

LAPORAN TUGAS BESAR
IF2211 STRATEGI ALGORITMA

**PENERAPAN STRING MATCHING DAN REGULAR
EXPRESSION DALAM DNA *PATTERN* MATCHING**

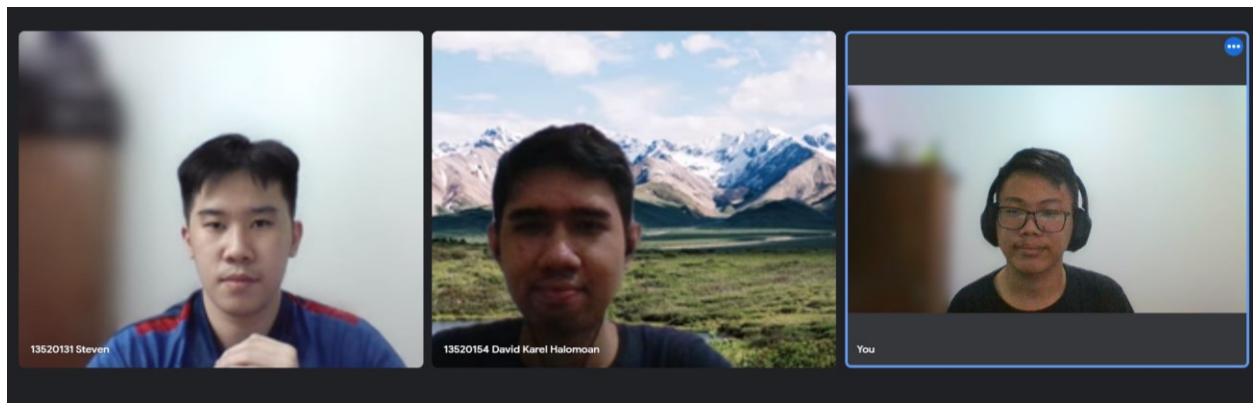
Disusun oleh:

Kelompok DNAsaurus

Suryanto 13520059

Steven 13520131

David Karel Halomoan 13520154



TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

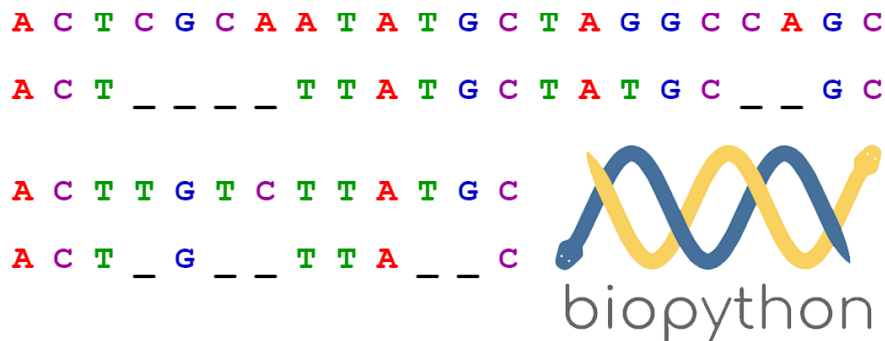
DAFTAR ISI

BAB I DESKRIPSI TUGAS	2
BAB II LANDASAN TEORI.....	3
2.1 Algoritma Knuth-Morris-Pratt	3
2.2 Algoritma Boyer-Moore	4
2.3 <i>Regular Expression</i>	5
2.4 DNAsaurus.....	6
BAB III ANALISIS PEMECAHAN MASALAH.....	7
3.1 Langkah-Langkah Pemecahan Masalah.....	7
3.2 Fitur Fungsional dan Arsitektur Aplikasi Web yang dibangun.....	7
BAB IV IMPLEMENTASI DAN PENGUJIAN	9
4.1 Spesifikasi Teknis Program	9
4.2 Penjelasan Tata Cara Penggunaan Program.....	10
4.3 Hasil Pengujian	12
4.4 Analisis Hasil Pengujian	18
BAB V KESIMPULAN DAN SARAN.....	20
5.1 Kesimpulan	20
5.2 Saran	20
5.3 Refleksi	20
LAMPIRAN.....	21
DAFTAR PUSTAKA	22

BAB I

DESKRIPSI TUGAS

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.

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 aplikasiinteraktif untuk mendeteksi apakah seorang pasien mempunyai penyakit genetik tertentu. Hasil prediksi tersebut dapat disimpan pada basis data untuk kemudian dapat ditampilkan berdasarkan query pencarian.

BAB II LANDASAN TEORI

2.1 Algoritma Knuth-Morris-Pratt

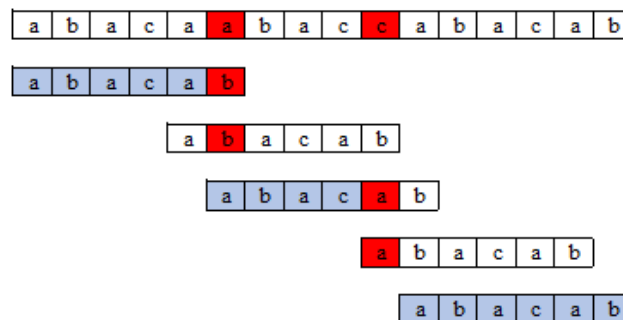
Algoritma Knuth-Morris-Pratt(KMP) adalah suatu algoritma pencocokkan string yang mirip dengan algoritma *brute force*. Pencocokkan dilakukan dari kiri ke kanan dengan pergeseran yang ditentukan dengan *border function*, sehingga algoritma ini lebih efektif dibandingkan dengan *brute force* yang melakukan pergeseran secara satu-persatu.

Pada pencocokkan *pattern P* pada *text T*, apabila terjadi *mismatch* pada posisi karakter $T[i]$ dan $P[j]$, maka algoritma ini akan melakukan pergeseran seukuran panjang dari *prefix* terpanjang dari $P[0..j-1]$ yang juga merupakan *suffix* dari $P[1..j-1]$. Oleh karena itu, terdapat digunakan sebuah fungsi yang dikenal sebagai *border/failure function* yang digunakan untuk menentukan ukuran pergeseran untuk tiap kejadian *mismatch* pada indeks P .

Misal akan dilakukan pencocokkan string pada *text T* = *abacaabaccabacab* dengan *pattern P* = *abacab*. Pada *mismatch pertama* ($T[5]$ dan $P[5]$), maka pencocokkan selanjutnya akan dilakukan pada $T[5]$ dan $P[b(5-1)]$. Dengan nilai $b(4) = 1$, maka $P[0]$ tidak perlu diperiksa lagi karena pencocokkan akan dimulai pada $P[1]$. Pencocokkan akan terus dilakukan hingga tidak terjadi *mismatch* pada $P[j] = T[i]$ dengan j adalah indeks terakhir dari P . Kasus lainnya adalah saat T telah mencapai indeks terakhir tanpa menemukan *match* dengan P , yang berarti *pattern P* tidak ditemukan pada *text T*.

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

Gambar 2 Ilustrasi Border Function



Gambar 3 Ilustrasi Pencocokan dengan Algoritma KMP

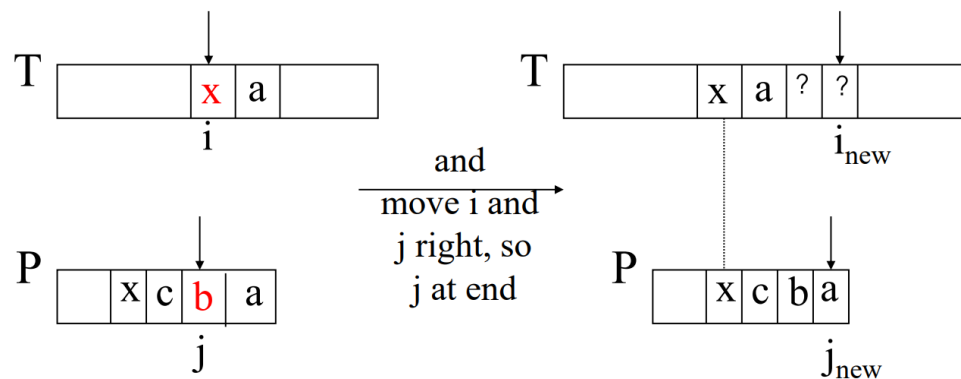
2.2 Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) adalah suatu algoritma pencocokkan string yang menggunakan dua buah teknik, yaitu *looking-glass technique* dan *character-jump technique*. *Looking-glass technique* adalah teknik mencari *pattern* P pada *text* T dengan pencocokkan mundur melalui *pattern* P, dimulai dari akhirnya. Sedangkan *character-jump technique* adalah teknik yang digunakan untuk melakukan “pergeseran” *pattern* P pada saat ditemukan ketidakcocokan saat melakukan *looking-glass technique*.

Terdapat 3 kemungkinan kasus untuk *character-jump technique*, dengan urutan pengecekan kasus dari terawal ke terakhir:

- Kasus 1

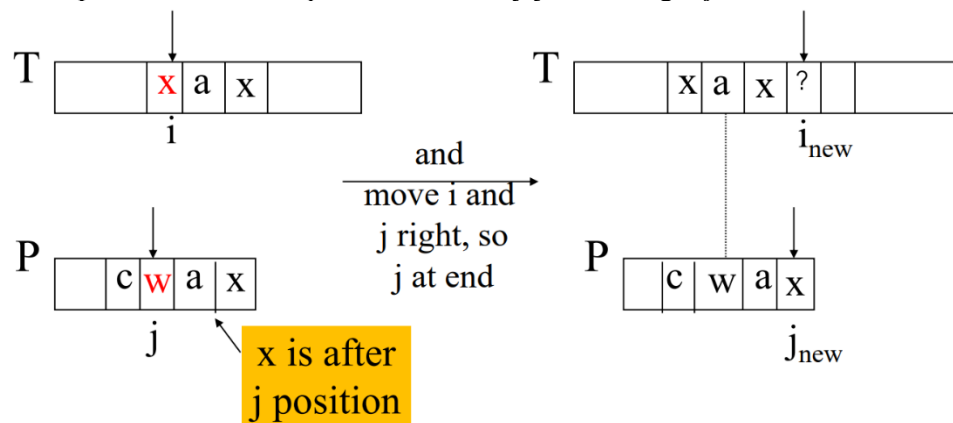
Misal ditemukan ketidakcocokan karakter pada indeks i pada *text* dan indeks j pada *pattern* dan karakter pada indeks i pada *text* adalah x . Jika *pattern* mengandung x pada indeks sebelah kiri j (lebih kecil dari j), *pattern* digeser ke kanan sampai karakter x pada *pattern* “sejajar” dengan karakter x pada *text* yang berada pada indeks i (karakter x pada *pattern* yang digunakan adalah yang terakhir (paling kanan), $T[i]$ disejajarkan dengan karakter x terakhir pada *pattern*). Indeks j lalu dipindahkan ke akhir *pattern* dan i disejajarkan dengan j .



Gambar 4 Ilustrasi Pencocokan dengan Algoritma Boyer-Moore (1)

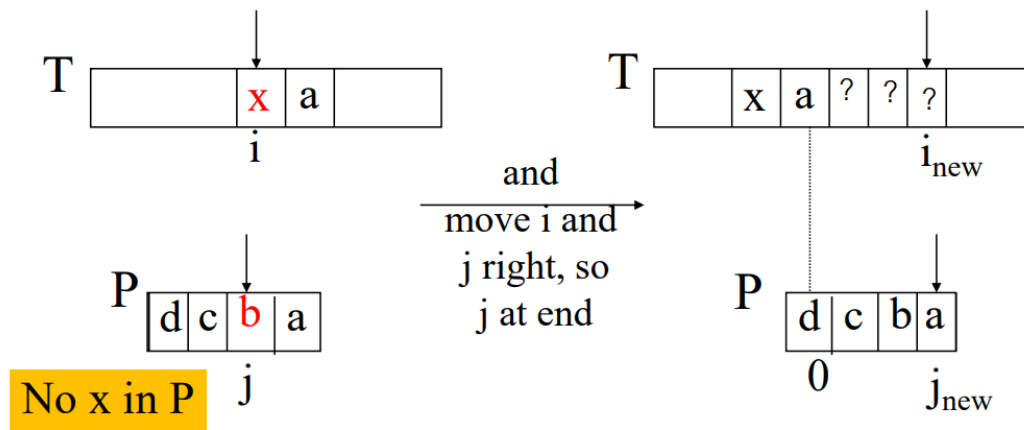
- Kasus 2

Misal ditemukan ketidakcocokan karakter pada indeks i pada *text* dan indeks j pada *pattern* dan karakter pada indeks i pada *text* adalah x . Jika *pattern* mengandung x pada indeks sebelah kanan j (lebih besar dari j), *pattern* digeser ke kanan sejauh satu karakter ($P[j]$ disejajarkan dengan $T[i+1]$). Indeks j lalu dipindahkan ke akhir *pattern* dan i disejajarkan dengan j .



Gambar 5 Ilustrasi Pencocokan dengan Algoritma Boyer-Moore (2)

- Kasus 3
Misal ditemukan ketidakcocokan karakter pada indeks i pada *text* dan indeks j pada *pattern* dan karakter pada indeks i pada *text* adalah x . Jika *pattern* tidak mengandung x , *pattern* digeser ke kanan sampai karakter pertama pada *pattern* sejajar dengan karakter yang berada di sebelah kanan x ($P[0]$ disejajarkan dengan $T[i+1]$). Indeks j lalu dipindahkan ke akhir *pattern* dan i disejajarkan dengan j .



Gambar 6. Ilustrasi Pencocokan dengan Algoritma Boyer-Moore (3)



Setelah melakukan *character-jump technique*, algoritma akan kembali melakukan *looking-glass technique*. Hal ini akan terus dilakukan sampai *pattern* ditemukan pada *text* atau *pattern* tidak ditemukan sampai akhir *text*.

Algoritma Boyer-Moore juga menggunakan suatu fungsi yang disebut *last occurrence function* (dilambangkan dengan $L()$). $L()$ memetakan semua kemungkinan karakter pada *pattern* dan *text* menjadi sebuah bilangan bulat (*integer*). $L(x)$ didefinisikan sebagai indeks i terbesar pada *pattern* dengan $P[i]$ sama dengan x atau -1 jika karakter tidak ditemukan pada *pattern*. Fungsi ini biasanya disimpan dalam struktur data yang berbentuk tabel, seperti *array*. Fungsi ini dikalkulasikan saat algoritma pertama kali membaca *pattern*.

P.

2.3 Regular Expression

Regular expression (regex) adalah notasi standar yang mendeskripsikan suatu pola (*pattern*) berupa urutan karakter atau string. Regex dapat digunakan untuk pencocokan string (string matching) dengan efisien karena dapat memeriksa sebuah *pattern* pada teks dengan lebih luas. Misalnya terdapat sebuah notasi regex berupa "[Ii][Tt][Bb]", maka dapat diperiksa apakah terdapat *pattern* "ITB" (baik ditulis sebagai kapital atau tidak, bahkan kombinasi kapital dan non-kapital) pada sebuah teks.

Regex book	Version History	Feedback	Blog
Options		Quick Reference	
.	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 <i>n</i> of the preceding element.		
{n,}	<i>n</i> or more of the preceding element.		
{m,n}	Between <i>m</i> and <i>n</i> of the preceding element.		
??,*?+,?	Same as above, but as few as possible.		
{n}?, etc.			
(expr)	Capture <i>expr</i> for use with \1, etc.		
(?:expr)	Non-capturing group.		
(?=expr)	Followed by <i>expr</i> .		
(?!expr)	Not followed by <i>expr</i> .		
Near-complete reference			

Gambar 7. Aturan Penulisan dengan Regular Expression

2.4 DNAsaurus

DNAsaurus adalah aplikasi berbasis web yang dibangun dengan *framework* Vue.js dan basisdata MongoDB. Aplikasi ini dapat melakukan berbagai perintah seperti menambahkan penyakit, memprediksi penyakit berdasarkan DNA, dan melihat *history* prediksi. Seluruh program di atas diimplementasikan dengan memanfaatkan algoritma *string matching* dan *regular expression*.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

- a. Menambahkan *Sequence* penyakit baru
 - Melakukan pencocokkan *sequence* penyakit dengan regex untuk memastikan bahwa hanya memuat Karakter A,C,G,dan T saja.
 - Jika *sequence* DNA memiliki karakter ilegal (selain dari keempat karakter yang telah disebutkan), maka akan dimunculkan pesan kesalahan dan penyakit gagal dimasukkan ke dalam basis data.
 - Jika *sequence* valid, maka *sequence* akan dimasukkan ke dalam basisdata.
- b. Melakukan pencocokkan DNA seseorang dengan penyakit
 - Melakukan pencocokkan *sequence* DNA dengan regex untuk memastikan bahwa hanya memuat Karakter A,C,G,dan T saja.
 - Jika *sequence* DNA memiliki karakter ilegal, maka akan dimunculkan pesan kesalahan bahwa terdapat kesalahan dari DNA user.
 - Jika *sequence* valid, maka akan dilakukan pencocokan string dengan algoritma KMP untuk mengecek apakah pengguna menderita penyakit yang diprediksi. Apabila ditemukan, maka informasi tingkat kemiripan akan disimpan sebesar 100%.
 - Jika algoritma KMP gagal menemukan *pattern* penyakit pada DNA pengguna, maka akan dilakukan pencocokan dengan algoritma Hamming. Pengguna akan disimpulkan memiliki penyakit apabila DNA memiliki tingkat kemiripan lebih besar atau sama dari 80% dengan penyakit yang diprediksi.
 - Informasi pengecekan akan disimpan pada basisdata.
- c. Mengecek *history* pengecekan yang telah dilakukan
 - Melakukan pengecekan string pada *query* pencarian dengan menggunakan regex
 - Apabila *query* hanya berupa tanggal (contoh : 27 April 2022), maka akan dilakukan pencarian pada database terkait prediksi penyakit yang dilakukan pada tanggal yang bersangkutan.
 - Apabila *query* hanya berupa nama penyakit (contoh : malaria), maka akan dilakukan pencarian pada database terkait prediksi penyakit yang berhubungan dengan nama penyakit bersangkutan.
 - Pencarian juga dapat dilakukan dengan *query* yang merupakan gabungan dari tanggal prediksi dan penyakit yang diprediksi (contoh : 27 April 2022 malaria).
 - Menampilkan hasil dari pencarian dengan format Tanggal – nama_user – nama_penyakit – keterangan_mengidap – tingkat_kecocokan. Misal 27 April 2022 – user1 – malaria – false – 63.1579%.

3.2 Fitur Fungsional dan Arsitektur Aplikasi Web yang dibangun

- a. Fitur Fungsional
 - Menambahkan penyakit baru ke basisdata penyakit (*disease*)
 - Melakukan prediksi penyakit pada DNA seseorang
 - Menampilkan tingkat kecocokan *sequence* DNA seseorang dengan *sequence* DNA penyakit
 - Menampilkan *history* prediksi dengan berbagai kategori, diantaranya adalah per tanggal, per nama penyakit, dan kombinasi keduanya

- Menampilkan pesan error jika masukan *sequence* DNA tidak valid.
- b. Arsitektur Aplikasi DNAsaurus:
- *Frontend*
Frontend dari DNAsaurus menggunakan *framework* Vue JS. Aplikasi terdiri dari 4 *pages*, yaitu *Home Page*, *Insert Disease Page*, *Check Disease Page*, dan *List Disease Page*. Terdapat komponen *back button* yang diterapkan pada setiap *page*, kecuali *Home Page*.
 - *Backend*
Terdapat beberapa file yang berisi algoritma dan kode yang dibutuhkan oleh aplikasi, seperti *kmp.js* yang memuat algoritma KMP, *boyerMoore.js* yang memuat algoritma BM, dan *dbFunction* yang memuat *method-method* yang digunakan untuk mengakses basisdata. Basisdata aplikasi ini dibangun menggunakan teknologi MongoDB.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

a. Struktur Data

Data yang disimpan pada sisi basisdata disimpan dalam bentuk JSON (Javascript Object Notation). Berikut adalah contoh JSON yang terdapat pada basisdata DNAsaurus:

```
User = {
  _id : ObjectId("6269473bac33c8fa121b8826"),
  date : '27 April 2022',
  name : 'user5',
  disease : 'hepatitis',
  isInfected : false,
  percentage : '50.0000'
};

Disease = {
  id : ObjectId("626946c1ac33c8fa121b8822"),
  name : '19covid',
  gene : 'ACG'
}
```

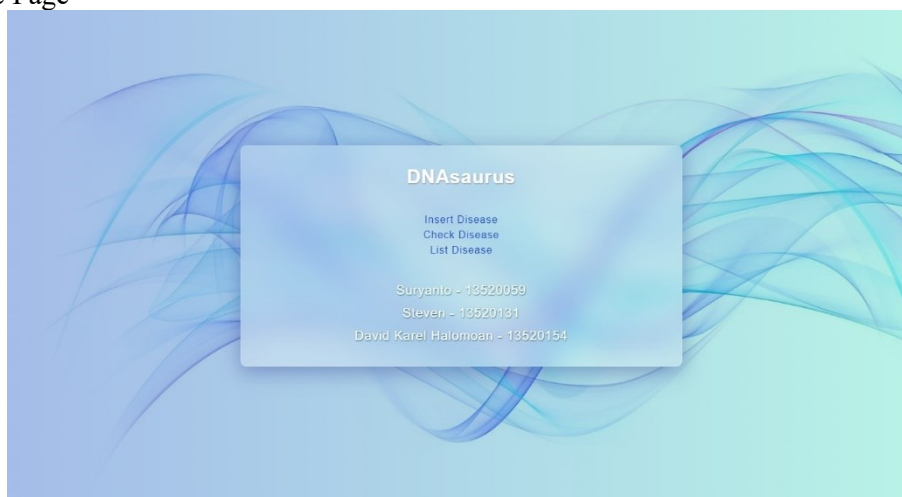
b. Fungsi dan Prosedur

No	Nama Fungsi/Prosedur	Keterangan
1	computeFail(<i>pattern</i>)	Fungsi yang mengembalikan sebuah array yang merupakan <i>border function</i> dari setiap karakter dari <i>pattern</i>
2	kmpMatch(<i>text</i> , <i>pattern</i>)	Implementasi dari algoritma KMP yang akan memeriksa apakah terdapat <i>pattern</i> pada <i>text</i> . Apabila ditemukan <i>pattern</i> pada <i>text</i> , maka akan dikembalikan <i>integer</i> yang merupakan indeks pada <i>text</i> yang sesuai dengan <i>pattern</i> . Sebaliknya, fungsi akan mengembalikan nilai -1.
3	bmMatch(<i>text</i> , <i>pattern</i>)	Implementasi dari algoritma Boyer-Moore. Memiliki fungsionalitas dan karakteristik pengembalian yang sama dengan kmpMatch, yaitu mengembalikan indeks pada <i>text</i> jika ditemukan <i>pattern</i> pada <i>text</i> atau nilai -1 jika tidak ditemukan.
4	buildLast(<i>pattern</i>)	Fungsi yang mengembalikan array yang menyimpan indeks dari <i>last occurrence</i> tiap karakter pada <i>pattern</i> .
5	insertNewDisease(mongoClient, diseaseName, diseaseGene)	Fungsi untuk menambahkan penyakit dan <i>sequence</i> -nya ke dalam basisdata.

6	insertNewUser(mongoClient, date, name, disease, isInfected, percentage)	Fungsi yang menambahkan data user ke dalam basisdata. Fungsi ini akan dipanggil setiap user melakukan prediksi penyakit.
7	getDiseaseGene(mongoClient, name)	Fungsi untuk mendapatkan <i>sequence</i> DNA dari suatu penyakit berdasarkan nama penyakit.
8	getUserFromDate(mongoClient, date)	Fungsi untuk mendapatkan informasi user berdasarkan tanggal melakukan prediksi penyakit.
9	getUserFromDisease(mongoClient, diseaseName)	Fungsi untuk mendapatkan informasi user berdasarkan penyakit yang dicek.
10	getUserFromDateAndDisease(mongoClient, date, diseaseName)	Fungsi untuk mendapatkan informasi user berdasarkan tanggal dan penyakit yang dicek.
11	getUser(mongoClient)	Fungsi untuk mengambil semua informasi user pada database
12	getMatch(str1, str2)	Fungsi untuk mendapatkan banyaknya karakter yang sama untuk string str1 dan str2.
13	hamming(str1, str2)	Fungsi untuk melakukan pemeriksaan dengan <i>hamming distance</i> apabila tidak ditemukan kecocokan dengan algoritma KMP.
14	autocomplete(str1, str2)	Fungsi untuk melakukan pemeriksaan pencocokan string guna membantu dalam mengenerate autocomplete hasil pada list disease view

4.2 Penjelasan Tata Cara Penggunaan Program

a. Home Page



Gambar 8. Home Page dari Website Aplikasi DNAsaurus

Pada *Home Page*, tersedia 3 buah *text* yang dapat diketik yaitu “*Insert Disease*”, “*Check Disease*”, dan *List Disease*” yang di mana masing-masing *text* tersebut akan men-*direct ke* laman yang berbeda-beda. Tidak hanya itu, terdapat pula nama kelompok dan nama anggota kelompok pada laman ini.

b. Insert Disease Page

Gambar 9. Insert Disease Page dari Website Aplikasi DNAsaurus

Pada *Insert Disease Page*, terdapat *input box* bertipe *text* yang di mana dapat diisi dengan nama penyakit yang ingin ditambahkan. Tidak hanya itu, terdapat juga *input box* type *file* yang dimana dapat diisi dengan file bertipe *txt* yang berisi karakter A/C/T/G. Apabila isi file terdapat karakter selain A/C/T/G, maka program akan mengembalikan error kepada pengguna apabila pengguna menekan tombol submit.

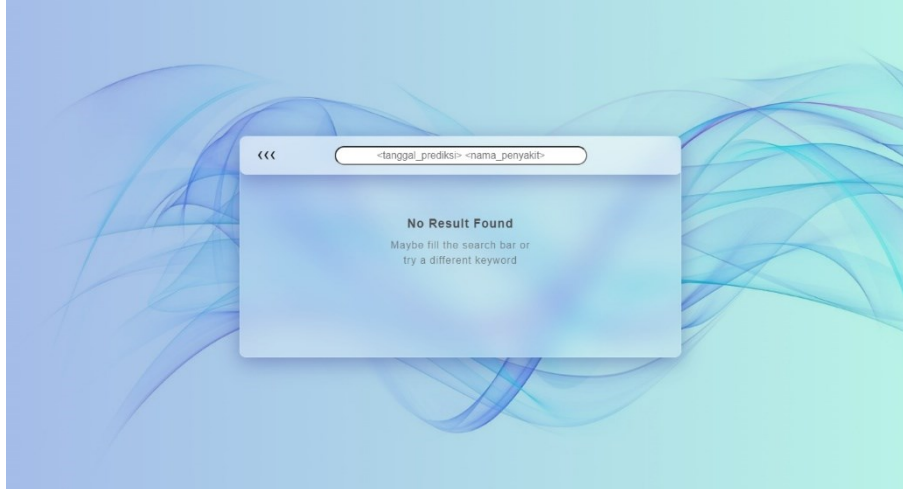
c. Check Disease Page

Gambar 10. CheckDisease Page dari Website Aplikasi DNAsaurus

Pada *Check Disease Page*, terdapat 2 buah *input box* bertipe *text* dan 1 buah *input box* bertipe *file*. *Input box* bertipe *text* ini digunakan untuk menginput nama pengguna dan nama penyakit yang

ingin diprediksi oleh pengguna sedangkan *input box* bertipe *file* digunakan untuk menginput file rantai DNA pengguna yang di mana berlaku juga seperti laman *Insert Disease Page*, apabila fila yang dimasukkan pengguna mengandung karakter lain selain A/C/T/G, maka akan dikembalikan pesan error apabila pengguna menekan tombol submit.

d. List Disease Page



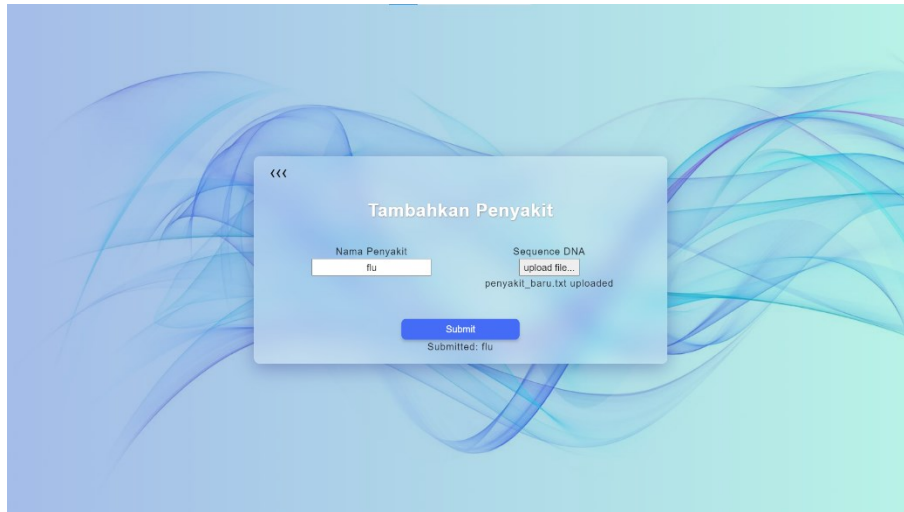
Gambar 11. List Disease Page dari Website Aplikasi DNAsaurus

Pada *List Disease Page*, terdapat sebuah *input box* bertipe *text* yang di mana dapat diisi dengan tanggal atau nama penyakit, ataupun tanggal dan nama penyakit sekaligus. Setelah pengguna berhenti memberikan inputan selama waktu yang ditentukan (di set 200ms), maka program akan langsung menerima inputan pengguna dan mengeksekusinya dengan cara mencocokkan apakah inputan pengguna memenuhi format tanggal dan/atau nama penyakit dengan menggunakan bantuan regex. Setelah itu, akan dilakukan pengecekan pada database untuk melihat apakah terdapat data yang sesuai dengan inputan pengguna. Apabila ada, maka akan dikembalikan data tersebut. Akan tetapi, apabila gagal, maka akan dikembalikan pesan “*No Result Found Maybe fill the search bar or try a different keyword*”. Untuk input tanggal dan nama penyakit serta input nama penyakit saja, diimplementasikan fitur mirip autocomplete yang di mana apabila user mengetik $\geq 80\%$ dari nama penyakit, maka akan dicari nama penyakit yang memiliki kemiripan paling tidak 80% dari inputan pengguna lalu menampilkannya pada layar.

4.3 Hasil Pengujian

Berikut adalah tangkapan layar dari skenario-skenario yang terjadi pada aplikasi DNAsaurus.

- a. Menambahkan Penyakit Baru
 - Penambahan Berhasil



Gambar 12. Screenshot yang menunjukkan input penyakit berhasil

- Penambahan gagal



Gambar 13. Screenshot yang menunjukkan input penyakit gagal

b. Prediksi Penyakit

- Prediksi berhasil dan User Terinfeksi(100%)



Gambar 14. Screenshot yang menunjukkan prediksi berhasil dengan tingkat kemiripan 100%

- Prediksi Berhasil dan User Terinfeksi ($\geq 80\%$)



Gambar 15. Screenshot yang menunjukkan prediksi berhasil dengan tingkat $\geq 80\%$

- Prediksi Berhasil dan User Tidak Terinfeksi ($<80\%$)



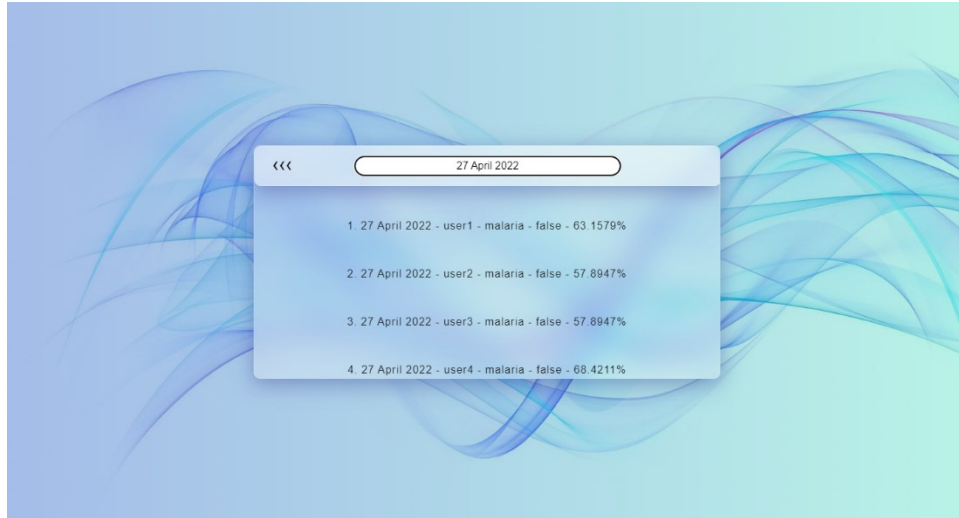
Gambar 16. Screenshot yang menunjukkan prediksi gagal dengan tingkat kemiripan <80%

- Prediksi Gagal (Sequence DNA Tidak Valid)



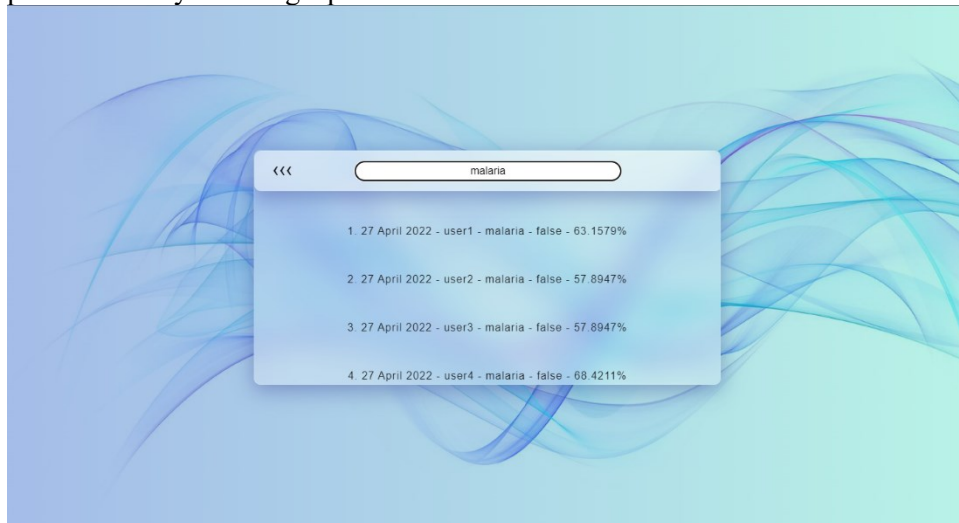
Gambar 17. Screenshot yang menunjukkan prediksi gagal karena sequence DNA user yang tidak valid

- c. Menampilkan History Prediksi Penyakit
 - Input Tanggal Prediksi



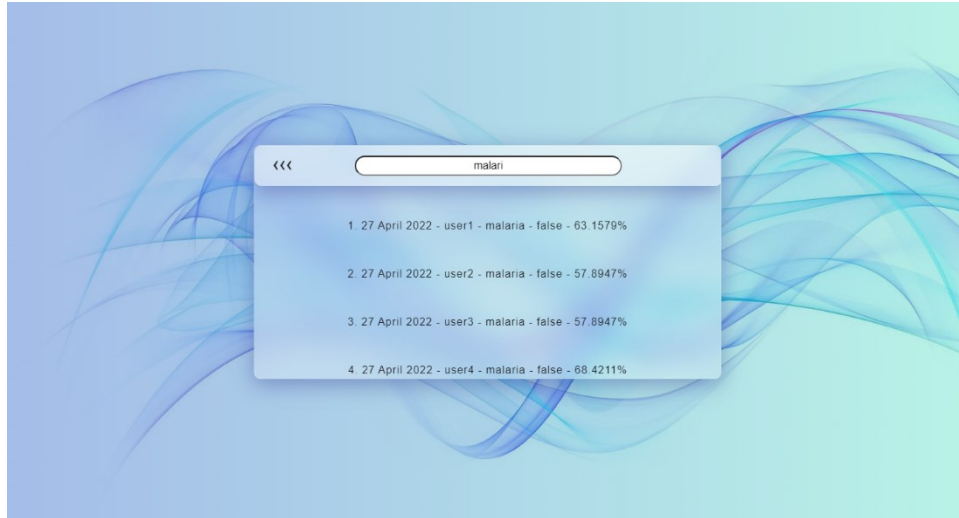
Gambar 18. Screenshot yang menunjukkan history pemeriksaan berdasarkan tanggal

- Input Nama Penyakit Lengkap



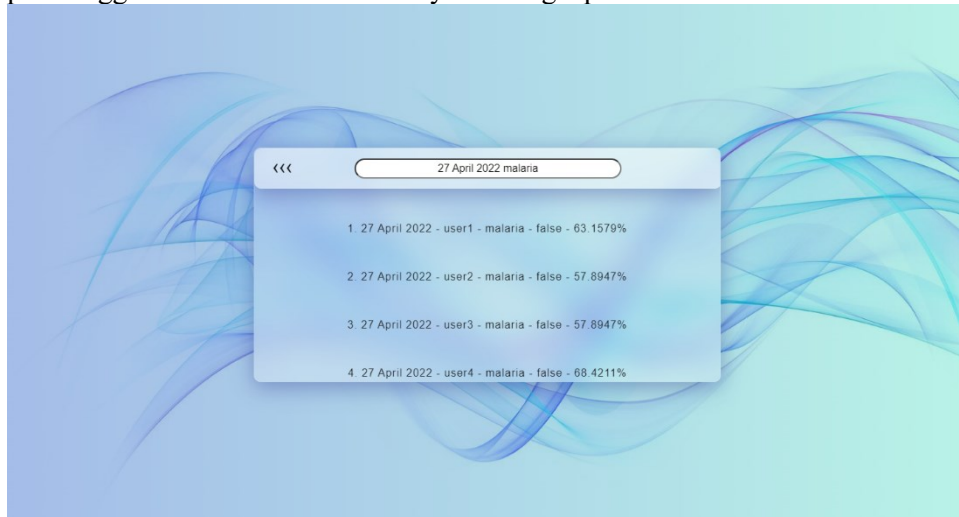
Gambar 19. Screenshot yang menunjukkan history pemeriksaan berdasarkan nama penyakit lengkap

- Input Nama Penyakit Tidak Lengkap



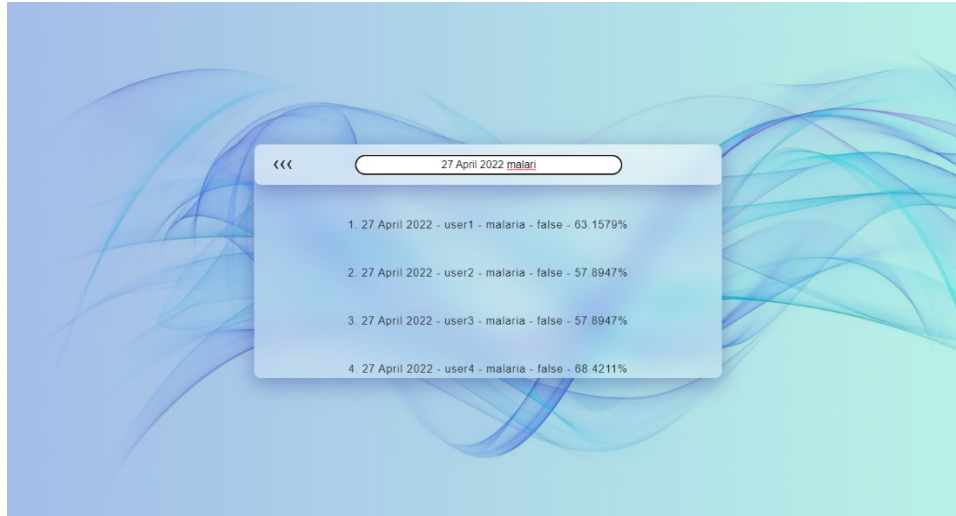
Gambar 20. Screenshot yang menunjukkan history pemeriksaan berdasarkan nama penyakit tidak lengkap

- Input Tanggal Prediksi dan Nama Penyakit Lengkap



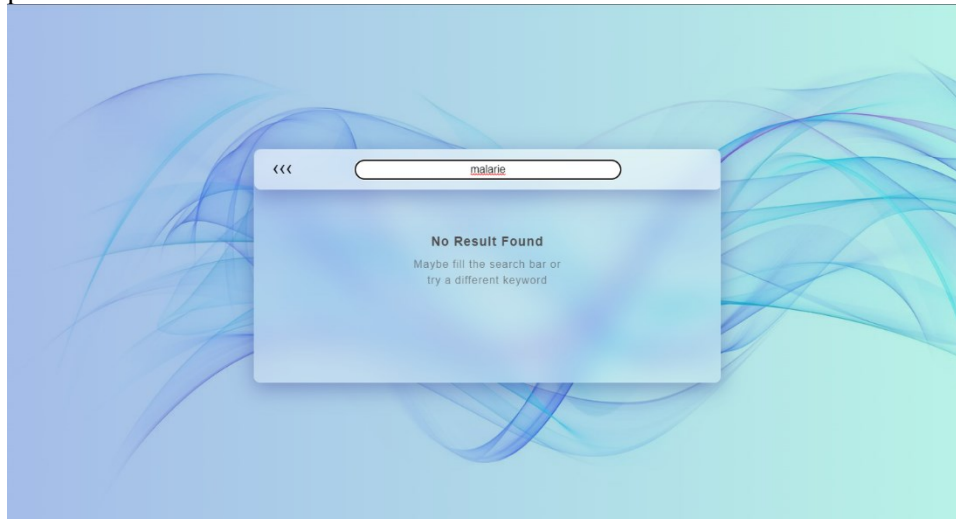
Gambar 21. Screenshot yang menunjukkan history pemeriksaan berdasarkan tanggal pemeriksaan beserta nama penyakit lengkap

- Input Tanggal Prediksi dan Nama Penyakit Tidak Lengkap



Gambar 22. Screenshot yang menunjukkan history pemeriksaan berdasarkan tanggal pemeriksaan beserta nama penyakit tidak lengkap

- Input Salah



Gambar 23. Screenshot yang menunjukkan history pemeriksaan dengan masukan yang tidak ada pada basisdata

4.4 Analisis Hasil Pengujian

Kami memutuskan untuk menggunakan algoritma KMP untuk menyelesaikan tugas besar kali ini. Kami memilih algoritma ini karena algoritma ini merupakan algoritma yang baik untuk *text* dan *pattern* yang memiliki alfabet (kemungkinan karakter) yang sedikit, sehingga cocok dengan tugas besar ini yang hanya memiliki 4 alfabet, yaitu A, C, G, dan T. Algoritma ini juga tidak pernah bergerak mundur dalam *text* masukan sehingga cocok untuk memproses *text* yang memiliki ukuran besar. Ini berbeda dengan algoritma Boyer-Moore yang dikenal cukup lambat dalam memproses *text* dan *pattern* yang memiliki alfabet yang sedikit sehingga tidak cocok untuk tugas besar ini.

Dari hasil pengujian yang kami lakukan, algoritma ini dikombinasikan dengan regex berhasil mengenali *pattern* penyakit yang berada dalam *text* dengan performa yang cukup baik dan sesuai hasil yang diharapkan.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Aplikasi DNA *Pattern Matching* bernama DNAsaurus berhasil dibangun dengan memanfaatkan algoritma *string matching* dan *regular expression*. Kedua algoritma *string matching* yaitu algoritma KMP dan Boyer-Moore telah diimplementasikan, tetapi kelompok kami memutuskan untuk menggunakan KMP karena algoritma ini dapat bekerja lebih baik pada kasus ini. Aplikasi telah memenuhi keseluruhan spesifikasi serta bonus baik pemrosesan tingkat kemiripan DNA maupun pen-*deploy*-an aplikasi secara daring.

5.2 Saran

Penulis menyarankan pembaca untuk melakukan lebih banyak eksplorasi terkait pengembangan website, baik dari sisi *frontend* maupun dari sisi *backend*. Aplikasi ini juga dapat dikembangkan menjadi lebih baik lagi jika dengan pemanfaatan regex secara lanjut.

5.3 Refleksi

Penulis merasa tugas ini cukup menarik karena langsung mengimplementasikan algoritma yang telah diajarkan di kelas. Sehingga penulis dapat lebih memahami algoritma terkait *string matching*, terutama algoritma KMP dan Boyer-Moore. Selain itu, penulis juga dapat belajar terkait pengembangan aplikasi website melalui eksplorasi yang dilakukan dalam pengerjaan tugas besar ini.

LAMPIRAN

Repository Development : https://github.com/SurTan02/Tubes3_13520059
Repository Deployment : <https://github.com/loopfree/DNAsaurus>
Tautan Aplikasi Web : <https://dnasaurus.herokuapp.com/>

DAFTAR PUSTAKA

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

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf>