berikut adalah contoh LCS untuk string lain :

"gxtxayb"
 "abgtab"
 "gyaytahjb"

The LCS is shown using red characters in the 3 given strings.

Gambar 2.4 Contoh LCS untuk 3 string

5) Gestalt Pattern Matching

Gestalt Pattern Matching, atau yang disebut juga sebagai Ratcliff/Obershelp Pattern Recognition, merupakan algoritma string-matching yang dikembangkan oleh John W. Ratcliff dan John A. Obershelp pada tahun 1983 dan digunakan untuk menentukan tingkat kesamaan (*similarity*) dari dua buah string. Algoritma tersebut memanfaatkan Longest Common Subsequence yang sudah ditentukan sebelumnya. Persamaan untuk algoritma gestalt adalah sebagai berikut :

$$D = \frac{2K_m}{|S_1| + |S_2|}$$

Di mana D didefinisikan sebagai tingkat kesamaan atau *similarity* dari dua buah string dan memiliki range nilai antara 0 sampai 1. Dari definisi formalnya, K_m merupakan hasil penjumlahan dari Longest Common Substring dengan banyaknya karakter yang sesuai pada bagian tidak match pada kedua sisi Longest Common Substring. Dengan demikian, K_m bisa juga dibilang sebagai panjang dari Longest Common Subsequence dari dua buah string. $|S_1|$ dan $|S_2|$ masing – masing merupakan panjang dari string yang akan diperiksa kemiripannya. Berikut adalah contoh penggunaan algoritma Gestalt Pattern Matching untuk menentukan *similarity* dari 2 buah string :

S ₁	W	I	K	I	M	E	D	I	A
S ₂	W	I	K	I	M	A	N	I	A

$$\frac{2K_m}{|S_1| + |S_2|} = \frac{2 \cdot (| \text{"WIKIM"} | + | \text{"IA"} |)}{|S_1| + |S_2|} = \frac{2 \cdot (5 + 2)}{9 + 9} = \frac{14}{18} = 0.\bar{7}$$

Gambar 2.5 Contoh Penggunaan Algoritma Gestalt Pattern Matching

6) Aplikasi Web yang Dibangun

Aplikasi web yang dibangun dibagi menjadi bagian *front-end*, *back-end*, dan basis data. Pada sisi *back-end*, bahasa pemrograman yang digunakan adalah Go dengan framework echo. Alasan kami menggunakan Go adalah karena kemudahannya dalam menginisialisasi sebuah server. Data kami disimpan menggunakan database Postgre. Dalam sisi *back-end* tersedia juga algoritma string matching dengan Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan juga algoritma perhitungan kemiripan antara dua string menggunakan Longest Common Subsequence (LCS). Pada sisi *front-end*, kami menggunakan bahasa pemrograman typescript dengan framework *react* untuk mempermudah proses penggunaan komponen. Tampilan aplikasi web kami ditulis pada sisi *front-end* ini, yang terdiri dari 3 halaman yakni halaman pertama untuk penambahan DNA penyakit (*add disease*), halaman untuk memprediksi apakah sebuah DNA mengandung pola penyakit (*predict*), serta halaman untuk menampilkan riwayat prediksi (*history*). Selain itu, *front-end* kami juga menggunakan framework *tailwind* untuk mempermudah dalam pembuatan CSS untuk mendesain tampilan halaman.

The screenshot shows the 'DNA Test' form in the BONEK DNA Tester application. The form includes a 'Patient Name' input field, a 'DNA Sequence' input field with a 'Choose File' button and 'No file chosen' text, a 'Disease Prediction' input field, and a 'String Matching Algorithm' dropdown menu currently set to 'Knuth-Morris-Pratt'. A 'Submit' button is at the bottom. The top navigation bar shows 'Add Disease', 'DNA Test', and 'Results'.

The screenshot shows the 'Results' page in the BONEK DNA Tester application. It features a search bar with a 'Search' button. Below the search bar is a list of 6 results, each containing a date, name, disease, percentage, and a boolean result.

No.	Date	Name	Disease	Percentage	Result
1.	2022-04-28	Ahmad Alfani Handoyo	COVID-19	81%	True
2.	2022-04-28	Jova	COVID-19	100%	True
3.	2022-04-28	Jova	COVID-19	100%	True
4.	2022-04-28	Saul	kanker payudara	100%	True
5.	2022-04-28	Saul	kanker payudara	100%	True
6.	2022-04-28	Liza	autis	73%	False

BAB III

ANALISIS PEMECAHAN MASALAH

1. Langkah Penyelesaian Masalah Tiap Fitur

- **Aplikasi dapat menerima input penyakit baru berupa nama penyakit dan sequence DNA-nya (dan dimasukkan ke dalam database).** Langkah penyelesaiannya adalah:

Pada tampilan aplikasi web, pengguna dapat mengakses halaman *add disease* yang berguna untuk menambahkan penyakit baru. Terdapat 2 form yang dapat diisi oleh pengguna. Yang pertama adalah form nama penyakit yang dapat diinput manual dengan diketik. Yang kedua adalah form sequence DNA yang dapat diisi dengan mengupload sebuah *file* txt. Aplikasi kemudian akan melakukan validasi terlebih dahulu apakah input sequence DNA sudah benar, yakni memiliki format karakter yang terdiri dari singkatan basa nitrogen A,G,C,T. Validasi dilakukan dengan menggunakan Regex dan akan bernilai benar apabila pola Regex sequence DNA sama seperti `[AGCT]*`. Apabila sequence DNA nya valid, maka program akan lanjut memanggil API untuk menambahkan data baru tersebut ke dalam database dan mengirimkan pesan ke tampilan web dalam bentuk alert bahwa data berhasil ditambahkan. Apabila pola Regex tidak valid, maka akan menampilkan pesan bahwa sequence DNA yang diberikan tidak valid dan tidak ditambahkan ke database.

- **Aplikasi dapat memprediksi apakah seorang pasien menderita penyakit tertentu berdasarkan sequence DNA penyakitnya yang dibanding dengan DNA pasien menggunakan algoritma KMP atau BM.** Langkah penyelesaiannya adalah:

Pada tampilan aplikasi web, pengguna dapat mengakses halaman *predict* yang berguna untuk memeriksa apakah sequence DNA seseorang mengandung sebuah DNA penyakit atau tidak. Terdapat 3 form yang dapat diisi oleh pengguna. Yang pertama adalah form nama pasien yang dapat diinput manual dengan diketik. Yang kedua adalah form sequence DNA manusia yang dapat diisi dengan mengupload sebuah *file* txt. Aplikasi kemudian akan melakukan validasi terlebih dahulu apakah input sequence DNA sudah benar, yakni memiliki format karakter yang terdiri dari singkatan basa nitrogen A,G,C,T. Validasi dilakukan dengan menggunakan Regex dan akan bernilai benar apabila pola Regex sequence DNA sama seperti `[AGCT]*`. Yang ketiga adalah nama penyakit yang ingin diperiksa. Terdapat sebuah toggle juga untuk menentukan algoritma string matching apakah menggunakan Apabla sequence DNA nya tidak valid, maka halaman akan menampilkan bahwa DNA tidak valid. Apabila DNA valid, maka program akan lanjut memanggil API untuk memanggil database dan mencari sequence DNA dari penyakit tersebut. Apabila penyakit tidak ada dalam database, maka aplikasi akan menampilkan bahwa penyakit tidak ada. Apabila ada, program akan lanjut memanggil API lain untuk melakukan fungsi *predict* dan akan menampilkan hasil pemeriksaannya yang berupa nama pasien, tanggal pemeriksaan, nama penyakit, persentase kemiripan, serta true or false memiliki penyakitnya.

- **(BONUS) Aplikasi dapat menghitung tingkat kemiripan DNA pengguna dengan DNA penyakit pada tes DNA menggunakan algoritma LCS.** Langkah penyelesaiannya adalah:

Saat penggunaan API untuk melakukan *predict*, fungsi yang dipanggil sudah memanggil fungsi lain untuk menghitung kemiripan yang sekaligus ikut dikembalikan.

Algoritma yang digunakan untuk melakukan perhitungan ini adalah algoritma Longest – Common – Subsequence. Langkah – langkah pengerjaan untuk algoritma LCS sendiri akan dijelaskan di bawah ini juga.

- **Aplikasi memiliki halaman yang menampilkan urutan hasil prediksi dengan kolom pencarian di dalamnya, di mana kolom pencarian dapat bekerja sebagai filter. Filter dapat bekerja dengan 3 kasus, yaitu nama penyakit saja, tanggal saja, atau keduanya. Langkah penyelesaiannya adalah:**

Pada tampilan aplikasi web, pengguna dapat mengakses halaman *history* yang berguna untuk melihat riwayat pemeriksaan yang difilter berdasarkan sebuah query. Terdapat 1 form yang dapat diisi oleh pengguna yakni untuk Querynya. Terdapat 3 kasus query, yang pertama hanya tanggal, kedua hanya nama penyakit, dan ketiga adalah tanggal sekaligus nama penyakit. Perbedaan kasus tersebut dapat dilakukan menggunakan regex, dengan pola-nya masing masing secara berturut turut adalah $\backslash d\{4\}-\backslash d\{2\}-\backslash d\{2\}$, $[\backslash S]^*$, serta $\backslash d\{4\}-\backslash d\{2\}-\backslash d\{2\}\backslash s+[\backslash S]^*$. Setelah memfilter querynya, Aplikasi akan memanggil API untuk memanggil query dengan where clause yang disesuaikan berdasarkan kasusnya. Kemudian, aplikasi web akan menampilkan hasil query yang didapat yang ditampilkan dalam bubble untuk tiap barisnya. Apabila tidak ada hasil sama sekali, maka aplikasi akan menampilkan bahwa tidak ditemukan hasil apapun.

- **Langkah penyelesaian pencocokan string DNA dengan algoritma KMP adalah sebagai berikut :**

- 1) Membuat *Border Function* atau *fail function* dari string DNA penyakit P yang akan diperiksa keberadaannya di dalam string DNA manusia S. *Fail function* direpresentasikan dengan sebuah array untuk tiap indeks k
- 2) Menginisiasi sebuah array dengan panjang sebesar panjang P – 1 untuk mengisi *fail function*.
- 3) Iterasi untuk tiap indeks k pada array *fail function* diisi dengan panjang prefix terbesar yang sesuai dengan suffixnya.
- 4) Memulai pencarian dengan algoritma KMP dari huruf pertama pada S dan P. Menggunakan indeks i untuk menunjukkan huruf pada S yang sedang diperiksa dan indeks j untuk menunjukkan huruf pada P yang sedang diperiksa. Masukkan nilai i dan j awal sebagai 0.
- 5) Dari kiri ke kanan, lakukan pencocokkan huruf pada S[i] dengan P[j].
- 6) Jika kedua huruf cocok, maka tambah indeks i dan j dengan 1.
- 7) Jika kedua huruf tidak cocok dan j tidak bernilai 0, maka ubah nilai j menjadi value pada *fail function* di indeks j – 1.
- 8) Jika kedua huruf tidak cocok dan j bernilai 0, maka tambah i dengan 1.
- 9) Ulangi dari langkah 5 hingga semua huruf pada P sudah berhasil diperiksa (ditemukan), yakni dengan nilai j sebesar panjang P atau hingga nilai i melebihi panjang S (tidak ditemukan).

- **Langkah penyelesaian pencocokan string DNA dengan algoritma Boyer-Moore adalah sebagai berikut:**

- 1) Membuat *Last Occurrence Function* dari string DNA penyakit P yang akan diperiksa keberadaannya di dalam string DNA manusia S. *Last Occurrence Function* direpresentasikan dengan sebuah map yang memiliki key tiap huruf yang terkandung dalam P, dan valuenya index terakhir huruf tersebut ditemukan.
- 2) Menginisiasi sebuah map untuk menyimpan *Last Occurrence Function*.
- 3) Iterasi untuk tiap huruf pada P. Apabila huruf belum ada dalam map *Last Occurrence Function*, maka masukkan key tersebut dalam map dengan valuenya sebagai index huruf tersebut. Apabila

huruf tersebut sudah ditemukan, maka update value dari map dengan key tersebut dengan index baru huruf tersebut.

- 4) Memulai pencarian dengan algoritma BM di huruf terakhir pada P. Menggunakan indeks i untuk menunjukkan huruf pada S yang sedang diperiksa dan indeks j untuk menunjukkan huruf pada P yang sedang diperiksa. Masukkan nilai i dan j awal sebagai panjang P – 1.
- 5) Dari kanan ke kiri, lakukan pencocokkan huruf pada S[i] dengan P[j].
- 6) Jika kedua huruf cocok, maka menerapkan *The Looking-Glass Technique* dengan mengurangi indeks i dan j sebesar 1.
- 7) Jika kedua huruf tidak cocok, maka menerapkan *The Character Jump Technique*. Pergeseran indeks berdasarkan kasus yang didapat.
- 8) Apabila kasus 1 yakni huruf yang salah pada S terdapat pada *Last Occurrence Function* dan dapat digeser hingga align, maka kita menggeser S agar align dengan kemunculan terakhir huruf tersebut dalam P. Tambahkan i sebesar panjang P-1 dikurangi dengan indeks kemunculan terakhir huruf tersebut yang diperoleh dari *Last Occurrence Function*.
- 9) Apabila kasus 2 yakni huruf yang salah pada S terdapat pada *Last Occurrence Function* namun tidak bisa digeser, maka kita cukup geser S sebesar 1 huruf ke kanan terhitung dari awal pemeriksaan. Tambahkan i sebesar panjang P dikurangi dengan indeks j.
- 10) Apabila kasus 3 yakni huruf yang salah pada S tidak terdapat pada *Last Occurrence Function*, maka kita menggeser S sejauh panjang P. Tambahkan i sebesar panjang P.
- 11) Reset kembali indeks j menjadi panjang P – 1
- 12) Ulangi dari langkah 5 hingga semua huruf pada P sudah berhasil diperiksa (ditemukan), yakni dengan nilai j sebesar -1 atau hingga nilai i melebihi panjang S (tidak ditemukan).

• **Langkah Penyelesaian Perhitungan Kemiripan DNA penyakit dengan Algoritma LCS adalah sebagai berikut:**

- 1) Melakukan iterasi untuk tiap substring pada string S (DNA manusia) dengan panjang yang sama dengan panjang pola P (DNA penyakit).
- 2) Menghitung LCS dari substring S dan P menggunakan pendekatan *Dynamic Programming*.
- 3) Membuat membuat tabel atau matriks M yang memiliki dimensi $(n+1)(m+1)$ di mana n adalah panjang dari string S dan m adalah panjang dari pola P yang akan diperiksa keberadaannya dalam P.
- 4) Isi kolom pertama dan baris pertama dari M dengan 0
- 5) Iterasi untuk mengisi untuk tiap cell dalam matriks yang masih kosong, dimulai dari baris kedua dengan arah dari kiri ke kanan. Iterasi menggunakan indeks i yang menunjuk ke huruf pada S dan indeks j yang menunjuk ke huruf pada P.
- 6) Apabila $S[i-1] = P[j-1]$, maka isi $M[i][j]$ dengan nilai $M[i-1][j-1] + 1$
- 7) Apabila $S[i-1] \neq P[j-1]$, maka ambil nilai yang terbesar antara $M[i][j-1]$ dan $M[i-1][j]$ kemudian isi nilai tersebut ke dalam $M[i][j]$
- 8) Setelah semua cell dalam M sudah terisi, maka nilai dari LCS dapat diperoleh pada cell $M[\text{panjangP}][\text{panjangS}]$.
- 9) Menghitung kemiripan antara substring S tersebut dengan P, memanfaatkan formula Gestalt dan LCS antara substring S dan P.
- 10) Ambil substring dengan kemiripan tertinggi. Maka nilai kemiripan dari DNA penyakit yang terkandung dalam DNA manusia adalah nilai kemiripan substring tersebut.

2. Fitur Fungsional dan Arsitektur Aplikasi Web

Fitur Fungsional :

- Aplikasi dapat menerima input penyakit baru berupa nama penyakit dan sequence DNA-nya (dan dimasukkan ke dalam database).
- Aplikasi dapat memprediksi apakah seorang pasien menderita penyakit tertentu berdasarkan sequence DNA penyakitnya yang dibanding dengan DNA pasien menggunakan algoritma KMP atau BM.
- **(BONUS)** Aplikasi dapat menghitung tingkat kemiripan DNA pengguna dengan DNA penyakit pada tes DNA menggunakan algoritma LCS.
- Aplikasi memiliki halaman yang menampilkan urutan hasil prediksi dengan kolom pencarian di dalamnya, di mana kolom pencarian dapat bekerja sebagai filter. Filter dapat bekerja dengan 3 kasus, yaitu nama penyakit saja, tanggal saja, atau keduanya.

Arsitektur Aplikasi Web :

Front-end React dijalankan dari *App.tsx* yang di dalamnya me-render suatu *Navigation Bar* yang mengarah ke ketiga halaman penambahan DNA penyakit, prediksi penyakit pasien, dan riwayat prediksi, serta sebuah switch router halaman. Switch ini berfungsi untuk menampilkan salah satu dari tiga halaman yang dipilih sesuai dengan *link* halaman yang sedang dikunjungi. Router ini digunakan supaya tidak perlu adanya repetisi penampilan halaman dengan *Navigation Bar* yang sama. Router ini diatur pada *routes.tsx* yang menampilkan ketiga halaman pada file-file *.tsx* yang terdapat pada folder *pages* yang kemudian dipanggil menggunakan Router dan Switch yang disediakan oleh library *wouter*.

Halaman *AddDisease.tsx* berisi tampilan untuk halaman penambahan DNA penyakit. Terdapat form HTML yang meminta input nama penyakit dan juga *sequence* DNA dari penyakit berupa file *.txt*. File ini kemudian dikonversi terlebih dahulu menjadi sebuah *string* menggunakan *FileReader*. Kombinasi nama penyakit dan *sequence* DNA penyakit dicek (termasuk pengecekan regex AGCT pada *sequence* DNA) dan ditambahkan ke basis data penyakit. Nama penyakit dicek terlebih dahulu dengan mengambil nama-nama penyakit yang ada pada basis data menggunakan *axios* yang memanggil method GET pada halaman */get* yang akan mengembalikan *string* JSON berisi nama-nama penyakit. Bila belum ada maka nama penyakit dan DNA penyakit ditambahkan ke basis data dengan memanggil *axios* bermethod POST pada halaman */add* dengan *string* JSON (name, dna).

Halaman *DNATest.tsx* berisi tampilan untuk halaman prediksi penyakit suatu pasien. Terdapat form HTML yang meminta input nama pasien, *sequence* DNA pasien berupa file *.txt*, prediksi penyakit, dan algoritma *string-matching* yang digunakan. Seperti halnya pada halaman *AddDisease.txt*, file DNA *sequence* dikonversi terlebih dahulu menjadi sebuah *string* menggunakan *FileReader*. Nama penyakit dicek apakah sudah ada pada basis data dengan method GET pada */get*. Prediksi kemudian dilakukan pada *back-end* dengan memanggil method POST pada */predict* dengan *string* JSON (name, dna, diseasename, mode). Hasil dari prediksi ditampilkan pada *alert box*.

Halaman *Results.tsx* berisi tampilan untuk halaman *query* riwayat prediksi. Form HTML menerima suatu input *text* yang bila di-submit akan dicek regex-nya apakah berupa kombinasi *query* tanggal dan nama penyakit, tanggal saja, atau nama penyakit saja. Tergantung pada jenis *query*-nya dari tiga pilihan sebelumnya, dipanggil method POST dengan *string* JSON (date, diseasename) yang kemudian mengembalikan semua data yang memenuhi kondisi kombinasi tanggal dan/atau

nama penyakit yang dimasukkan pengguna. Hasil *query* disimpan pada array yang ditampilkan pada komponen *QueryResult*.

Setelah menerima masukan dari pengguna melalui antarmuka web, segala bentuk pemrosesan dan interaksi dengan database dilakukan pada bagian *backend*. Terdapat empat API yang bisa diakses untuk mendapat ataupun mengubah data yang diinginkan. Pertama API */predict* yang menerima nama pasien, sekuens DNA pasien, nama penyakit yang ingin diuji, dan algoritma yang ingin digunakan. API tersebut akan mengembalikan hasil uji penyakit ke *frontend* untuk kemudian ditampilkan ke pengguna. Kedua API */history* yang menerima tanggal dan/atau nama penyakit sesuai query yang diberikan. API tersebut akan mengembalikan histori dari pengecekan yang pernah dilakukan beserta hasilnya juga. Ketiga API */get* yang akan mengembalikan semua nama penyakit yang ada di basis data. Yang terakhir API */add* yang menerima nama penyakit dan sekuens DNA nya lalu akan menambahkan data tersebut ke basis data. Bagian *backend* ini berjalan pada server yang terpisah dari bagian *frontend*.

Data-data yang diperlukan disimpan dalam DBMS PostgreSQL yang dijalankan pada layanan Heroku. Terdapat dua tabel pada basis data yang digunakan yaitu tabel *disease* dan tabel *history*. Tabel *disease* menyimpan nama penyakit dan sekuens DNA-nya. Sedangkan tabel *history* menyimpan hasil-hasil pengujian terdahulu. API */predict* dan */history* akan berinteraksi dengan tabel *history*. Untuk dua API lainnya, */get* dan */add*, akan berinteraksi dengan tabel *disease*.

BAB IV IMPLEMENTASI DAN PENGUJIAN

1. Spesifikasi Teknis Program

Algoritma utama kami terletak pada *file* `algorithm.go` yang tertulis dalam bahasa pemrograman Golang dalam folder `src/backend`. Berikut adalah spesifikasi dari *file* tersebut :

Struktur Data :

- Array of runes. Struktur data ini digunakan untuk mendapatkan panjang dari sebuah string secara akurat untukantisipasi kasus di mana string mengandung char yang berupa unicode sehingga byte nya akan bertambah dan kalkulasi panjangnya menjadi salah.
- Slice of integer. Struktur data ini digunakan untuk menyimpan *border function* atau *fail function* dari algoritma KMP. Struktur data ini digunakan sebagai pengganti dari array of integer sebab dalam bahasa pemrograman Golang tidak memungkinkan untuk menginisiasi array secara dinamis ataupun dengan panjang yang dari variabel.
- Map of string to int. Struktur data ini digunakan untuk menyimpan *last occurrence function* dari algoritma Boyer-Moore. Struktur data ini menyimpan mapping dari sebuah key char (huruf dalam pola yang akan diperiksa) dan valuenya adalah index dari kemunculan terakhir huruf tersebut dalam pola.
- Matrix of int. Struktur data ini digunakan untuk melakukan kalkulasi algoritma LCS menggunakan pendekatan *dynamic programming*.

Fungsi / Prosedur :

- `function kmpBorderFunc(pola String) → [] int.`
Fungsi ini berguna untuk mendapatkan *border function* atau *fail function* dari sebuah string. Fungsi ini menerima parameter sebuah string pola yang akan dibuat *fail function*-nya, kemudian mengembalikan sebuah slice of integers.
- `function kmp(dna string, pola string) → bool.`
Fungsi ini berguna untuk melakukan pemeriksaan pencocokkan string suatu pola di dalam suatu string dna menggunakan algoritma KMP. Fungsi ini menerima parameter sebuah string dna dan pola yang akan diperiksa apakah terkandung dalam dna, kemudian mengembalikan sebuah bool true apabila ditemukan dan false apabila tidak ditemukan
- `function boyermooreLastIndex(dna string, pola string) → map[string]int`
Fungsi ini berguna untuk mendapatkan map *Last Occurences* huruf dari sebuah string. Fungsi ini menerima parameter string pola untuk mengetahui indeks kemunculan terakhir dari tiap hurufnya dan disimpan ke dalam map serta menerima parameter string dna untuk mengetahui huruf yang ada dalam dna tetapi tidak ada dalam pola sehingga bisa diset kemunculan terakhirnya menjadi -1. Fungsi ini mengembalikan map of string to int tersebut.
- `function boyermoore(dna string, pola string) → bool.`
Fungsi ini berguna untuk melakukan pemeriksaan pencocokkan string suatu pola di dalam suatu string dna menggunakan algoritma Boyer-Moore. Fungsi ini menerima parameter sebuah string dna dan pola yang akan diperiksa apakah terkandung dalam dna, kemudian mengembalikan sebuah bool true apabila ditemukan dan false apabila tidak ditemukan
- `function lcs(dna string, pola string) → int`

Fungsi ini berguna untuk mendapatkan panjang dari *Longest-Common-Sequence* antara dua string. Fungsi ini menggunakan pendekatan secara *dynamic programming* untuk menghitung panjang LCS nya sehingga menginisiasi sebuah matriks di dalamnya. Fungsi ini menerima parameter berupa string dna dan string pola yang akan dikalkulasikan panjang dari LCSnya, kemudian mengembalikan integer panjang dari LCS tersebut.

- function GestaltPatternSimilarity(lcs int, dna string, pola string) → int
Fungsi ini berguna untuk mendapatkan persentase kemiripan antara dua buah string yang sudah diketahui panjang dari *Longest-Common-Sequence*-nya menggunakan formula Gestalt. Fungsi ini menerima parameter berupa integer lcs yakni panjang dari lcs kedua string, serta string dna dan string pola yang akan diperiksa persentase kemiripannya. Kemudian, fungsi mengembalikan integer yakni persentase kemiripan kedua string.
- function lcsSimilarityPercentage(dna string, pola string) → int.
Fungsi ini berguna untuk menggabungkan fungsi lcs dan fungsi GestaltPatternSimilarity sehingga dapat secara langsung digunakan untuk mendapatkan persentasenya. Fungsi ini menerima parameter berupa string dna dan string pola yang akan dikalkulasikan persentase kemiripannya. Kemudian, fungsi mengembalikan integer yakni persentase kemiripan kedua string.
- function lcsHighestSimilarity(dna string, pola string) → int
Fungsi ini berguna untuk mendapatkan persentase kemiripan tertinggi antara dua string yang diperoleh dari mencocokkan tiap substringsnya dengan panjang yang sama. Fungsi ini memanfaatkan algoritma lcsSimilarityPercentage untuk menghitung kemiripan tiap substringsnya dengan pendekatan *brute force*. Fungsi ini menerima parameter berupa string dna yang akan diiterasikan untuk tiap substringsnya dan string pola yang akan dikalkulasikan persentase kemiripannya. Kemudian, fungsi mengembalikan integer yakni persentase kemiripan kedua string.

Backend kami terletak pada *file server.go* yang tertulis dalam bahasa pemrograman Golang dalam folder *src/backend*. Berikut adalah spesifikasi dari *file* tersebut :

Struktur Data :

- struct predictJSON, memiliki atribut Name, Dna, Diseasename, Mode. Struktur data ini digunakan untuk menampung *body request* dari API /predict
- struct diseaseJSON, memiliki atribut Names yang merupakan array of string. Struktur data ini digunakan untuk menampung hasil dari API /get yang akan dikembalikan ke *frontend*
- struct recordJSON, memiliki atribut Date, Name, Disease, Result, dan Similarity. Struktur data ini digunakan untuk menampung hasil dari API /predict.
- Struct historyJSON, memiliki atribut recordJSON yang merupakan sebuah array of recordJSON. Struktur data ini digunakan untuk menampung hasil dari API /history.
- Struct historyReqJSON, memiliki atribut Date dan Diseasename. Struktur data ini digunakan untuk menampung *body request* dari API /history.

Fungsi / Prosedur :

Terdapat lima fungsi utama dalam *backend* yang dibuat, yaitu

- Prosedur main yang tugasnya menjalankan server dan mendefinisikan API apa saja yang bisa diakses. Pada prosedur ini juga terdapat pemetaan API mana akan dijalankan menggunakan fungsi apa seperti yang dapat dilihat pada gambar di bawah ini

```
e.POST("/predict", predict)
e.POST("/history", history)
e.GET("/get", getDisease)
e.POST("/add", addDisease)
```

- Fungsi predict() yang melayani API /predict
- Fungsi history() yang melayani API /history
- Fungsi getDisease() yang melayani API /get
- Fungsi addDisease() yang melayani API /add

Cara kerja keempat fungsi terakhir sama persis seperti yang telah dijelaskan di bagian arsitektur aplikasi web. Secara umum, keempat fungsi tersebut akan mengambil *body request* bila ada, menginisiasi koneksi dengan basis data, berinteraksi dengan basis data (mengambil atau menambahkan data), lalu mengembalikan suatu nilai ke *frontend* bila diperlukan.

2. Penjelasan Tata Cara Penggunaan Program

Terdapat dua cara untuk menggunakan program yang telah digunakan. Cara pertama adalah dengan cara menjalankan aplikasi web di lokal dengan cara menjalankan perintah berikut

```
//masuk ke folder src/frontend dari folder repository ini
cd src/frontend
//pastikan sudah menginstall yarn
yarn start
```

Pengguna tidak perlu menjalankan server *backend* dan menginisiasi basis data di lokal karena *frontend* yang dijalankan dengan perintah di atas sudah terkoneksi dengan *backend* dan basis data di cloud yang telah di *deploy* menggunakan Heroku.

Cara kedua yang lebih mudah adalah dengan langsung mengakses aplikasi web yang sudah di *deploy* melalui alamat <https://bonek-dna.netlify.app/>

Deploy website dilakukan dengan dua *deploy* yang terpisah untuk *front-end* dan *back-end*/basis data. *Front-end* di-*deploy* di Netlify, sedangkan *back-end* dan basis data di-*deploy* di Heroku.

Untuk *deploy front-end*, buatlah sebuah *site* baru di Netlify. Letakkan semua komponen *front-end* pada folder *src/frontend* ke suatu *repository* terpisah. *Link repository* terpisah ini *site* yang sudah dibentuk sebelumnya. Pada titik ini seharusnya Netlify akan mencoba untuk selalu *men-deploy* bila terdapat perubahan pada *repository* tersebut. Penting bahwa untuk *men-deploy*, ubah *build command* pada bagian 'Build settings' menjadi command 'yarn build'. Pada saat ini, *front-end* seharusnya sudah berjalan dengan lancar.

Untuk *deploy back-end*, buatlah sebuah *app* baru di Heroku. Pada folder yang berisi *back-end*, yaitu folder *src/backend*, jalankan command:

```
heroku login
git init
heroku git:remote -a [APP_NAME]
git add .
git commit -am "[COMMIT_MESSAGE]"
```

```
git push heroku main
```

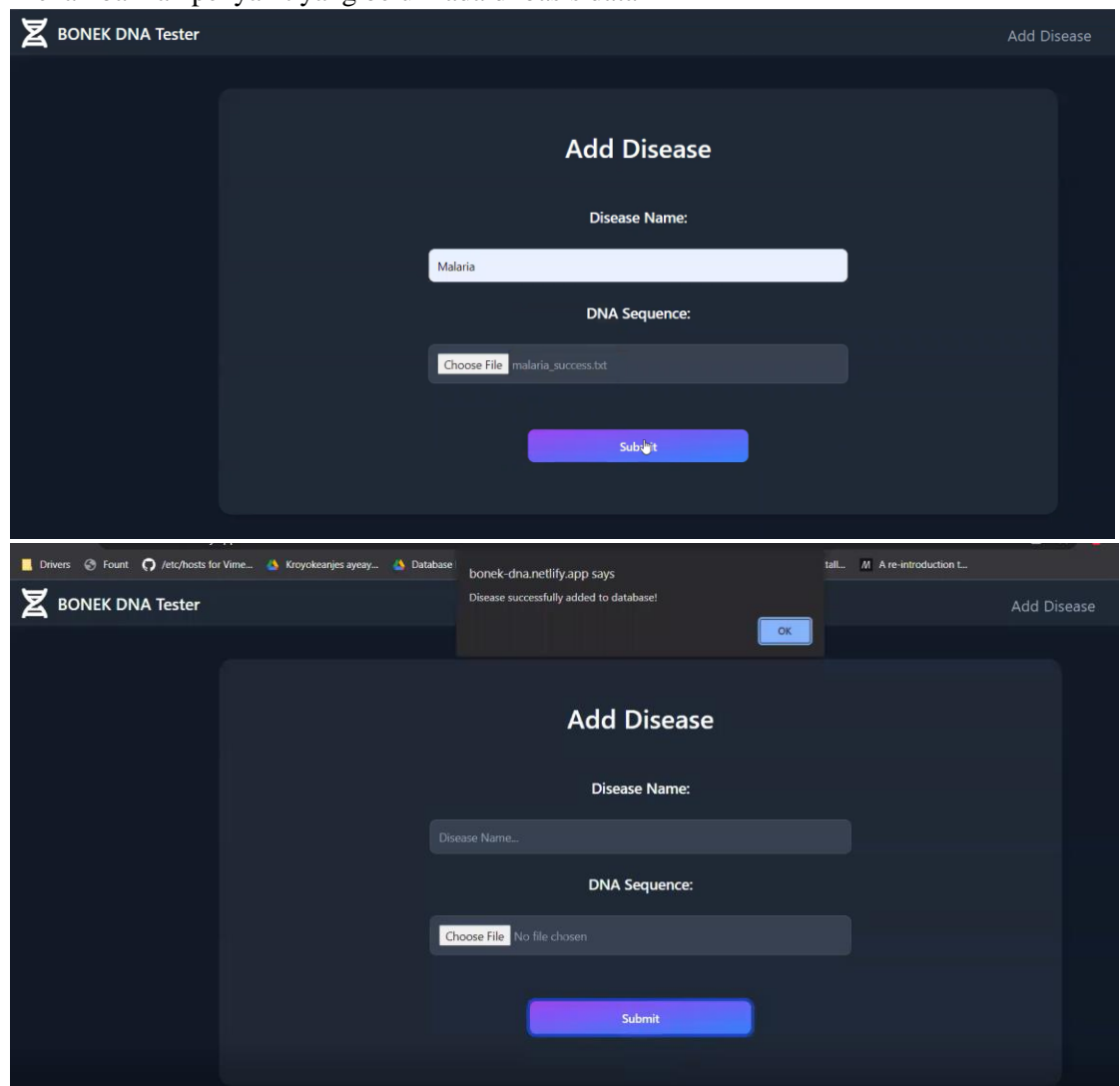
Dengan [APP_NAME] menyesuaikan nama App pada Heroku dna dan COMMIT_MESSAGE pesan untuk commit tersebut. Heroku kemudian akan men-*deploy back-end* pada halaman [APP_NAME].herokuapp.com. Perlu diperhatikan bahwa semua url **backend-bonek-dna.herokuapp.com** pada method GET dan POST yang dipanggil *axios* pada *front-end* perlu diganti dengan [APP_NAME].herokuapp.com.

Untuk *deploy* basis data, siapkan *dnamatching.sql* yang terdapat pada folder *src/backend*. Tambahkan add-on Heroku Postgres. Lalu, jalankan command:

```
heroku pg:psql --app [APP_NAME] < dnamatching.sql
```

3. Hasil Pengujian

- Menambahkan penyakit yang belum ada di basis data



name	dna
cacarair	ATGGTGACGAT
flu	ATGCTGACGAT
COVID-19	CACATAGATTG
thalasemia	AAAAA
turner	TGGCACTG
klinefelter	CCGGCGTACGCGTCCCATAT
diabetes	AAACCTGTCATAACTTACCT
sickle cell	GAGACTACTTGAAATGTGG
down syndrome	CTAGATCTTTGCCACGCAC
hemofilia	CTAATCGGTCCACGTTTGGT
kanker payudara	GCGGGTACTAGATGA
kanker serviks	CTGCAGGGACTCCGA
alzheimer	CGTTAAGTACATTAC
autis	CCGTCATAGGCGCC
migrain	GTTCAGGATCACGTT
arthritis	ACCGCCATAAGATGG
trisomy 18	GAGCATGACTTCTTC
trisomy 13	TCCGCTGCGCCACG
ambis	CCTATAACCTTCTG
bucin	TCCGCTGCGCCACG
Malaria	TTTTTATGGAGCTCG
(21 rows)	
-- More --	

- Menambahkan penyakit yang sudah ada di basis data

BONEK DNA Tester bonek-dna.netlify.app says Add Disease

Error! Disease name already exists! OK

Add Disease

Disease Name:

DNA Sequence:

Choose File No file chosen

Submit

- Pengujian DNA

The screenshot shows the BONEK DNA Tester application interface. The form is filled with the following data:

- Patient Name: Jova
- DNA Sequence: person_covid.txt
- Disease Prediction: COVID-19
- String Matching Algorithm: Knuth-Morris-Pratt

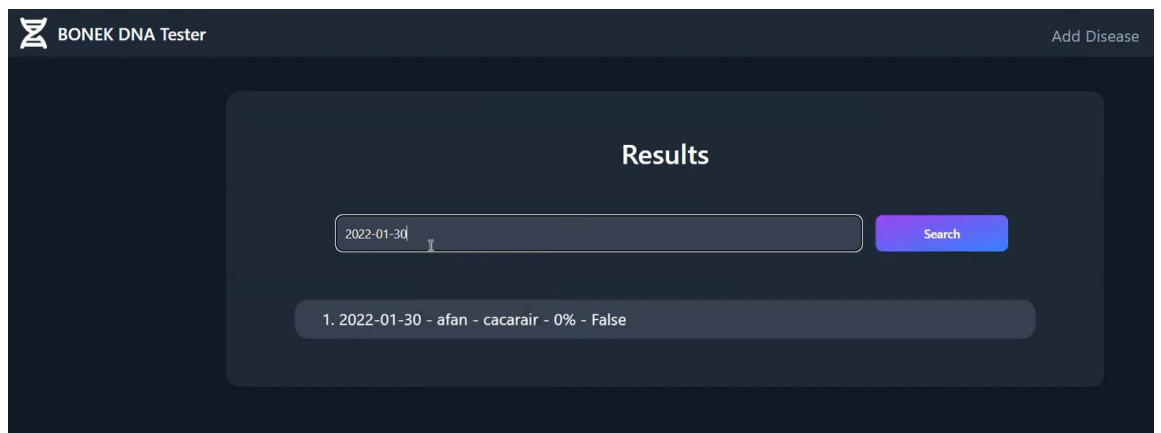
The Submit button is highlighted. A modal dialog box is displayed, showing the result: "Result using Knuth-Morris-Pratt Algorithm: 2022-04-28 - Jova - COVID-19 - 100% - True".

The screenshot shows the BONEK DNA Tester application interface. The form is filled with the following data:

- Patient Name: Liza
- DNA Sequence: person_random.txt
- Disease Prediction: autism
- String Matching Algorithm: Knuth-Morris-Pratt

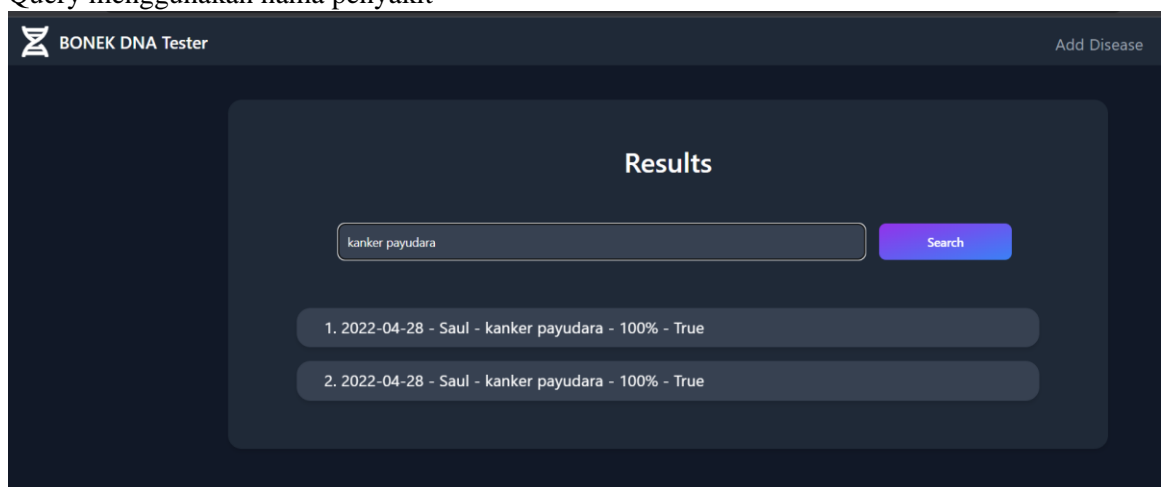
The Submit button is highlighted. A modal dialog box is displayed, showing the result: "Result using Knuth-Morris-Pratt Algorithm: 2022-04-28 - Liza - autism - 73% - False".

- Query menggunakan tanggal



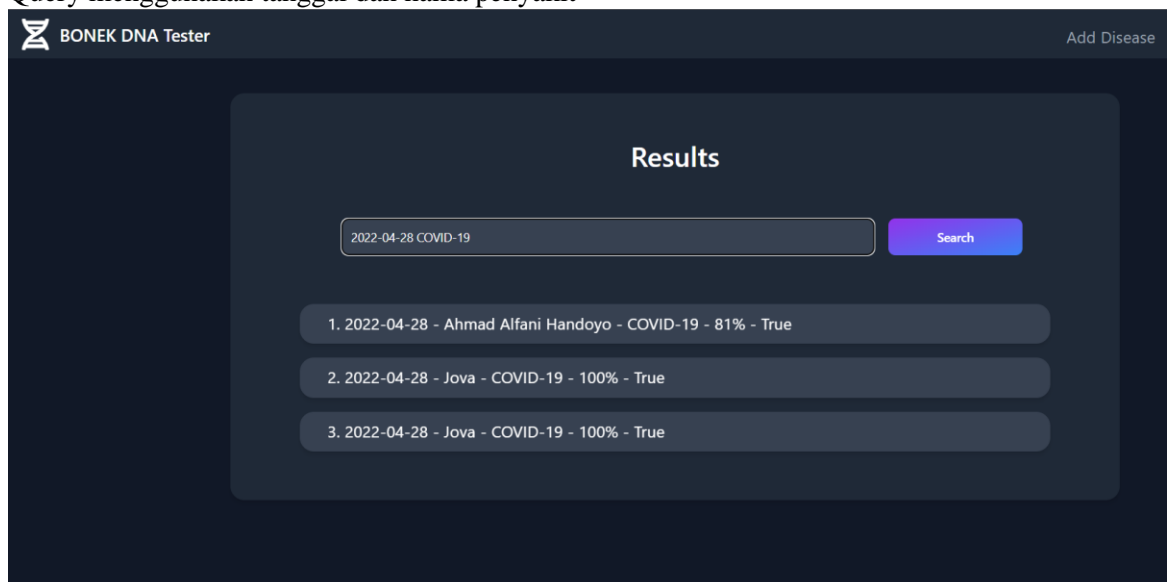
The screenshot shows the 'BONEK DNA Tester' interface. The header includes the logo and 'Add Disease' link. The main section is titled 'Results'. A search bar contains the text '2022-01-30' and a blue 'Search' button. Below the search bar, a single result is displayed in a light blue box: '1. 2022-01-30 - afan - cacarair - 0% - False'.

- Query menggunakan nama penyakit



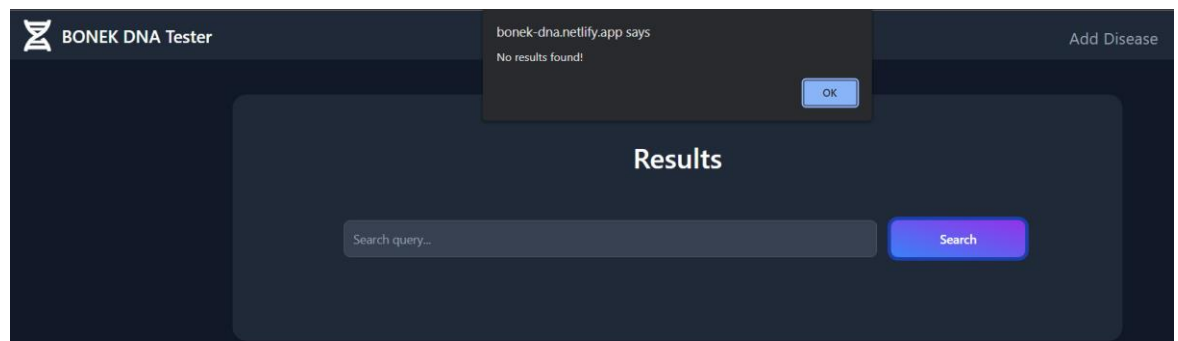
The screenshot shows the 'BONEK DNA Tester' interface. The header includes the logo and 'Add Disease' link. The main section is titled 'Results'. A search bar contains the text 'kanker payudara' and a blue 'Search' button. Below the search bar, two results are displayed in light blue boxes: '1. 2022-04-28 - Saul - kanker payudara - 100% - True' and '2. 2022-04-28 - Saul - kanker payudara - 100% - True'.

- Query menggunakan tanggal dan nama penyakit



The screenshot shows the 'BONEK DNA Tester' interface. The header includes the logo and 'Add Disease' link. The main section is titled 'Results'. A search bar contains the text '2022-04-28 COVID-19' and a blue 'Search' button. Below the search bar, three results are displayed in light blue boxes: '1. 2022-04-28 - Ahmad Alfani Handoyo - COVID-19 - 81% - True', '2. 2022-04-28 - Jova - COVID-19 - 100% - True', and '3. 2022-04-28 - Jova - COVID-19 - 100% - True'.

- Query dengan hasil nol baris



4. Analisis Hasil Pengujian

- Pada semua pengujian, hasil sudah sesuai seperti yang diharapkan
- Pengujian di *backend* sebenarnya menggunakan dua metode (KMP / boyer moore dan perhitungan kemiripan) dan kedua metode tersebut menghasilkan hasil yang konsisten, bila True selalu menghasilkan kemiripan 100% , yang menandakan alur jalannya algoritma yang sudah benar.
- Untuk melakukan pengujian, pengguna harus mengetahui ada nama penyakit apa saja yang ada di basis data. Padahal belum tentu semua pengguna tahu

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

1. Kesimpulan

Pada tugas besar mata kuliah IF2211 Strategi Algoritma yang ke-3 ini, telah berhasil diimplementasikan hasil pembelajaran algoritma string matching dengan Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) dalam bentuk aplikasi interaktif berbasis web yang berguna untuk melakukan sebuah prediksi apakah suatu sequence DNA pasien mengandung sequence DNA penyakit genetik tertentu.

Berdasarkan hasil pengerjaan kami terhadap tugas besar ini, dapat kami simpulkan bahwa dalam membuat suatu aplikasi berbasis web dapat dipisahkan menjadi dua bagian yaitu *front-end* dan *back-end*. Pada sisi *front-end*, aplikasi dapat menerima input baik dalam bentuk form ataupun file txt. Pengerjaan sisi *front-end* dapat dilakukan dengan berbagai bahasa, di antaranya HTML, CSS, Typescript. Untuk mempermudah melakukan pembuatan tampilan aplikasi, dapat digunakan beberapa framework seperti *react* dan *tailwind*. Untuk menggabungkan kedua sisi, maka diimplementasikan API yang dapat dipanggil oleh *front-end*. Sisi *back-end* dapat dikerjakan menggunakan bahasa pemrograman golang, dan untuk mempermudah pembuatan *api* dapat menggunakan framework echo.

Diperlukan suatu database untuk menyimpan data yang diperoleh dari pengguna. Salah satu database yang dapat digunakan dan dihubungkan dengan *back-end* adalah PostgreSQL. Dari hasil eksekusi algoritma KMP dan BM yang telah dibuat, dapat disimpulkan bahwa kedua algoritma ini secara efektif dapat melakukan pencocokan string yang lebih sangkil dibandingkan *brute-force*, baik dari segi efisiensi waktu ataupun banyaknya perbandingan yang dilakukan oleh program. Meskipun demikian, kesangkilan kedua algoritma tersebut cukup berbeda, bergantung oleh string yang diuji. Algoritma KMP lebih sangkil apabila banyak alfabet dalam string lebih sedikit, sementara algoritma BM lebih sangkil apabila banyak alfabet dalam string lebih besar. Dengan demikian, karena sequence DNA hanya terdiri dari 4 alfabet, dapat disimpulkan bahwa penggunaan algoritma Boyer-Moore lebih sangkil

Dengan demikian, kelompok kami dapat menyimpulkan bahwa dengan mengerjakan Tugas Besar III IF2211 Strategi Algoritma Semester 2 Tahun 2021/2022 ini, dapat diketahui bahwa kegiatan string matching dapat dilakukan dengan membuat sebuah program yang mengimplementasikan salah dua materi pembelajaran yaitu algoritma KMP dan BM.

2. Saran

Berdasarkan tugas besar yang telah kami lakukan, kami menyadari program kami baik dari sisi *front-end*, *back-end*, ataupun algoritma masih jauh dari kata sempurna sehingga kami berharap bahwa program kami masih dapat dikembangkan menjadi semakin semakin fungsional, sangkil, dan ringkas. Berikut adalah beberapa bagian yang masih dapat dikembangkan dari program kelompok kami :

- Menambah halaman yang dapat menampilkan seluruh data pada database untuk mempermudah pengecekan.
- Optimasi efisiensi algo dengan menggabungkan KMP atau BM sekaligus dengan LCS agar sekali kerja dan lebih sangkil.
- Mempelajari lebih lanjut mengenai penungguan data dan sifat tipe data dalam Typescript.

- Mempelajari lebih lanjut mengenai POST dan GET dalam pemanggilan API.

3. Refleksi

Kami ucapkan terimakasih kepada seluruh kakak asisten yang telah menjawab pertanyaan, membimbing, serta mengarahkan kami dalam semua Tugas Kecil ataupun Tugas Besar yang diberikan. Kami juga mengucapkan terimakasih kepada seluruh dosen pengampu mata kuliah IF2211 Strategi Algoritma yang telah mengajari kami dalam perkuliahan. Berkat Tugas – tugas yang diberikan ini, kami dapat semakin menguasai ilmu yang diberikan karena dapat secara langsung menerapkan dan mengimplementasikannya dalam bahasa pemrograman. Dalam konteks tugas besar 3 ini, kami menjadi semakin memahami bagaimana cara untuk membuat aplikasi interaktif yang berbasis web. Kami juga menyadari bahwa masih banyak yang belum kita kuasai dan masih banyak lagi bagian yang dapat dikembangkan.

LINK PENTING

Github Repository: https://github.com/saulsayerz/Tubes3_13520023

Link to Video : <https://youtu.be/wGxHUG6Gq4M>

Link to Deployed Website : <https://bonek-dna.netlify.app/>

DAFTAR PUSTAKA

Reactjs.org. Tutorial: Intro to React. Diakses pada 17 April 2022, dari <https://reactjs.org/tutorial/tutorial.html>

Medium.com. (2021, 20 October). How To Make API calls in React Applications. Diakses pada 19 April 2022, dari <https://medium.com/bb-tutorials-and-thoughts/how-to-make-api-calls-in-react-applications-7758052bf69>

Techmoro.com. (2021, 9 December). Submit Form Data To REST API In A React App. Diakses pada 19 April 2022, dari <https://www.techmoro.com/submit-a-form-data-to-rest-api-in-a-react-app/>

Sciencedirect.com. (2016, 25 July). LCS: A refined similarity measure. Diakses pada 22 April 2022, dari <https://www.sciencedirect.com/science/article/pii/S0304397515010877>

Informatika.stei.itb.ac.id/~rinaldi.munir. (2021). Stima #23 Pencocokan string (String matching/pattern matching). Diakses pada 22 April 2022, dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>

Informatika.stei.itb.ac.id/~rinaldi.munir. (2019). Stima #24 Pencocokan string dengan Regular Expression (Regex). Diakses pada 22 April 2022, dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf>

Baeldung.com. (2021, 13 March). String Similarity Metrics: Sequence Based. Diakses pada 22 April 2022, dari <https://www.baeldung.com/cs/string-similarity-sequence-based#:~:text=The%20longest%20common%20subsequence%20finds,to%20establish%20a%20similarity%20measure>

Academia.edu. (2017, July). Sequence Similarity between Genetic Codes using Improved Longest Common Subsequence Algorithm. Diakses pada 22 April 2022, dari https://www.academia.edu/36906994/Sequence_Similarity_between_Genetic_Codes_using_Improved_Longest_Common_Subsequence_Algorithm