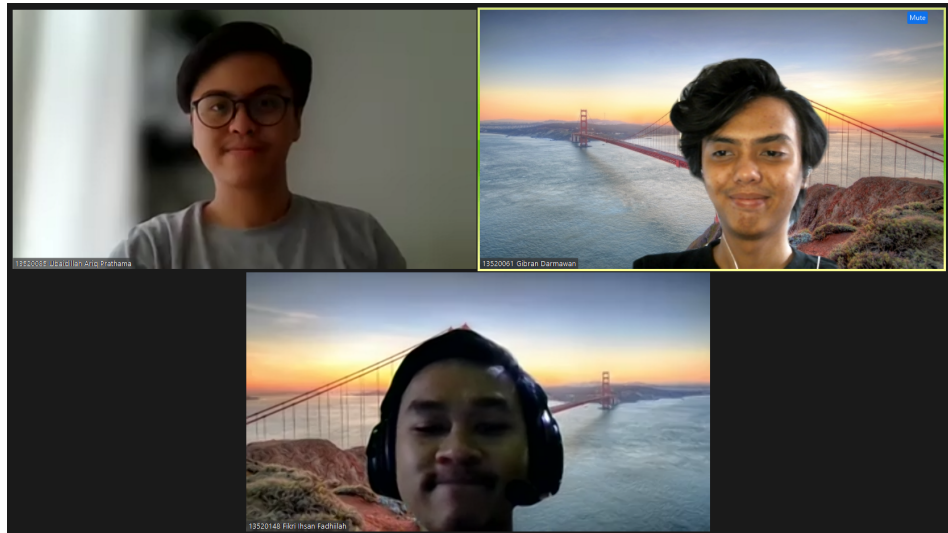


**LAPORAN TUGAS BESAR STRATEGI ALGORITMA**  
**Penerapan String Matching dan Regular Expression dalam DNA**  
**Pattern Matching**  
**IF2211 STRATEGI ALGORITMA**



**Dosen pengajar : Dr. Masayu Leylia Khodra, S.T., M.T.**

**Kelompok : RS Borromeus**

**Disusun oleh :**

**Gibran Darmawan (13520061)**

**Ubaidillah Ariq Prathama (13520085)**

**Fikri Ihsan Fadhiilah (13520148)**

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SETKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2021**

## **BAB I**

### **DESKRIPSI TUGAS**

Dalam tugas besar ini, kami diminta untuk membangun sebuah aplikasi DNA Pattern Matching. Dengan memanfaatkan algoritma String Matching dan Regular Expression yang telah kami pelajari di kelas IF2211 Strategi Algoritma, kami 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).
  - 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).
2. Aplikasi dapat memprediksi seseorang menderita penyakit tertentu berdasarkan sequence DNA-nya.
  - a. Tes DNA dilakukan dengan menerima input nama pengguna, sequence DNA pengguna, dan nama penyakit yang diuji. Asumsi sequence DNA pengguna > sequence DNA penyakit.
  - 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, tidak ada spasi, dll).
  - c. Pencocokan sequence DNA dilakukan dengan menggunakan algoritma string matching.
  - d. 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
  - e. Semua komponen hasil tes ini dapat ditampilkan pada halaman web (refer ke poin 3 pada “Fitur-Fitur Aplikasi”) dan disimpan pada sebuah tabel database
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: , 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.
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.

## **BAB II**

### **LANDASAN TEORI**

#### **A. Algoritma Knuth-Morris-Pratt**

Algoritma Knuth-Morris-Pratt adalah salah satu algoritme pencarian string. Jika kita melihat algoritme brute force lebih mendalam, kita mengetahui bahwa dengan mengingat beberapa perbandingan yang dilakukan sebelumnya kita dapat meningkatkan besar pergeseran yang dilakukan. Hal ini akan menghemat perbandingan, yang selanjutnya akan meningkatkan kecepatan pencarian.

Perhitungan penggeseran pada algoritme ini adalah sebagai berikut, bila terjadi ketidakcocokkan pada saat pattern sejajar dengan teks[ $i..i+n-1$ ], kita bisa menganggap ketidakcocokan pertama terjadi di antara teks[ $i+j$ ] dan pattern[ $j$ ], dengan  $0 < j < n$ . Berarti, teks[ $i..i+j-1$ ]=pattern[ $0..j-1$ ] dan  $a=\text{teks}[i+j]$  tidak sama dengan  $b=\text{pattern}[j]$ . Ketika kita menggeser, sangat beralasan bila ada sebuah awalan  $v$  dari pattern akan sama dengan sebagian akhiran  $u$  dari sebagian teks. Sehingga kita bisa menggeser pattern agar awalan  $v$  tersebut sejajar dengan akhiran dari  $u$ .

Dengan kata lain, pencocokkan 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 algoritme Knuth-Morris-Pratt pada saat mencocokkan string:

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

Algoritma ini menemukan semua 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. Semua besaran  $O$  tersebut tidak tergantung pada besarnya ruang alfabet.

#### **B. Algoritma Boyer-Moore**

Algoritme Boyer-Moore adalah salah satu algoritme pencarian string. Algoritme ini dianggap sebagai algoritme yang paling efisien pada aplikasi umum. Tidak seperti algoritme pencarian string yang ditemukan sebelumnya, algoritme Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritme ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat.

Misalnya ada sebuah usaha pencocokan yang terjadi pada teks[i..i+n-1], dan anggap ketidakcocokan pertama terjadi di antara 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=teks[i+j] tidak sama dengan b=pattern[j]. Jika u adalah akhiran dari pattern sebelum b dan v adalah sebuah awalan dari pattern, maka penggeseran-penggeseran yang mungkin adalah:

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

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

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

Tabel untuk penggeseran bad-character dan good-suffix dapat dihitung dengan kompleksitas waktu dan ruang sebesar  $O(n + \sigma)$  dengan  $\sigma$  adalah besar ruang alfabet. Sedangkan pada fase pencarian, algoritme ini membutuhkan waktu sebesar  $O(mn)$ , pada kasus terburuk, algoritme ini akan melakukan  $3n$  pencocokkan karakter, namun pada performa terbaiknya algoritme ini hanya akan melakukan  $O(m/n)$  pencocokkan.

### C. Regular Expression

Regular Expression atau Regex adalah serangkaian karakter yang mendefinisikan sebuah *pola pencarian*. Pola tersebut biasanya digunakan oleh algoritma pencarian string

untuk melakukan operasi "cari" atau "cari dan ganti" pada string, atau untuk memeriksa string masukan. Ekspresi reguler merupakan teknik yang dikembangkan dalam bidang ilmu komputer teori dan teori bahasa formal. Berikut beberapa sintaks yang umum pada regex:

```
\      // the escape character - used to find an instance of a metacharacter like a period, brackets, etc.
.      // match any character except newline
x      // match any instance of x
^x     // match any character except x
[x]    // match any instance of x in the bracketed range - [abxyz] will match any instance of a, b, x, y, or z
|      // an OR operator - [x|y] will match an instance of x or y
()     // used to group sequences of characters or matches
{}     // used to define numeric quantifiers
{x}    // match must occur exactly x times
{x,}   // match must occur at least x times
{x,y}  // match must occur at least x times, but no more than y times
?      // preceding match is optional or one only, same as {0,1}
*      // find 0 or more of preceding match, same as {0,}
+      // find 1 or more of preceding match, same as {1,}
^      // match the beginning of the line
$      // match the end of a line

\d     // matches a digit, same as [0-9]
\D     // matches a non-digit, same as [^0-9]
\s     // matches a whitespace character (space, tab, newline, etc.)
\S     // matches a non-whitespace character
\w     // matches a word character
\W     // matches a non-word character
\b     // matches a word-boundary (NOTE: within a class, matches a backspace)
\B     // matches a non-wordboundary
```

## D. Arsitektur Web

### a. Frontend

Frontend untuk web yang kami buat menggunakan framework React.js. React membuat mungkin sebuah web yang konsepnya hanya terdiri dari satu web yang terdiri dari beberapa komponen. Komponen ini dapat diakses menggunakan routes. Untuk kerangka web sendiri tentunya tetap menggunakan html dan juga css untuk styling. Untuk melakukan request ke backend digunakan axios sebagai alat bantu untuk request ke api backend.

### b. Backend

Backend untuk web yang kami buat menggunakan bahasa Go dengan framework Go Echo. Go Echo merupakan framework untuk membuat backend Go yang minimalis tetapi memiliki performa yang baik. Hal utama yang dipakai dalam backend web kami adalah middleware dan route. Route ini akan dipanggil oleh frontend untuk membuat

request yang dibutuhkan frontend. Pada setiap fungsi route terdapat pengambilan json dari frontend, pengolahan data menggunakan algoritma yang ada, search dan insert ke database, serta return hasil ke frontend. Algoritma sendiri terdapat pada folder algorithm yang dapat diimport dan digunakan.

c. Database

Database untuk web yang kami buat menggunakan PostgreSQL. Postgre pada dasarnya mirip dengan SQL pada umumnya dengan beberapa fitur tambahan. Database ini, kami hosting pada heroku. Backend dapat mengakses dan mengubah database dengan cara membuat koneksi ke database dengan credential yang ada.

### **BAB III**

#### **ANALISIS PEMECAHAN MASALAH**

##### **A. Langkah Penyelesaian Masalah Setiap Fitur**

###### **a. Input Penyakit Baru**

User akan mengisi input pada text box, terdapat dua text box untuk nama penyakit dan file string dna penyakit dalam format txt. Ketika klik tombol submit, akan dihandle pada frontend kasus ketika terdapat field yang kosong dan akan menyebabkan error. Jika kedua field sudah terisi dan diklik tombol submit, frontend akan melakukan request ke backend dengan menggunakan Axios. Semua ini terjadi di frontend, dengan menggunakan React.js pada component AddPenyakit.

Backend akan menerima request dari frontend dengan method POST, url /disease, dan body berupa json yang berisi string nama penyakit dan string DNA. Json ini akan diparse menjadi string. DNA akan dicek valid atau tidak (hanya terdiri dari ACGT) menggunakan library algorithm yang telah dibuat dan diimplementasikan dengan regex. Jika tidak valid, akan mengirimkan return message “DNA Tidak Valid” ke frontend dan akan ditampilkan. Jika valid, akan dicek terlebih dahulu apakah penyakit tersebut sudah ada di database menggunakan query. Jika belum ada, tambahkan ke database menggunakan query lalu mengirimkan return message “Penyakit berhasil ditambahkan” ke frontend dan akan ditampilkan. Jika sudah ada, akan mengirimkan return message “Penyakit sudah ada” ke frontend dan akan ditampilkan.

###### **b. Prediksi Penyakit dan menghitung tingkat kemiripan**

User akan mengisi input pada text box, terdapat dua text box untuk nama user nama penyakit, satu file input untuk string DNA, dan satu pilihan untuk pemilihan algoritma (KMP/Boyer-Moore). Ketika klik tombol submit, akan dihandle pada frontend kasus ketika terdapat field yang kosong dan akan menyebabkan error. Jika keempat field sudah terisi dan diklik tombol submit, frontend akan melakukan request ke backend dengan menggunakan Axios. Semua ini terjadi di frontend, dengan menggunakan React.js pada component DataFill dan DNAPage.

Backend akan menerima request dari frontend dengan method POST, url /test, dan body berupa json yang berisi string nama pasien, string DNA, string nama penyakit, dan string penanda algoritma mana yang akan dipakai. Json ini akan diparse menjadi string. DNA akan dicek valid atau tidak (hanya terdiri dari ACGT) menggunakan library algorithm yang telah dibuat dan diimplementasikan dengan regex. Jika tidak valid, akan mengirimkan return message “DNA Tidak Valid” ke frontend dan akan ditampilkan. Jika valid, akan dicek terlebih dahulu apakah penyakit tersebut sudah ada di database menggunakan query. Jika belum



ada, akan mengirimkan return message “Penyakit tidak ditemukan” ke frontend dan akan ditampilkan. Jika sudah ada, akan dicek kesamaan string yang akan dites dengan string DNA penyakit dari database sesuai dengan algoritma yang dipilih. Jika terdapat string penyakit pada string yang akan dites, akan dikirimkan return message berupa json yang berisi tanggal, nama, penyakit, status, dan persentase (100%) ke frontend dan akan ditampilkan. Jika tidak sama, akan dicek kemiripannya menggunakan hamming distance lalu akan dikirimkan return berupa json yang berisi tanggal, nama, penyakit, status, dan persentase. Algoritma KMP, Boyer-Moore, dan Hamming Distance sudah diimplementasikan pada library algorithm. Selain itu, apapun hasil test yang didapat, hasil test ini akan dimasukkan ke dalam database hasil\_prediksi menggunakan query.

c. Pencarian Hasil Prediksi

User akan mengisi input pada text box, terdapat satu text box untuk search query dalam bentuk string. Query dapat berupa tanggal, penyakit, ataupun keduanya. Ketika klik tombol submit, akan dihandle pada frontend kasus ketika terdapat field yang kosong dan akan menyebabkan error. Jika field sudah terisi dan diklik tombol submit, frontend akan melakukan request ke backend dengan menggunakan Axios. Semua ini terjadi di frontend, dengan menggunakan React.js pada component SearchResult.

Backend akan menerima request dari frontend dengan method POST, url /result, dan body berupa json yang berisi string search query. Json ini akan diparse menjadi string. Query akan divalidasi terlebih dahulu menggunakan fungsi dari library algorithm yang diimplementasikan menggunakan regex. Jika tidak valid, akan dikirimkan return message “Query tidak valid” ke frontend dan akan ditampilkan. Jika valid, query akan diparse menjadi dua string yaitu string tanggal dan string penyakit. Kemudian, akan dicari di database hasil\_prediksi, apakah terdapat rows yang memiliki tanggal dan penyakit yang sesuai. Jika tidak ada, akan dikirimkan return message “Tidak ada history yang cocok” ke frontend dan akan ditampilkan. Jika ada, akan dikirimkan return berupa json yang berisi semua atribut pada tabel hasil\_prediksi di database yang memenuhi search query ke frontend. Karena jumlah rows tidak tentu, digunakan fungsi map pada React.js untuk menampilkannya.

B. Arsitektur Web

d. Frontend

Frontend untuk web yang kami buat menggunakan framework React.js. React membuat mungkin sebuah web yang konsepnya hanya terdiri dari satu web yang terdiri dari beberapa komponen. Komponen ini dapat diakses menggunakan routes. Untuk kerangka web sendiri tentunya tetap menggunakan html dan juga

css untuk styling. Untuk melakukan request ke backend digunakan axios sebagai alat bantu untuk request ke api backend.

e. Backend

Backend untuk web yang kami buat menggunakan bahasa Go dengan framework Go Echo. Go Echo merupakan framework untuk membuat backend Go yang minimalis tetapi memiliki performa yang baik. Hal utama yang dipakai dalam backend web kami adalah middleware dan route. Route ini akan dipanggil oleh frontend untuk membuat request yang dibutuhkan frontend. Pada setiap fungsi route terdapat pengambilan json dari frontend, pengolahan data menggunakan algoritma yang ada, search dan insert ke database, serta return hasil ke frontend. Algoritma sendiri terdapat pada folder algorithm yang dapat diimport dan digunakan.

f. Database

Database untuk web yang kami buat menggunakan PostgreSQL. PostgreSQL pada dasarnya mirip dengan SQL pada umumnya dengan beberapa fitur tambahan. Database ini, kami hosting pada heroku. Backend dapat mengakses dan mengubah database dengan cara membuat koneksi ke database dengan credential yang ada.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### A. Spesifikasi Teknis Program

##### a. Struktur Data

Struktur data buatan yang cukup penting adalah struct yang digunakan untuk menyimpan dan passing informasi ke frontend dalam bentuk json, yaitu:

```
type hasilPrediksi struct {  
    Id          int    `json:"id"`  
    Nama        string `json:"nama"`  
    Tanggal     string `json:"tanggal"`  
    Penyakit     string `json:"penyakit"`  
    Status      string `json:"status"`  
    Persentase  int    `json:"persentase"`  
}  
  
type penyakit struct {  
    Id          int    `json:"id"`  
    Penyakit    string `json:"penyakit"`  
    DNA        string `json:"dna"`  
}  
  
type message struct {  
    Message string `json:"message"`  
}
```

Struktur data lainnya cenderung straightforward (list dan map) dan sudah ada sebagai bawaan dari Go dan juga JavaScript.

##### b. Fungsi dan Prosedur

Berikut adalah fungsi dan prosedur pada library algorithm, yang terkait dengan string matching dan regex:

```
//Border untuk KMP  
func border(s string) []int {
```

```

    n := len(s) - 1
    next := make([]int, n)
    next[0] = -1
    j := -1
    for i := 1; i < n; i++ {
        for j != -1 && s[i] != s[j+1] {
            j = next[j]
        }
        if s[i] == s[j+1] {
            j++
        }
        next[i] = j
    }
    return next
}

//KMP Algorithm
//Return indeks pertama string ditemukan, jika tidak ada
return -1
func KMP(s, t string) int {
    next := border(t)
    i, j := 0, 0
    for i < len(s) && j < len(t) {
        if s[i] == t[j] {
            i++
            j++
        } else {
            if j == 0 {
                i++
            } else {
                j = next[j-1] + 1
            }
        }
    }
    if j == len(t) {
        return i - j
    }
    return -1
}

//Mencari last occurence dari setiap karakter

```

```

func lastOccurence(s string) map[string]int {
    last := make(map[string]int)
    for i := 0; i < len(s); i++ {
        last[string(s[i])] = i
    }
    return last
}

//Boyer Moore Algorithm
//Return indeks pertama string ditemukan, jika tidak ada
return -1

func BoyerMoore(s, t string) int {
    last := lastOccurence(t)
    i, j := len(t)-1, len(t)-1
    for i < len(s) && j < len(t) {
        if s[i] == t[j] {
            if j == 0 {
                return i
            } else {
                i--
                j--
            }
        } else {
            if last, ok := last[string(s[i])]; ok {
                if last+1 > j {
                    i += len(t) - j
                } else {
                    i += len(t) - last - 1
                }
                j = len(t) - 1
            } else {
                i += len(t)
                j = len(t) - 1
            }
        }
    }
    if j == 0 {
        return i
    }
    return -1
}

```

```

}

func HammingDistance(s, t string) int {
    count := 0
    max := 0
    for i := 0; i+len(t) <= len(s); i++ {
        count = 0
        for j := 0; j < len(t); j++ {
            if s[i+j] == t[j] {
                count++
            }
        }
        if count > max {
            max = count
        }
    }
    return 100 * max / len(t)
}

func ValidateInput(s string) bool {
    regex, err := regexp.Compile("^[ACGT]+$")
    if err != nil {
        fmt.Println("Error: ", err)
        return false
    }
    return regex.MatchString(s)
}

func ValidateQuery(s string) int {
    regex1, err :=
regexp.Compile(`^(31(\\|-|\\.|\\s)(0[13578]|1[02]|(Januari|Maret
|Mei|Juli|Agustus|Oktober|Desember)) (\\|-|\\.|\\s)((1[6-9]|[2-9]
\\d)\\d{2})) (\\s)(.*)$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex1.MatchString(s) {
        return 1
    }
}

```

```

    regex1, err =
regexp.Compile(`^(31(\\/-|\\.|\\s)(0[13578]|1[02]|(Januari|Maret
|Mei|Juli|Agustus|Oktober|Desember))\\/-|\\.|\\s)((1[6-9]|[2-9]
\\d)\\d{2}))$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex1.MatchString(s) {
        return 2
    }
    regex2, err :=
regexp.Compile(`^((29|30)(\\/-|\\.|\\s)(0[1,3-9]|1[0-2]|(Januari
|Maret|April|Mei|Juni|Juli|Agustus|September|Oktober|November|
Desember))\\/-|\\.|\\s)(([6-9]|[2-9]\\d)\\d{2}))\\s)(.*)$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex2.MatchString(s) {
        return 1
    }
    regex2, err =
regexp.Compile(`^((29|30)(\\/-|\\.|\\s)(0[1,3-9]|1[0-2]|(Januari
|Maret|April|Mei|Juni|Juli|Agustus|September|Oktober|November|
Desember))\\/-|\\.|\\s)(([6-9]|[2-9]\\d)\\d{2}))$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex2.MatchString(s) {
        return 2
    }
    regex3, err :=
regexp.Compile(`^(29(\\/-|\\.|\\s)(02|(Februari))\\/-|\\.|\\s)((
1[6-9]|[2-9]\\d)(0[48]|[2468][048]|[13579][26]))\\s)(.*)$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }

```

```

    if regex3.MatchString(s) {
        return 1
    }
    regex3, err =
regexp.Compile(`^(29(\\/-|\\.|\\s) (02| (Februari)) (\\/-|\\.|\\s) (((
1[6-9]|[2-9]\\d) (0[48]|[2468][048]|[13579][26]))))$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex3.MatchString(s) {
        return 2
    }
    regex4, err :=
regexp.Compile(`^(29(\\/-|\\.|\\s) (02| (Februari)) (\\/-|\\.|\\s) ((1
6|[2468][048]|[3579][26])00)) (\\s) (.*)$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex4.MatchString(s) {
        return 1
    }
    regex4, err =
regexp.Compile(`^(29(\\/-|\\.|\\s) (02| (Februari)) (\\/-|\\.|\\s) ((1
6|[2468][048]|[3579][26])00))$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex4.MatchString(s) {
        return 2
    }
    regex5, err :=
regexp.Compile(`^((0[1-9]|1\\d|2[0-8]) (\\/-|\\.|\\s) (0[1-9]| (Janu
ari|Februari|Maret|April|Mei|Juni|Juli|Agustus|September)) (\\/-
|\\.|\\s) ((1[6-9]|[2-9]\\d)\\d{2})) (\\s) (.*)$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }

```



```

    }
    if regex5.MatchString(s) {
        return 1
    }
    regex5, err =
regexp.Compile(`^((0[1-9]|1\d|2[0-8]) (\/|-|\.\s) (0[1-9]| (Janu
ari|Februari|Maret|April|Mei|Juni|Juli|Agustus|September)) (\/|
-|\.\s) ((1[6-9]|[2-9]\d)\d{2}))$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex5.MatchString(s) {
        return 2
    }
    regex6, err :=
regexp.Compile(`^((0[1-9]|1\d|2[0-8]) (\/|-|\.\s) (1[0-2]| (Okto
ber|November|Desember)) (\/|-|\.\s) ((1[6-9]|[2-9]\d)\d{2})) (\s
)(.*)$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex6.MatchString(s) {
        return 1
    }
    regex6, err =
regexp.Compile(`^((0[1-9]|1\d|2[0-8]) (\/|-|\.\s) (1[0-2]| (Okto
ber|November|Desember)) (\/|-|\.\s) ((1[6-9]|[2-9]\d)\d{2}))$`)
    if err != nil {
        fmt.Println("Error: ", err)
        return -1
    }
    if regex6.MatchString(s) {
        return 2
    }
    return 3
}

func ParseQuery(s string, n int) []string {

```

```

var result []string
i := 0
if n == 3 {
    query := []string{"", s}
    return query
}
for j := 0; j < len(s); j++ {
    if j == len(s)-1 {
        result = append(result, s[i:])
        i = j + 1
    }
    if s[j] == ' ' || s[j] == '/' || s[j] == '-' {
        result = append(result, s[i:j])
        i = j + 1
        if len(result) == 3 {
            break
        }
    }
}
if i == len(s) {
    result = append(result, "")
} else {
    result = append(result, s[i:])
}
if len(result[0]) == 1 {
    result[0] = "0" + result[0]
}
bulan := []string{"Januari", "Februari", "Maret", "April",
"Mei", "Juni", "Juli", "Agustus", "September", "Oktober",
"November", "Desember"}
for i := 0; i < len(bulan); i++ {
    if result[1] == bulan[i] {
        if i+1 < 10 {
            result[1] = fmt.Sprintf("0%d", i+1)
        } else {
            result[1] = fmt.Sprintf("%d", i+1)
        }
        break
    }
}
}

```

```

    query := []string{fmt.Sprintf("%s-%s-%s", result[2],
result[1], result[0]), result[3]}
    return query
}

```

## B. Cara Penggunaan Program

### a. Cara Menjalankan Program di Local

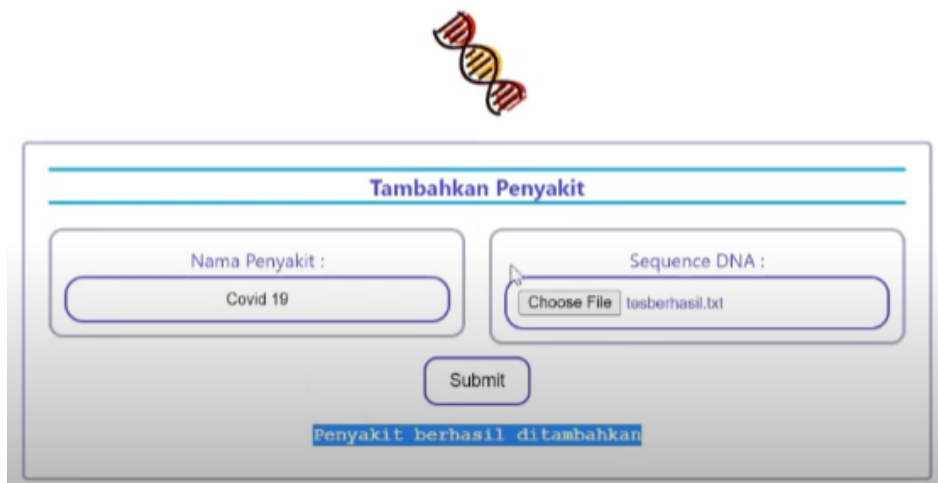
1. Clone repository ini, atau download zip dan extract semua file source code.
2. Pastikan sudah menginstall semua dependency yaitu Go dan Node.js (npm)
3. Masuk ke folder src
4. Nyalakan backend dengan cara:  
go run main.go
5. Nyalakan frontend dengan cara:  
npm install  
Npm start
6. Web dapat dijalankan sesuai fitur-fitur yang tersedia pada <http://localhost:3000>

### b. Interface Setiap Fitur

1. Home Page
2. Fitur Menambahkan Penyakit
3. Fitur Test DNA
4. Fitur Pencarian Hasil

## C. Hasil Pengujian

### a. Input Penyakit Berhasil



### b. Input Penyakit Gagal (DNA Tidak Valid)



**Tambahkan Penyakit**

Nama Penyakit : Covid 19

Sequence DNA : Choose File tesberhasil.txt

Submit

DNA tidak valid

c. Input Penyakit Gagal (Penyakit Sudah Ada)



**Tambahkan Penyakit**

Nama Penyakit : Covid

Sequence DNA : Choose File tesberhasil.txt

Submit

Penyakit sudah ada

d. Test DNA Berhasil (True KMP dan Boyer Moore)



**Welcome to DNA Checker!**

<p>Nama Pengguna :</p> <p>Fikri</p>	<p>Sequence DNA :</p> <p>Choose File tesberhasil.txt</p>	<p>Prediksi Penyakit :</p> <p>Covid 19</p>
-------------------------------------	--	--

KMP ▾

Submit

---

Hasil Tes

29/4/2022 - Fikri - Covid 19 - 100% - True



**Welcome to DNA Checker!**

<p>Nama Pengguna :</p> <p>Fikri</p>	<p>Sequence DNA :</p> <p>Choose File tesberhasil.txt</p>	<p>Prediksi Penyakit :</p> <p>Covid 19</p>
-------------------------------------	--	--

Boyer Moore ▾

Submit

---

Hasil Tes

29/4/2022 - Fikri - Covid 19 - 100% - True

e. Test DNA Berhasil (True tetapi tidak 100% KMP dan Boyer Moore)



**Welcome to DNA Checker!**

<p>Nama Pengguna :</p> <p>Fikri</p>	<p>Sequence DNA :</p> <p><input type="button" value="Choose File"/> tesparsial.txt</p>	<p>Prediksi Penyakit :</p> <p>Covid 19</p>
-------------------------------------	--	--

KMP

---

Hasil Tes

29/4/2022 - Fikri - Covid 19 - 94% - True



**Welcome to DNA Checker!**

<p>Nama Pengguna :</p> <p>Fikri</p>	<p>Sequence DNA :</p> <p><input type="button" value="Choose File"/> tesparsial.txt</p>	<p>Prediksi Penyakit :</p> <p>Covid 19</p>
-------------------------------------	--	--

Boyer Moore

---

Hasil Tes

29/4/2022 - Fikri - Covid 19 - 94% - True

f. Test DNA Berhasil (False KMP dan Boyer Moore)

**Welcome to DNA Checker!**

<p>Nama Pengguna :</p> <p>Fikri</p>	<p>Sequence DNA :</p> <p><input type="button" value="Choose File"/> tesberhasil.txt</p>	<p>Prediksi Penyakit :</p> <p>Pusing</p>
-------------------------------------	---	--

Boyer Moore

---

Hasil Tes

29/4/2022 - Fikri - Pusing - 44% - False



Welcome to DNA Checker!

Nama Pengguna :  
Fikri

Sequence DNA :  
Choose File tesberhasil.txt

Prediksi Penyakit :  
Pusing

KMP

Submit

Hasil Tes

29/4/2022 - Fikri - Pusing - 44% - False

g. Test DNA Gagal (DNA Tidak Valid)

Welcome to DNA Checker!

Nama Pengguna :  
Fikri

Sequence DNA :  
Choose File tesgagal.txt

Prediksi Penyakit :  
Covid 19

Boyer Moore

Submit

Hasil Tes

DNA tidak valid

h. Test DNA Gagal (Penyakit Tidak Ditemukan)

**Welcome to DNA Checker!**

Nama Pengguna :

Fikri

Sequence DNA :

Choose File
tesberhasil.txt

Prediksi Penyakit :

Covid 18

Boyer Moore ▼

Submit

---

**Hasil Tes**

Penyakit tidak ditemukan

i. Pencarian Berhasil (Tanggal Saja)

**Pencarian Hasil**

29 April 2022

*Tanggal dalam format : DD-MM-YYYY, DD/MM/YYYY, DD MM YYYY, atau DD YYYY  
 Input dapat berupa tanggal saja, penyakit saja, atau keduanya*

Carl

---

2022-04-29 Fikri-Covid 19-True-100%

2022-04-29 Fikri-Pusing-False-44%

j. Pencarian Berhasil (Penyakit Saja)



**Pencarian Hasil**

*Tanggal dalam format : DD-MM-YYYY, DD/MM/YYYY, DD MM YYYY, atau DD YYYY  
 Input dapat berupa tanggal saja, penyakit saja, atau keduanya*

---

2022-04-27 Fikri-Pusing-True-100%

2022-04-27 Fikri-Pusing-True-100%

2022-04-27 Fikri-Pusing-True-100%

2022-04-27 Ubay-Pusing-True-100%

k. Pencarian Berhasil (Tanggal dan Penyakit)



**Pencarian Hasil**

*Tanggal dalam format : DD-MM-YYYY, DD/MM/YYYY, DD MM YYYY, atau DD YYYY  
 Input dapat berupa tanggal saja, penyakit saja, atau keduanya*

---

2022-04-29 Fikri-Covid 19-True-100%

l. Pencarian Tidak Ditemukan



Pencarian Hasil

30 April 2022

*Tanggal dalam format : DD-MM-YYYY, DD/MM/YYYY, DD MM YYYY, atau DD YYYY  
Input dapat berupa tanggal saja, penyakit saja, atau keduanya*

Cari

Tidak ada history yang cocok

#### D. Analisis Hasil Pengujian

Dari pengujian, terlihat bahwa regular expression yang digunakan sudah benar karena akan menolak string yang berisi huruf selain ACGT baik saat test DNA maupun input pepnyakit. Algoritma KMP dan Boyer-Moore yang dibuat juga sudah benar, terlihat dari hasil yang didapat selalu sama terlepas dari algoritma apa yang digunakan. Untuk kecepatan algoritma, teorinya KMP akan lebih cepat karena huruf yang dicek cenderung sedikit, hanya 4 karakter. Akan tetapi, tidak ada perbedaan waktu yang signifikan karena test case yang tidak terlalu besar. Regular expression untuk tanggal juga sudah benar karena hasil yang ditampilkan sesuai dengan ekspektasi yang ada pada database.

Untuk sebuah teks dengan panjang  $n$  dan pattern dengan panjang  $m$ . Algoritma KMP memiliki kompleksitas waktu  $O(m+n)$  dimana  $m$  merupakan kompleksitas untuk menghitung border function dan  $n$  kompleksitas untuk pencarian string. Kelebihan KMP adalah algoritma tidak memerlukan mengecek mundur sehingga efektif untuk file yang besar. Kelemahan KMP adalah ketika jumlah karakter besar karena akan sering terjadi mismatch. Algoritma Boyer Moore memiliki kompleksitas  $O(mn + A)$  dimana  $A$  adalah banyaknya karakter yang terdapat pada teks. Kelebihan algoritma Boyer Moore adalah untuk teks yang memiliki banyak karakter dan kelemahannya ketika karakternya sedikit. Oleh karena itu, KMP dan Boyer Moore sama-sama bisa digunakan untuk DNA String Matching. Akan tetapi, KMP lebih efektif walaupun tidak terlalu terlihat karena test case yang digunakan cenderung kecil.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **A. Kesimpulan**

Program DNA Pattern Matching berhasil terealisasi dan terimplementasi dengan menggunakan algoritma String Matching dan Regular Expression. Algoritma Knuth Morris Pratt dan Algoritma Boyer-Moore keduanya dapat digunakan untuk mencari string. Pada himpunan yang jumlah karakternya sedikit, algoritma KMP lebih cepat dari Boyer-Moore tetapi tidak signifikan ketika test case yang diuji kecil. Regular Expression juga dapat digunakan sebagai alat untuk validasi string.

#### **B. Saran**

Penulis menyadari bahwa masih terdapat banyak kekurangan dalam pembuatan laporan ini. Penulis berharap bahwa persoalan dapat ditelusuri lebih lanjut agar dapat ditemukan solusi yang lebih baik dan efisien untuk menyelesaikan permasalahan ini.

## DAFTAR PUSTAKA

- 1) Munir, R. (n.d.). *Pencocokan string string/pattern matching*. Informatika ITB. Retrieved April 24, 2022, from <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>
- 2) Munir, R. (n.d.). *String matching dengan regular expression*. Informatika ITB. Retrieved April 24, 2022, from <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf>
- 3) *JavaScript Language Documentation*. MDN. (n.d.). Retrieved April 25, 2022, from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- 4) *Go Language Documentation*. Go. (n.d.). Retrieved April 25, 2022, from <https://go.dev/doc/>
- 5) *CSS Language Documentation*. DevDocs. (n.d.). Retrieved April 25, 2022, from <https://devdocs.io/css/>

## Link Penting

[Link](#) Repository Gabungan

[Link](#) Repository Frontend

[Link](#) Repository Backend

[Link](#) Video