**EKSTRAKSI INFORMASI DARI ARTIKEL BERITA DENGAN ALGORITMA PENCOCOKAN STRING**

**LAPORAN TUGAS KECIL**

Diajukan Untuk Memenuhi Tugas IF2211 Strategi Algoritma



Disusun oleh

**MICHAEL HANS**

**13518056**

**TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2020**

**DAFTAR ISI**

DAFTAR ISI 1

BAB I ALGORITMA 2

1.1 Algoritma Boyers-Moore 2

1.2 Algoritma Knuth-Morris-Pratt 4

1.3 Regular Expression 5

BAB II PERSOALAN EKSTRAKSI INFORMASI 7

2.1 Deskripsi Persoalan 7

2.2 Ide Penyelesaian Masalah 7

2.3 Regular Expression 8

2.3 Algoritma Pengekstrakan Data 8

BAB III PENGUJIAN PROGRAM 11

3.1 Spesifikasi Komputer 11

3.2 Pengujian Program 11

3.3 Checklist Program 16

DAFTAR PUSTAKA 17

**BAB I**

**ALGORITMA**

**1.1 Algoritma Boyers-Moore**

Algoritma Boyer-Moore (BM) merupakan salah satu algoritma dalam *pattern matching* yang pendekatannya berbeda dengan brute-force karena terdapat teknik-teknik tertentu dalam alur kerja algoritmanya. Berikut ini adalah mekanisme umum algoritma BM.

1. **The *looking-glass technique***

Memeriksa kecocokan pattern P dengan teks T, dimulai dari indeks terakhir pada P

Pemeriksaan terhadap T tetap dimulai dari awal, dalam hal ini indeks I akan dimulai pada nilai m-1 bila panjang P adalah m.

1. **The *character-jump* *technique***

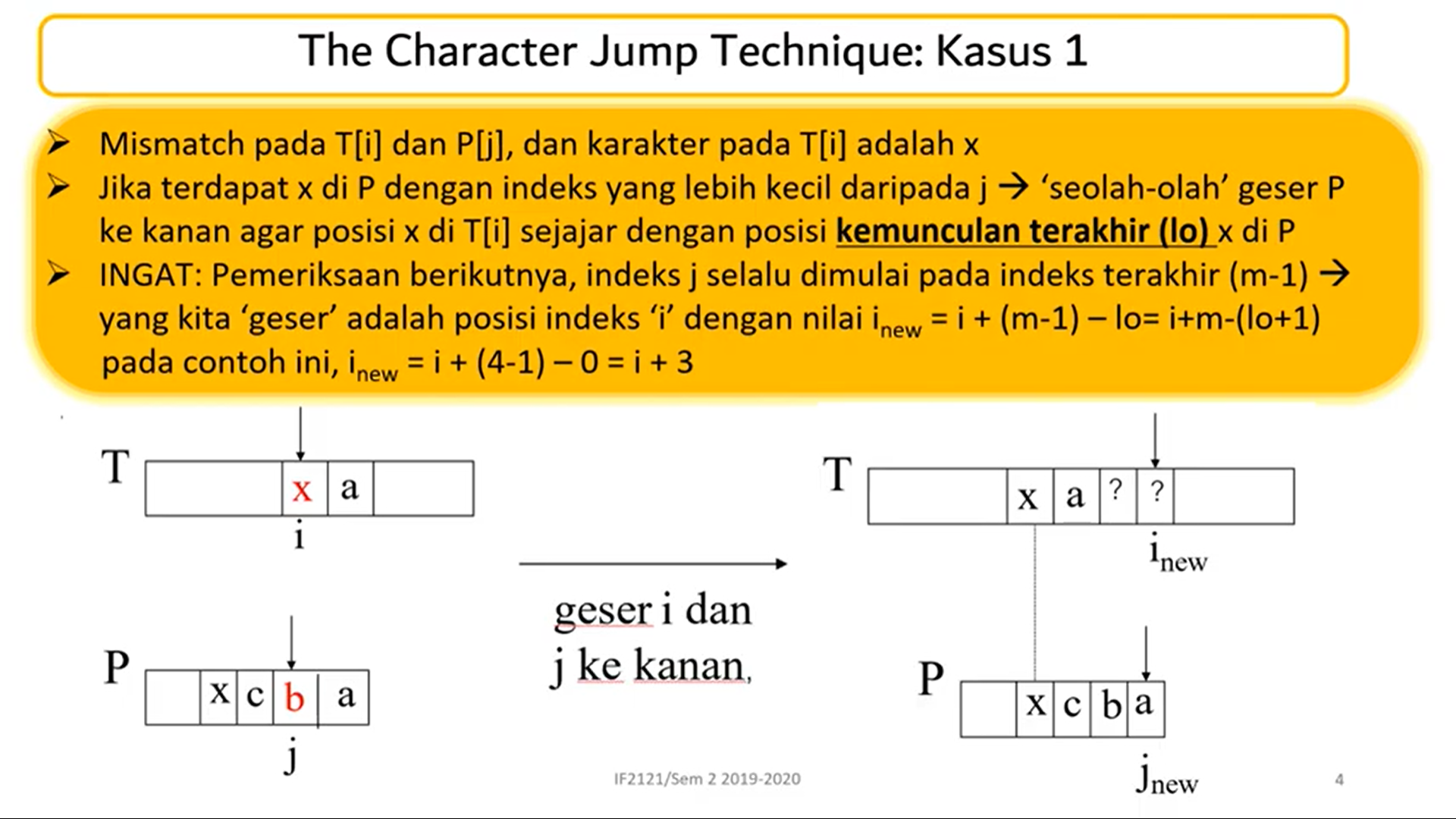
Dilakukan ketika terjadi mismatch (T[i] ≠ P[j]) dengan T[i] = x

Terdapat 3 kasus yang mungkin terjadi, namun pemeriksaan dilakukan secara satu per satu dengan urutan pemeriksaan sebagai berikut.

Mismatch pada T[i] dan P[j], dan karakter pada T[i] adalah x.

**a. Kasus 1:**

Jika terdapat x di P dengan indeks yang lebih kecil daripada j, maka ‘seolah-olah’ geser P ke kanan agar posisi x di T[i] sejajar dengan posisi **kemunculan terakhir (lo)** x di P.

Pemeriksaan berikutnya, indeks j selalu dimulai pada indeks terakhir (m-1) sehingga yang perlu digeser adalah posisi indeks i dengan nilai inew = i + (m-1) – lo

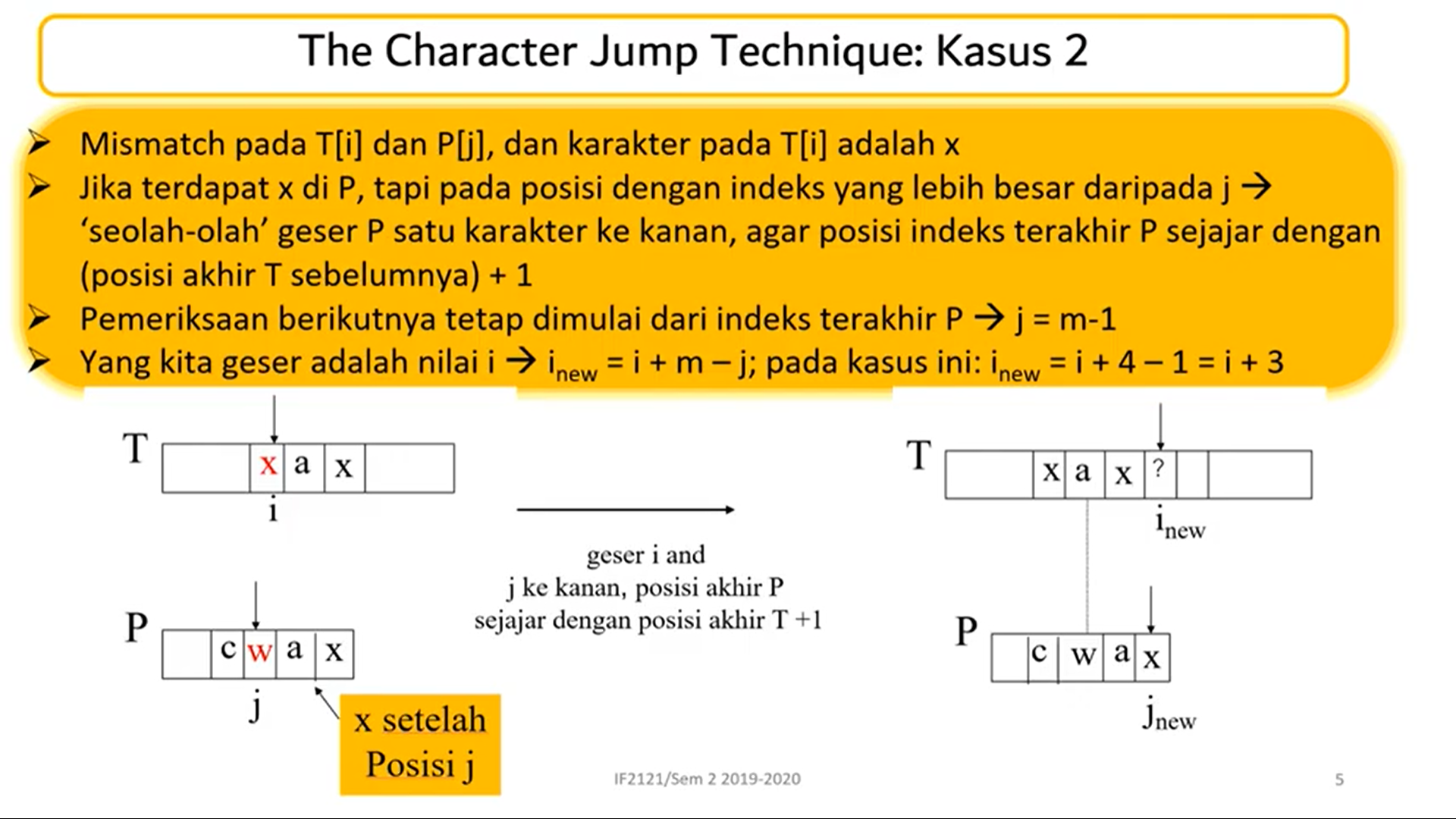
*Ilustrasi Kasus 1: Kemunculan terakhir x lebih kecil daripada j*

**b. Kasus 2:**

Mismatch pada T[i] dan P[j], dan karakter pada T[i] adalah x.

Jika terdapat x di P, tapi pada posisi dengan indeks yang lebih besar daripada j, maka ‘seolah-olah’ geser P satu karakter ke kanan, agar posisi indeks terakhir P sejajar dengan posisi akhir T sebelumnya + 1.

Pemeriksaan berikutnya tetap dimulai dari indeks terakhir P, yaitu j = m-1

Yang digeser adalah nilai i, yaitu inew = i + m – j

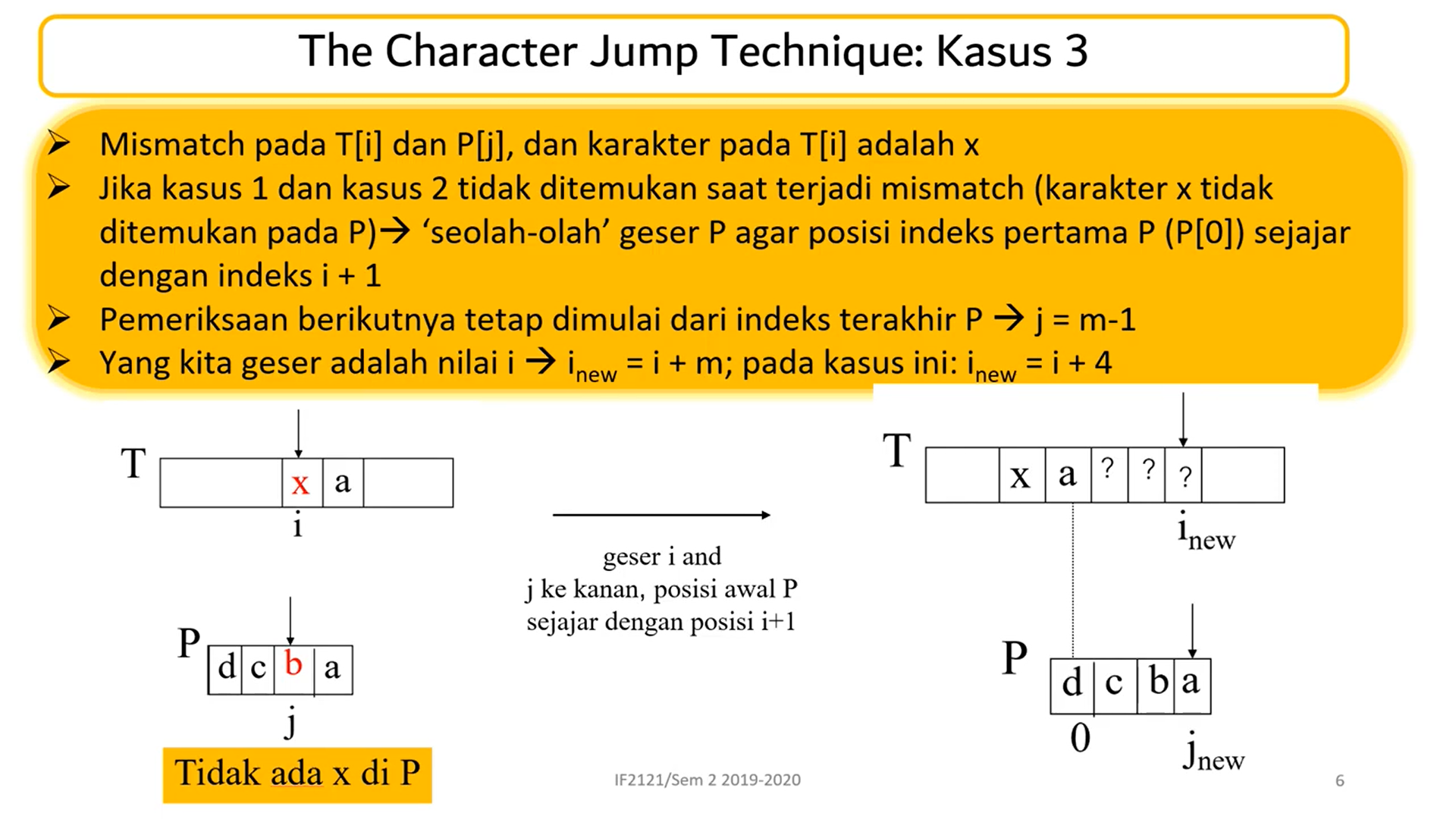
*Ilustrasi Kasus 2: Kemunculan terakhir x lebih besar daripada j*

**c. Kasus 3:**

Mismatch pada T[i] dan P[j], dan karakter pada T[i] adalah x.

Jika kasus 1 dan kasus 2 tidak ditemukan saat terjadi mismatch (karakter x tidak ditemukan pada P), maka ‘seolah-olah’ geser P agar posisi indeks pertama P (P[0]) sejajar dengan indeks i + 1.

Pemeriksaan berikutnya tetap dimulai dari indeks terakhir P, yaitu j = m-1

Yang digeser adalah nilai i, yaitu inew = i + m

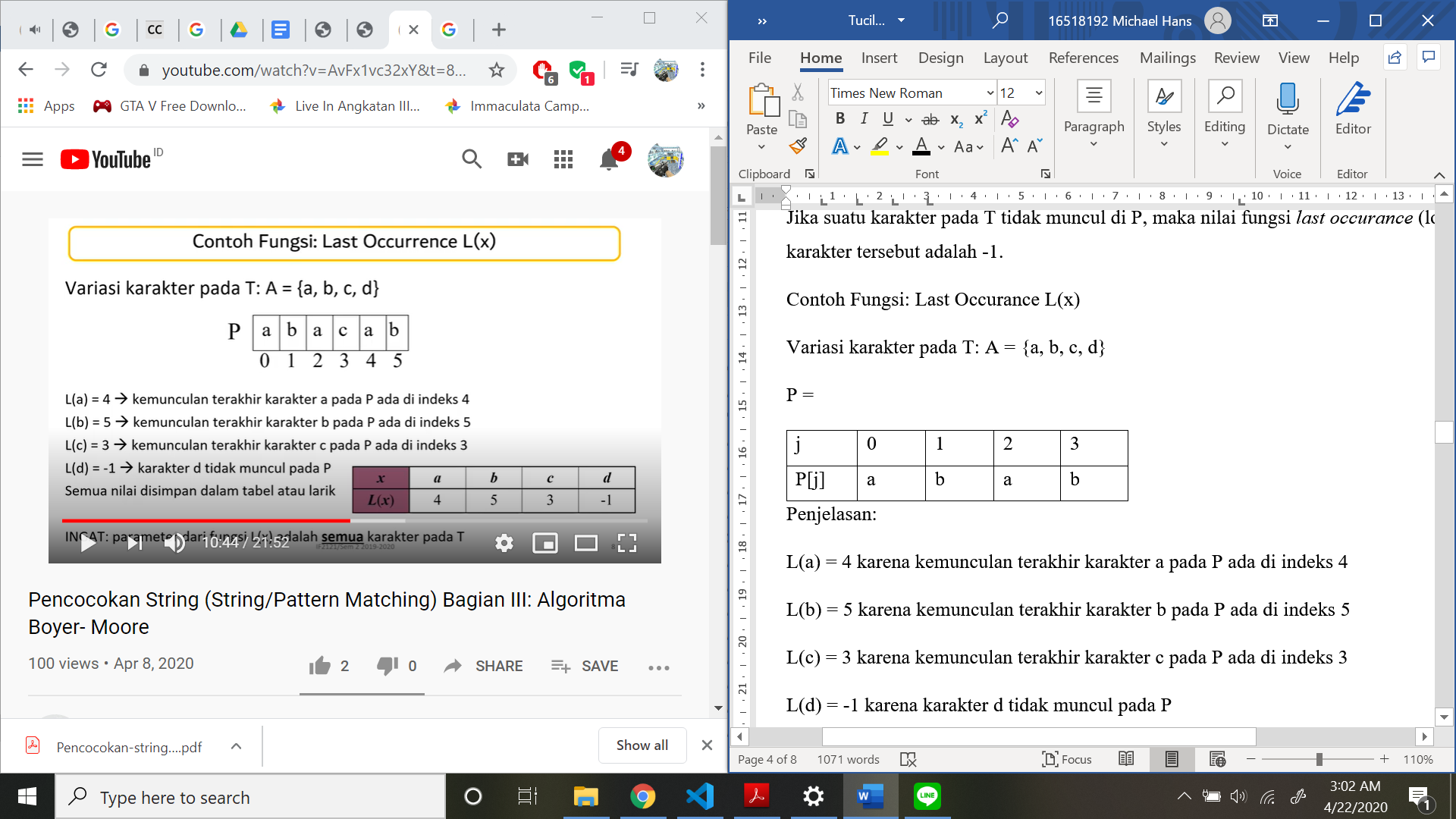
*Ilustrasi Kasus 3: Karakter x tidak muncul dalam pattern P*

**Last Occurance Function**

Untuk menangani kasus 1 dan 2, diperlukan informasi mengenai posisi kemunculan terakhir suatu karakter pada P. Oleh karena itu, akan dilakukan preproses fungsi sebelum melakukan matching P dengan T, yaitu menentukan posisi kemunculan terakhir untuk semua karakter unik pada teks T di dalam pattern P.

Jika suatu karakter pada T tidak muncul di P, maka nilai fungsi *last occurance* (lo) untuk karakter tersebut adalah -1.

Contoh Fungsi: Last Occurance L(x)

Variasi karakter pada T: A = {a, b, c, d}

Penjelasan:

L(a) = 4 karena kemunculan terakhir karakter a pada P ada di indeks 4

L(b) = 5 karena kemunculan terakhir karakter b pada P ada di indeks 5

L(c) = 3 karena kemunculan terakhir karakter c pada P ada di indeks 3

L(d) = -1 karena karakter d tidak muncul pada P

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 |
| P[j] | a | b | a | b |

Semua nilai disimpan dalam tabel atau larik.

**1.2 Algoritma Knutt-Morris-Pratt**

Algoritma Knutt-Morris-Pratt (KMP) merupakan salah satu algoritma dalam *pattern matching*. Algoritma KMP melakukan pemeriksaan dari kiri ke kanan, seperti pendekatan brute force. Namun, perbedaan dengan algoritma *brute-force* adalah pergeseran yang dilakukan lebih cerdas pada algoritma KMP.

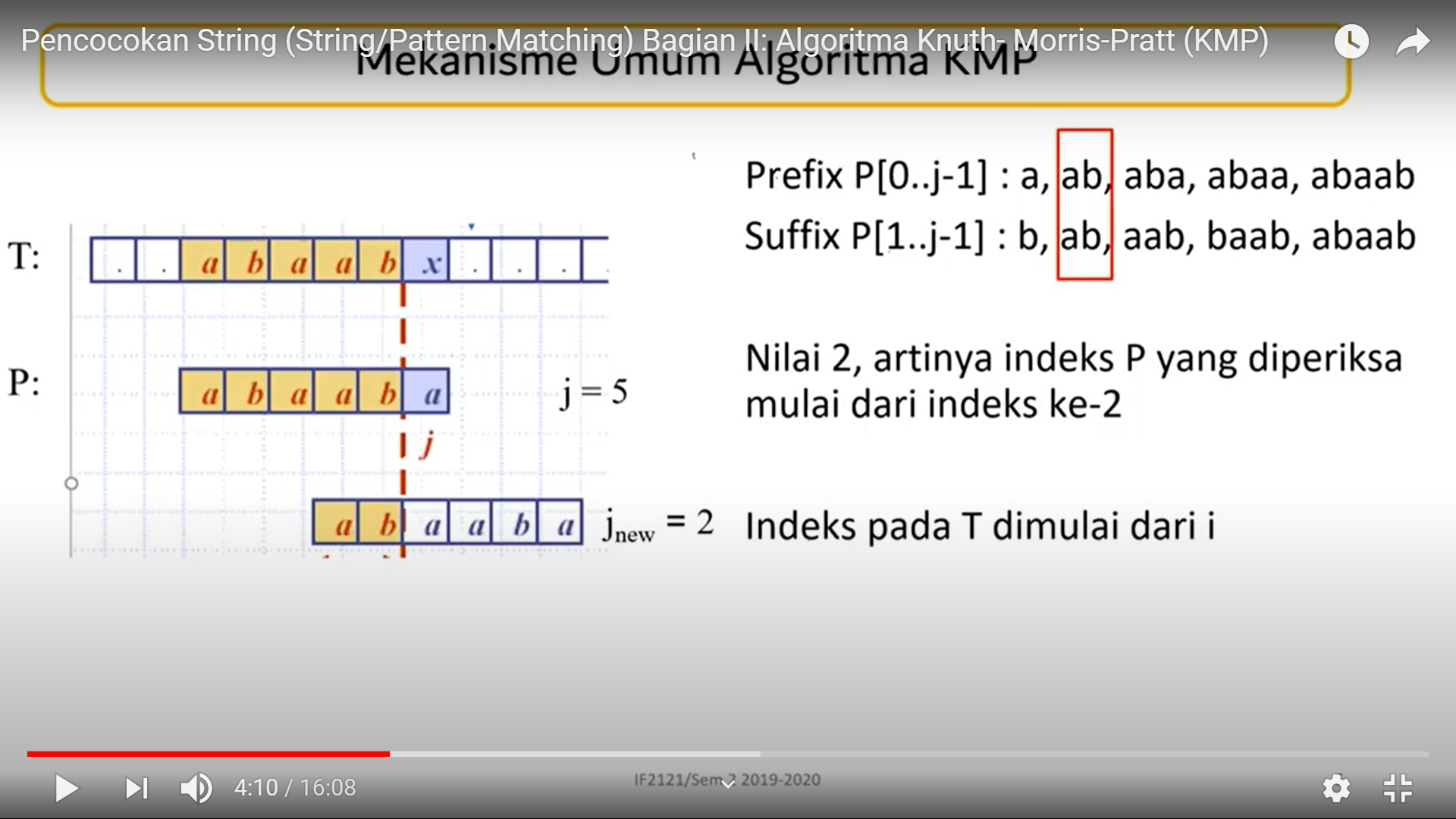
Pergeseran algoritma KMP lebih cerdas karena algoritma ini memanfaatkan konsep prefix dan suffix. Konsep prefix dan suffix yang diterapkan adalah sebagai berikut.

Prefix (awalan) adalah substring S [0..k]

Suffix (akhiran) adalah substing S [k..m-1]

Dengan m adalah panjang dari string pattern dan k adalah suatu index antara 0 hingga m-1

Pergeseran algoritma KMP lebih cerdas dengan mempertimbangkan sebanyak mungkin pergeseran sehingga tidak perlu mengulang pemeriksaan yang sama.

**Mekanisme Umum**

*Ilustrasi Mekanisme Umum Algoritma KMP*

Jika terjadi mismatch teks T pada T[i] dan pattern P pada P[j], yaitu T[i] ≠ P[j], pergeseran yang dilakukan adalah sebanyak:

* Nilai prefix P[0..j-1] terbesar yang juga merupakan suffix P[1..j-1]
* Prefix P[0..j-1] : a, ab, aba, abaa, abaab
* Suffix P[1..j-1] : b, ab, aab, baab

Nilai prefix terbesar adalah 2, artinya indeks P yang diperiksa selanjutnya mulai dari indeks ke-2.

**KMP Border Function**

Karena nilai prefix terbesar dipengaruhi oleh posisi j yang merupakan posisi terjadi mismatch, maka diperlukan suatu preposes terhadap pattern P untuk menyimpan semua nilai prefix terbesar P[0..j-1] terbesar yang merupakan suffix P[1..j-1] ketika terjadi mismatch pada P[j].

Preproses ini disebut sebagai KMP Border Function, yaitu sebuah fungsi pembatas untuk melakukan pergeseran berdasarkan kesamaan prefix dan suffix terbesar. KMP Border Function ini akan disimpan ke dalam sebuah senarai untuk memudahkan pengambilan nilai prefix terbesar yang sudah disimpan pada awal proses.

Berikut ini adalah contoh pemrosesan KMP Border Function.

Pattern P = ababababca

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| P[j] | a | b | a | b | a | b | a | b | c | a |
| k | - | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| b[k] | - | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 0 |

Penjelasan: Bila k = 6, maka

Prefix P[0..6] : a, ab, aba, abab, ababa, ababab, abababa

Suffix P[1..6] : a, ba, aba, baba, ababa, bababa

Perhatikan bahwa terdapat kesamaan prefix dan suffix pada a, aba, dan ababa. Karena substring ababa merupakan irisan prefix suffix terpanjang, nilai prefix terbesar adalah 5.

**1.3 Regular Expression**

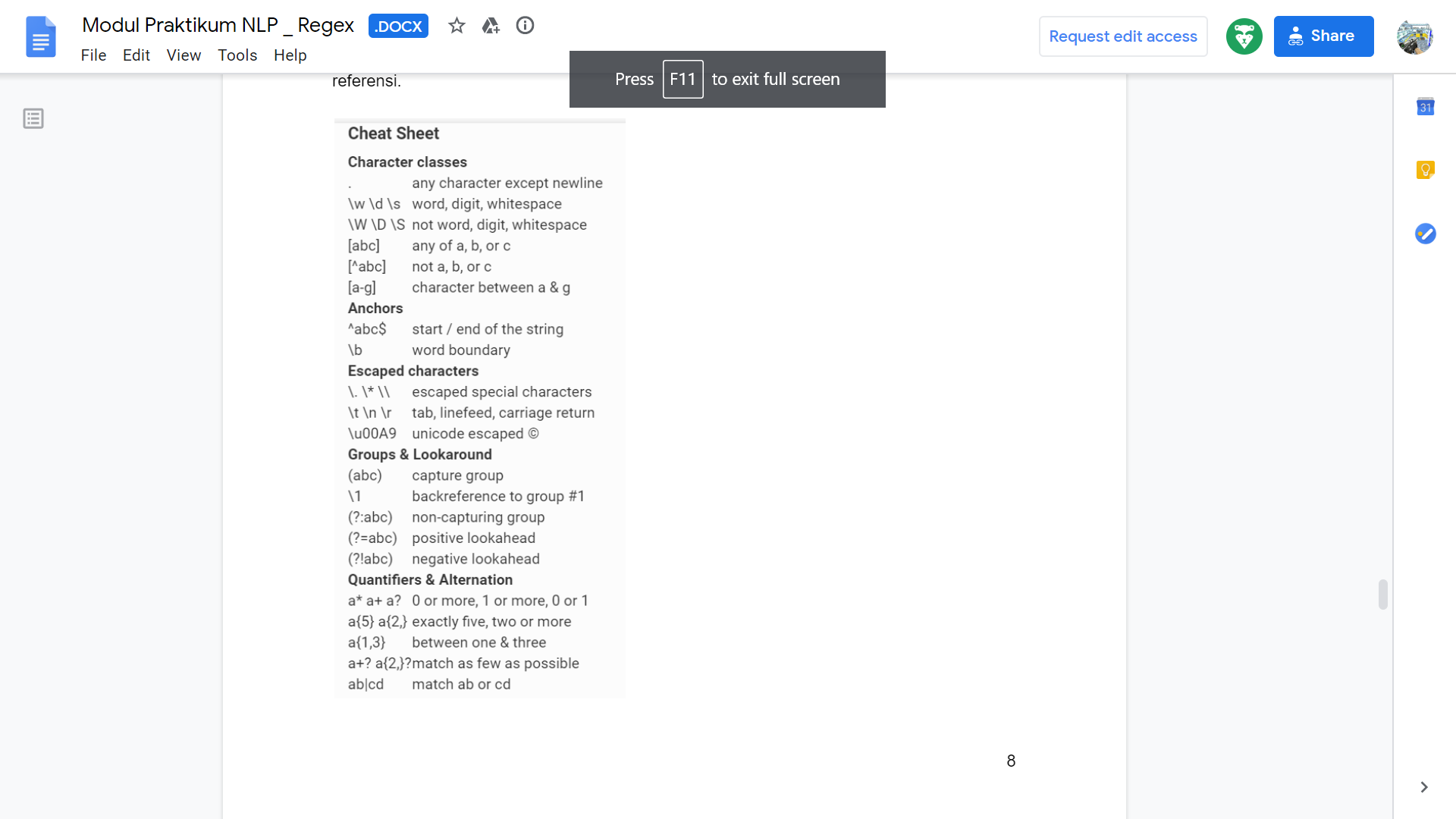
Regular expression (regex) adalah notasi standar yang mendeskripsikan suatu pola (pattern) berupa urutan karakter atau string. Regex digunakan untuk pencocokan string (string matching) dengan efisien. Regex ini sangat bermanfaat, terutama dalam membangkitkan dan mendefinisikan suatu bahasa yang dinyatakan dalam ekspresi tertentu. Regular expression mencocokkan string dengan suatu pola, dengan memanfaatkan state-state seperti pada mesin turing.

Pencocokan dibagi jadi 2, karakter literal dan karakter meta.

1. Karakter Literal, yaitu melakukan pencocokan seperti pada umumnya
2. Metakarakter, yaitu karakter khusus yang mencakup suatu kelompok karakter atau karakter tertentu yang tidak dapat dituliskan secara eksplisit.

Regular Expression ini dibuat sesuai dengan preferensi bahasa yang ingin diterima oleh pembuat regex. Terdapat banyak kamus data yang bisa digunakan dalam penulisan regex, seperti bila ingin mencocokkan suatu nomor yang diawali dengan angka 11, maka regex yang sesuai adalah menambahkan karakter literal berupa ‘11’ diawal ekspresi, yang kemudian diikuti oleh bilangan integer berapapun.

Berikut ini adalah beberapa cheatsheet terkait penggunaan regex tersebut.



**BAB II**

**PERSOALAN EKSTRAKSI INFORMASI**

**2.1 Deskripsi Persoalan**

Algoritma *Pattern Matching* dan Regular Expression sangat bermanfaat dalam konteks kehidupan sehari-hari, terutama pada pemrosesan kata. Salah satu penerapan algoritma *Pattern Matching* dan Regular Expression adalah melakukan ekstraksi data dari suatu artikel dengan *keyword* tertentu untuk mengambil dua buah data penting, seperti jumlah dan tanggal.

Data menjadi sesuatu yang penting, terutama terkait data virus corona yang melanda saat ini sehingga dengan adanya aplikasi ekstraksi data, pengguna dapat memperoleh data-data secara mudah dari banyak sumber secara bersamaan tanpa perlu membaca secara penuh. Oleh karena itu, dibuatlah suatu ekstraktor data yang menerapkan algoritma *Pattern Matching* dan Regular Expression tersebut.

**2.2 Ide Penyelesaian Masalah**

Persiapan aplikasi ekstraksi data dimulai dengan menentukan parameter-parameter yang dibutuhkan untk melakukan ekstraksi data. Berikut ini adalah paramater-parameter yang dibutuhkan sebelum proses ektraksi data dilakukan.

1. Artikel-artikel yang ingin dibaca dalam bentuk .txt
2. Keyword yang ingin ditelusuri dalam setiap artikel
3. Algoritma pencocokkan yang diterapkan (BM, KMP, dan Regex)

Parameter-parameter tersebut akan diisi oleh pengguna yang kemudian akan diproses oleh algoritma *pattern matching* dan regular expression tersebut. Alur penyelesaian persoalan adalah sebagai berikut.

1. Untuk setiap artikel yang mau diekstrak, dilakukan pemecahan dari sebuah teks besar ke dalam kumpulan kalimat. Pemecahan menjadi kalimat-kalimat tersebut menggunakan library nltk yang telah disediakan python.
2. Dari kumpulan kalimat tersebut, akan dilakukan pemfilteran untuk memperoleh kalimat-kalimat yang mengandung *keyword* yang dimaksud. Pemfilteran ini memanfaatkan salah satu algoritma pencocokkan sesuai preferensi pengguna.
3. Untuk setiap kalimat dalam kumpulan kalimat yang sudah difilter, akan dilakukan proses ekstraksi data menggunakan regular expression yang sudah didefinisikan oleh pengembang. Oleh karena itu, pengembang perlu mendefinisikan terlebih dahulu regular expression agar bisa menerima grammar berupa tanggal dan jumlah yang diperlukan untuk ekstraksi data.
4. Ekstraksi data dilakukan dengan mencari kecocokkan antara substring dalam suatu kalimat dengan regular expression yang telah didefinisikan. Tujuan menggunakan regex karena tanggal dan jumlah mempunyai variasi penulisan yang berbeda-beda sehingga regex mampu mencocokkan dengan penulisan tersebut sesuai notasi yang didefinisikan.
5. Kembalikan hasil ekstraksi data dan kalimat yang diekstrak tersebut ke layar sehingga pengguna mampu melihat hasil ekstraksi data dari artikel.

**2.3. Regular Expression**

Berikut ini adalah regular expression yang penulis definisikan dan implementasikan dalam aplikasi pengekstrak data ini.

import re

# Kamus Regular Expression

countAble= '(?:kasus|orang|manusia|korban|jiwa|pasien)'

numberFirstFormat = "(?:(?:\d{1,3}\.?)\*(?:\d+) %s)" % (countAble)

numberEnum = '(?:satu|dua|tiga|empat|lima|enam|tujuh|delapan|sembilan|sepuluh|puluh|ribu|juta)'

numberSecondFormat = "(?:(?:%s )+%s %s)" % (numberEnum, numberEnum, countAble)

genericNumber = re.compile("(?:%s|%s)" % (numberFirstFormat, numberSecondFormat))

# Construct Format for Date

datenum = '(?:\d{1,2})'

tahun = '(?:\d{4})'

bulan = '(?:januari|februari|maret|april|mei|juni|juli|agustus|september|oktober|november|desember)'

bulanSingkat = '(?:jan|feb|mar|apr|mei|jun|jul|ags|sep|okt|nov|des)'

day = '(?:senin|selasa|rabu|kamis|jumat|sabtu|minggu)'

dateFirstFormat = "(?:%s, %s (?:%s|%s) %s)" % (day, datenum, bulan, bulanSingkat, tahun)

dateSecondFormat = "(?:\d{1,2}/\d{1,2}/\d{4})"

dateThirdFormat = '(?:kemarin|besok|hari ini|lusa)'

dateOtherFormat = "(?:%s|%s|%s|%s|%s)" % (day, datenum, bulan, bulanSingkat, tahun)

# Construct Format for Time

timeFirstFormat = '(?:\d{2}:\d{2} wib)'

timeSecondFormat = '(?:pukul \d{2}\.\d{2} wib)'

timeThirdFormat = '(?:pagi|siang|sore|malam)'

# Combination of Date and Time

dateTimeFull = "(?:%s,? (?:%s|%s))" % (dateFirstFormat, timeFirstFormat, timeSecondFormat)

# Combine all things into one regular expression

genericDate = "(?:%s|%s|%s)" % (dateFirstFormat, dateSecondFormat, dateThirdFormat)

genericTime = "(?:%s|%s|%s)" % (timeFirstFormat, timeSecondFormat, timeThirdFormat)

genericDateTime = re.compile("(?:%s|%s|%s)" % (dateTimeFull,genericDate, genericTime))

**2.4. Algoritma Pengekstrakan Data**

Berikut ini adalah potongan kode program terkait pengekstrakan data dari berbagai artikel dengan keyword dan metode algoritma pencocokkan yang digunakan.

# Memulai Proses Ekstraksi Data dari beberapa Input Files

def BeginExtraction(keyword, method, files):

    listOfResult = []         # menyimpan daftar teks dari seluruh teks yang dipilih

    # Struktur listOfResult: (namafile, teks, extractedSentence, jumlahSentence)

    # Struktur extractedSentence : (sentencePattern, jumlah, tanggal)

    # Algoritma Utama

    for i in range(len(files)):

        extractedSentence = []  # menyimpan daftar kalimat yang sudah terekstraksi

        text = []               # menyimpan string dari sebuah text yang dibaca

        sentence = []           # menyimpan daftar kalimat yang telah diparsing

        sentencePattern = []    # menyimpan daftar kalimat yang mempunyai pattern

        text = ReadText(files[i])

        text = text.lower()

        articleDate = ArticleDate(text)

        sentence = sent\_tokenize(text)

        # Pattern Matching

        if (method == "Algoritma Boyers-Moore"):

            sentencePattern = GetPatternBMSentence(sentence, keyword)

        elif (method == "Algoritma Knuth-Morris-Pratt"):

            sentencePattern = GetPatternKMPSentence(sentence, keyword)

        elif (method == "Regular Expression"):

            sentencePattern = GetPatternRegex(sentence, keyword)

        # Jika tidak ditemukan kalimat yang mengandung keyword

        if (len(sentencePattern) == 0):

            print("Pencarian tidak ditemukan")

            listOfResult.append((files[i], text, sentencePattern, 0))

        # Jika ditemukan kalimat yang mengandung keyword

        else:

            # Extraction Data

            for j in range(len(sentencePattern)):

                jumlah = FindNumber(sentencePattern[j])

                tanggal = FindDate(sentencePattern[j])

if not(jumlah == "Tidak diketahui"):

                    sentencePattern[j] = BoldMatches(sentencePattern[j],jumlah)

                if not(tanggal == "Tidak diketahui"):

                    sentencePattern[j] = BoldMatches(sentencePattern[j],jumlah)

                else:

                    tanggal = articleDate

sentencePattern[j] = BoldMatches(sentencePattern[j], keyword)

                extractedSentence.append((sentencePattern[j].capitalize(), jumlah, tanggal.capitalize()))

            listOfResult.append((files[i], text, extractedSentence, len(sentencePattern)))

    return listOfResult

**BAB III**

**PENGUJIAN PROGRAM**

**3.1. Spesifikasi Komputer**

Berikut ini adalah spesifikasi dari komputer yang digunakan dalam mengimplementasikan dan mengeksekusi program polinom tersebut.

Operating System : Windows 10 Home Single Language 64-bit (10.0, Build 18362)

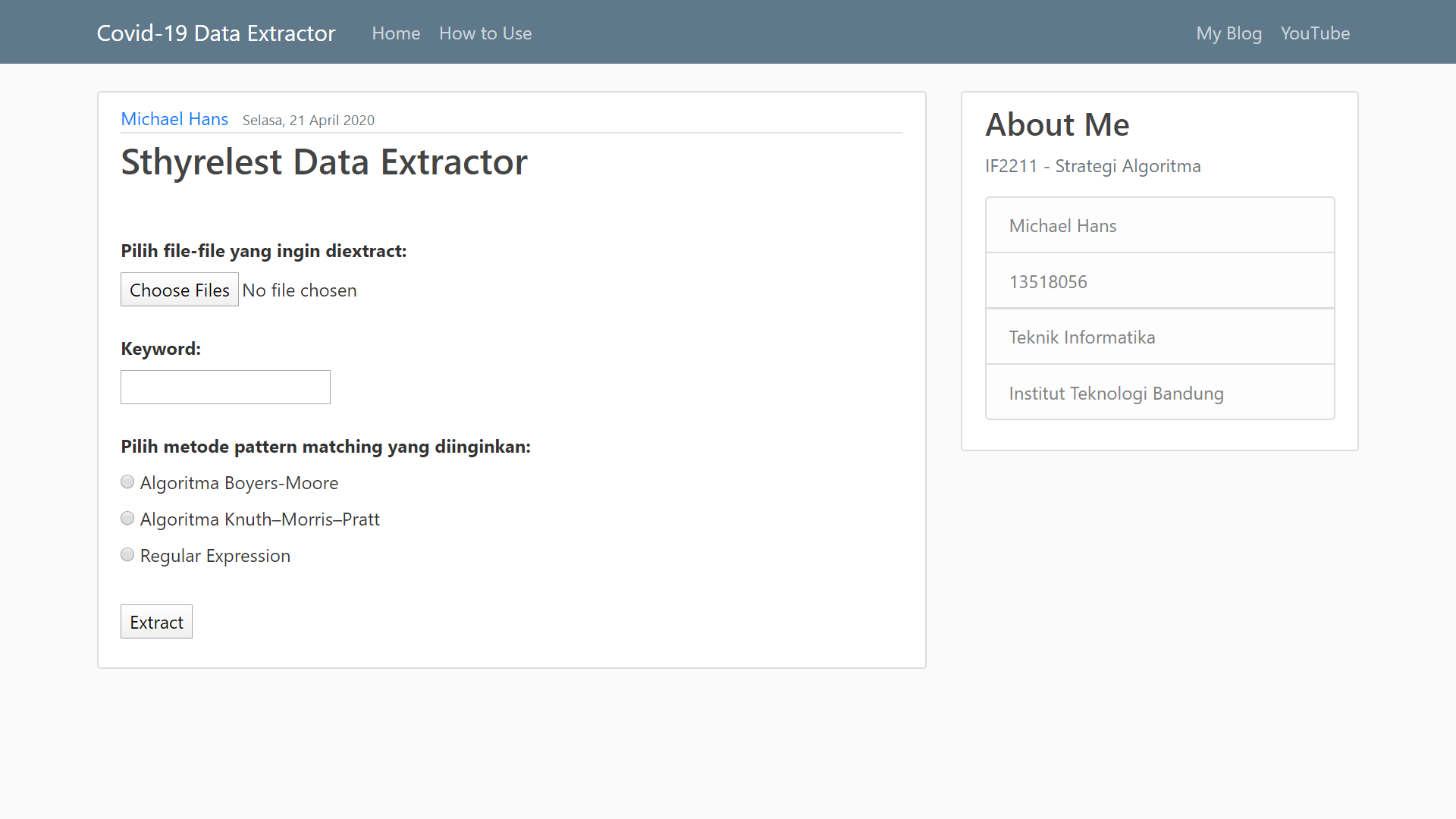
CPU / Processor : Intel® Core™ i7-8550U CPU @ 1.80 GHz (8 CPUs), ~2.0 GHz

RAM : 8192 MB

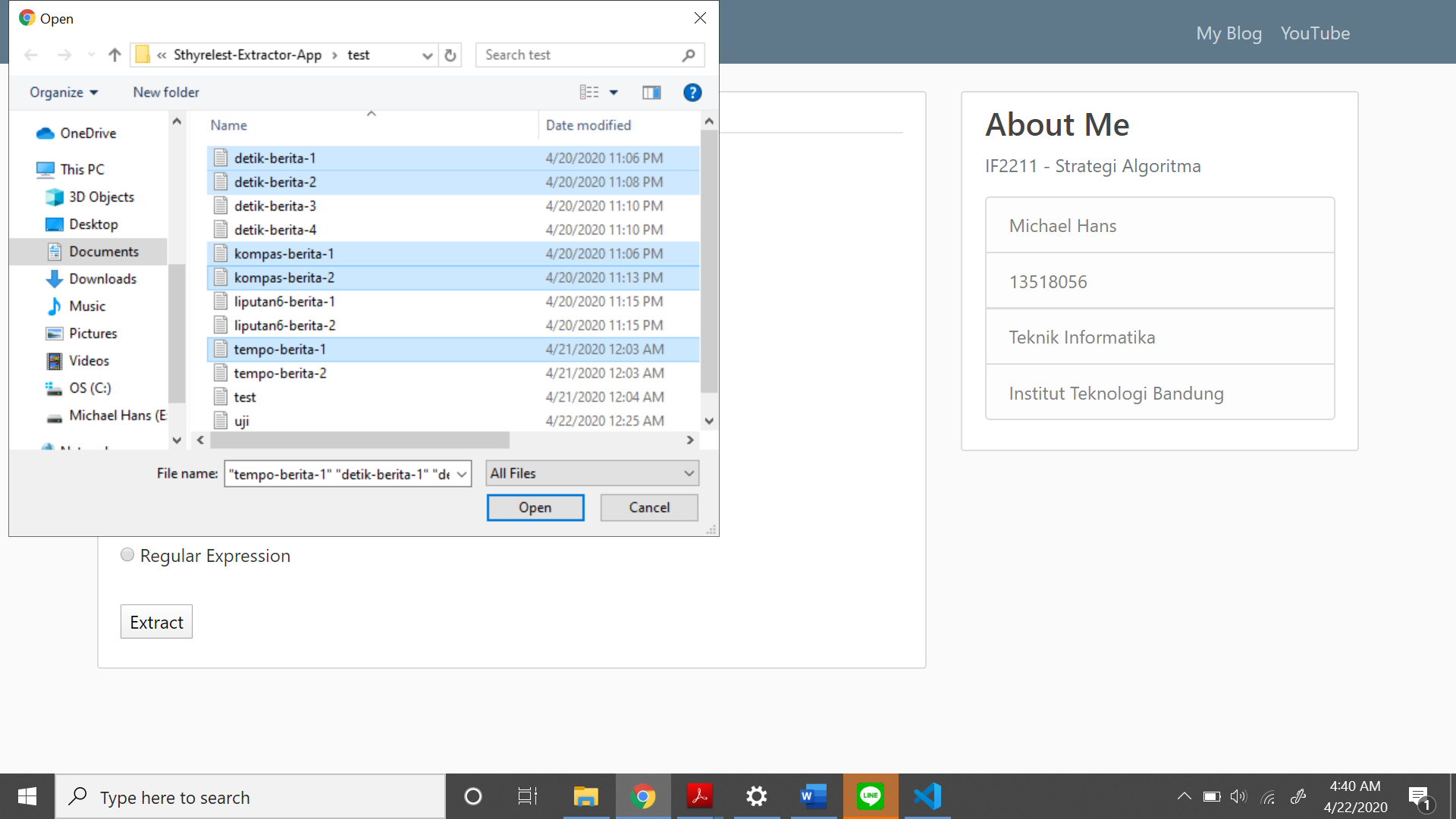
VGA : NVIDIA GeForce MX130 2 GB

**3.2. Pengujian Program**

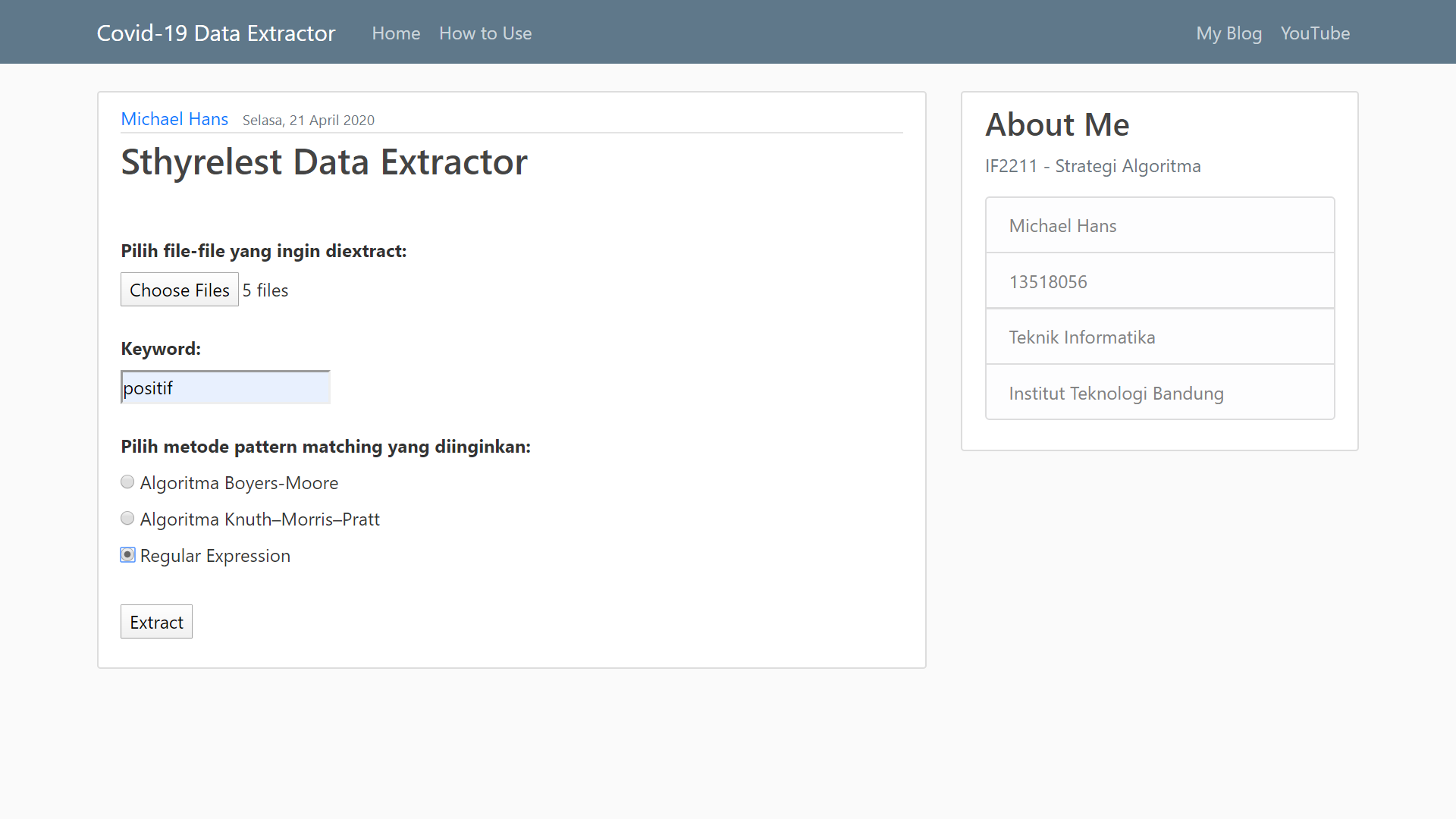
Berikut ini adalah beberapa screenshot yang dilakukan terkait penggunaan aplikasi extractor yang telah diimplementasikan.



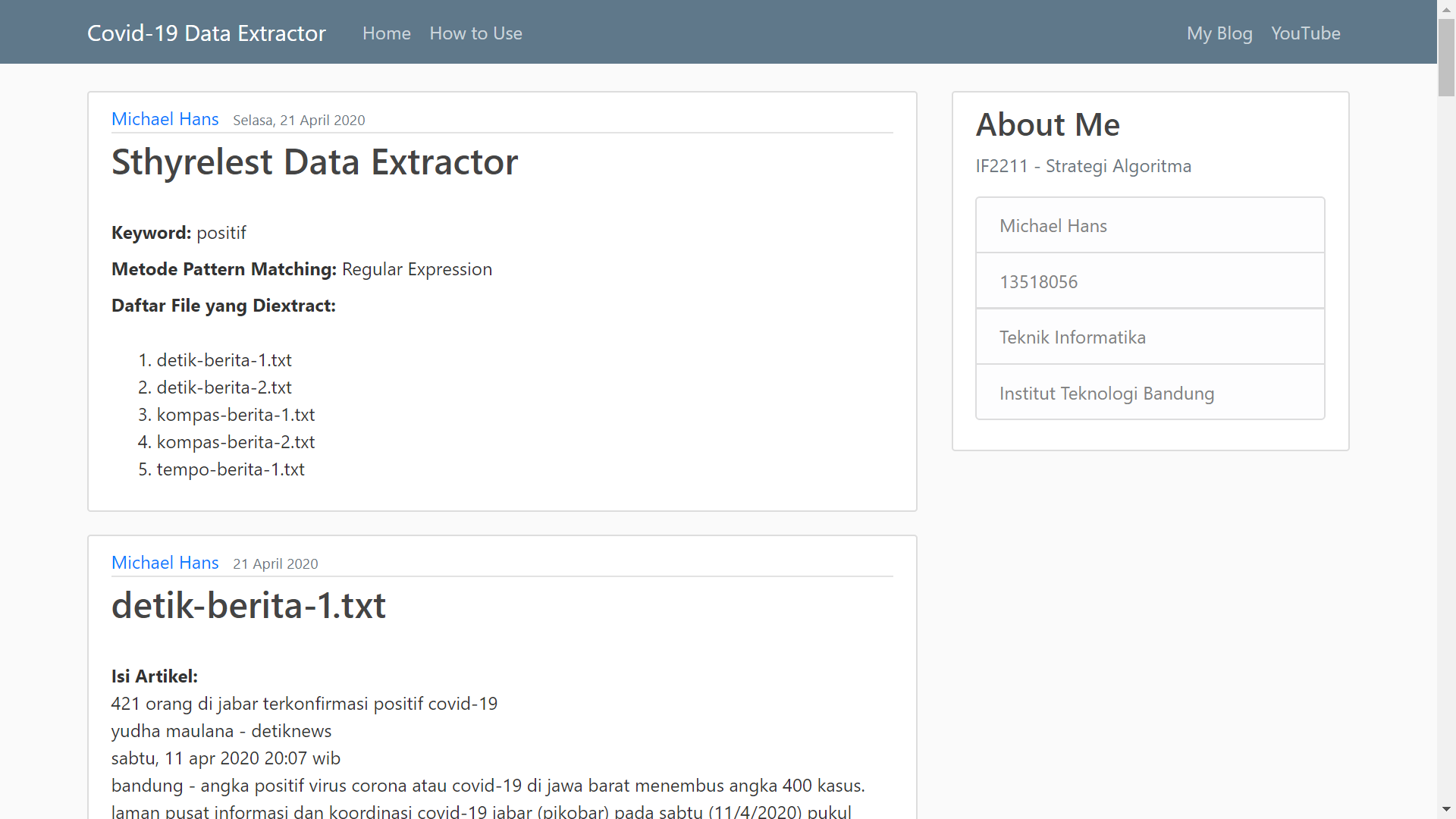
*Tampilan Awal Extractor Application*



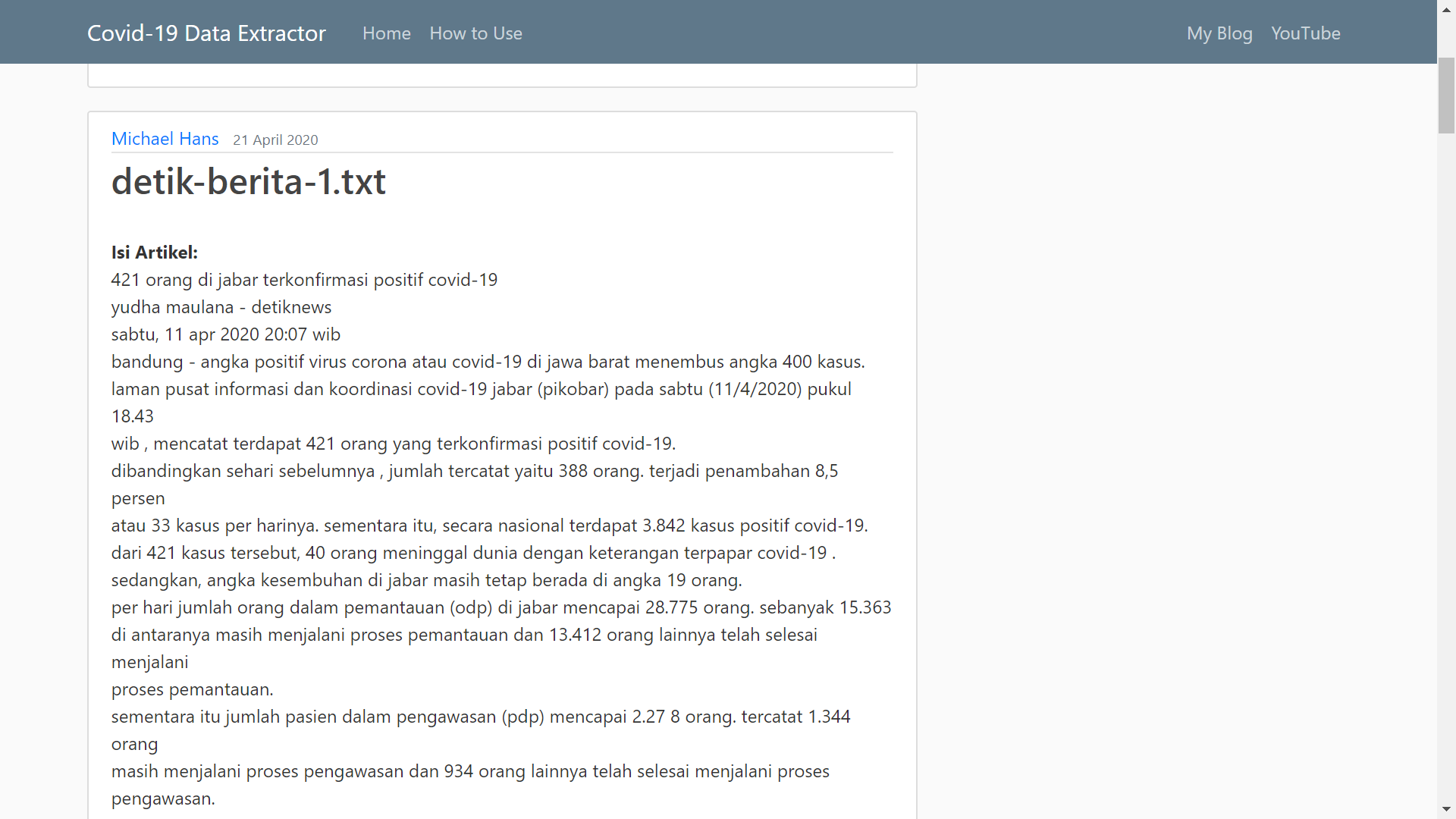
*Melakukan browse file terhadap artikel-artikel yang ingin diekstraksi datanya*



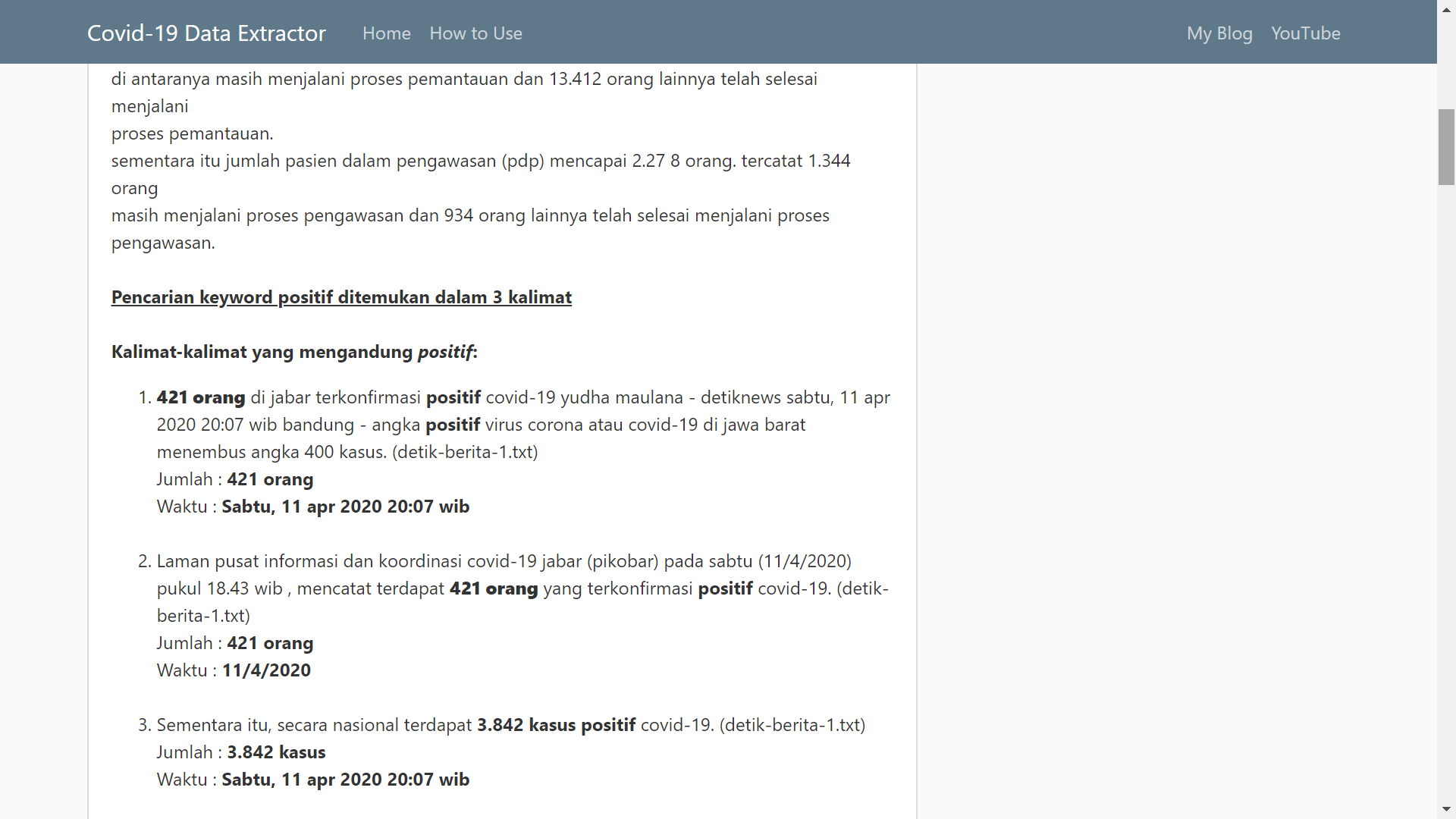
*Tampilan sesudah tiga buah parameter diisi oleh pengguna*



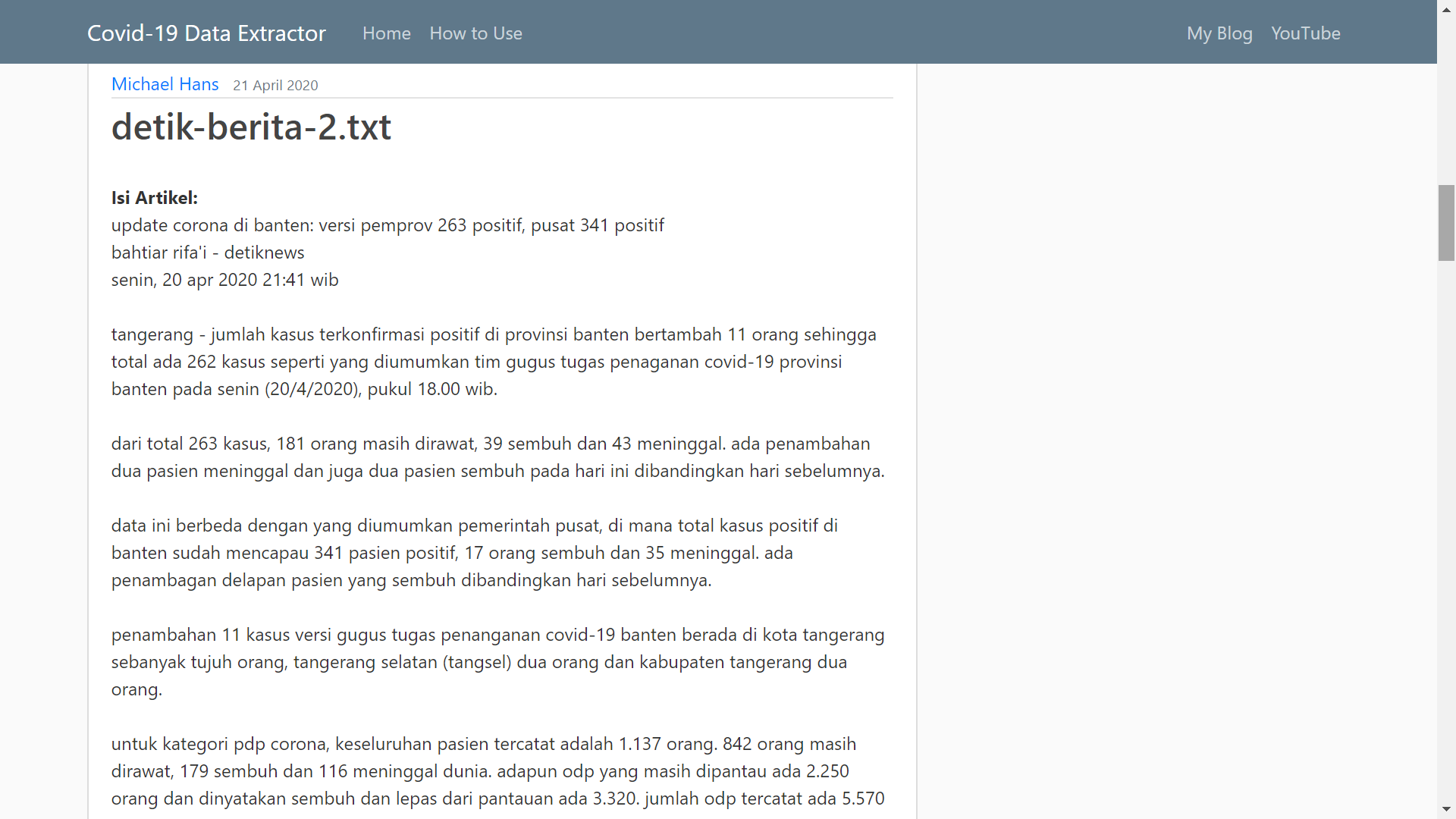
*Tampilan Awal Setelah Ekstraksi Data Berhasil*



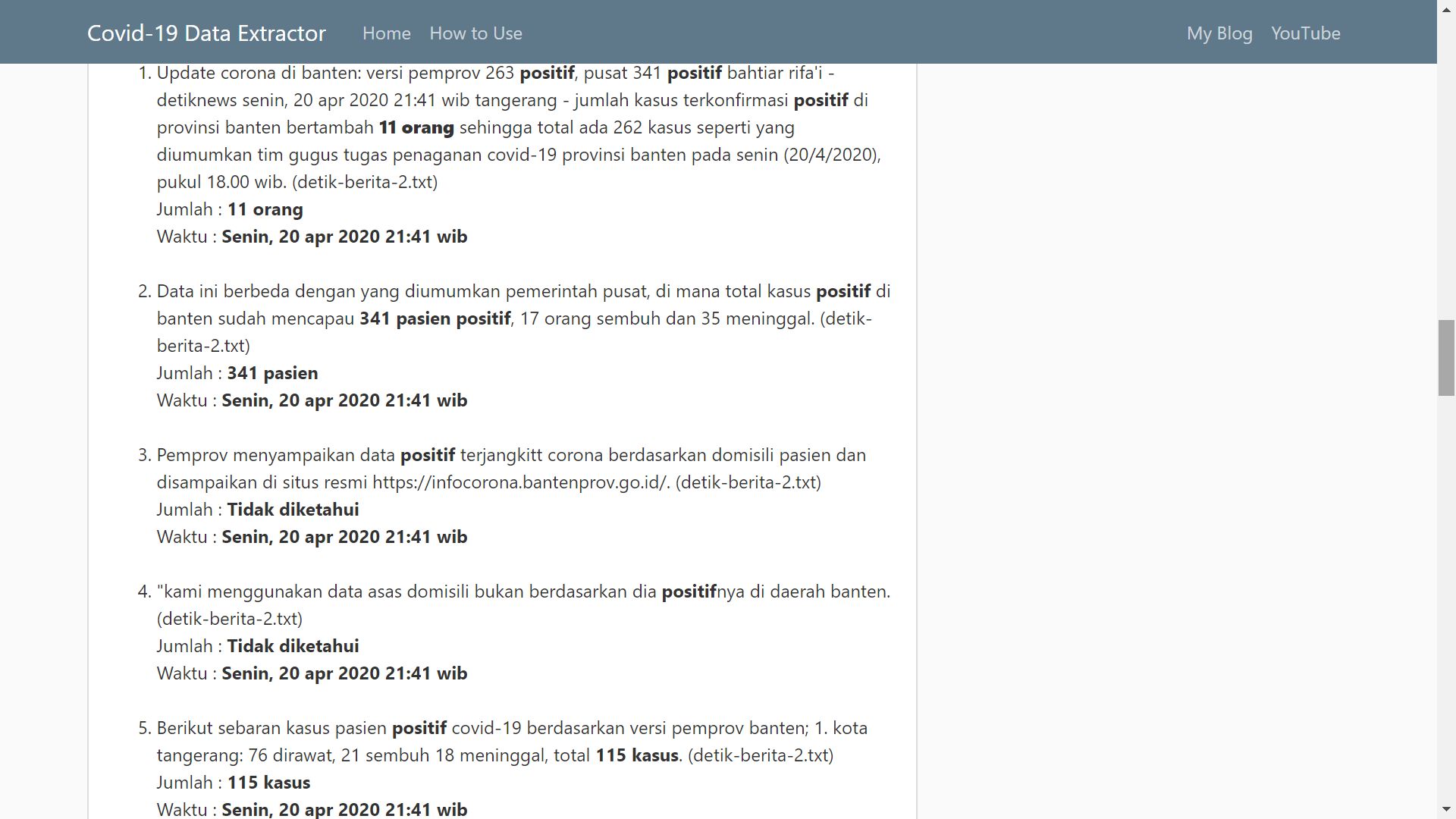
***Kasus 1****: Hasil Ekstraksi Data: Menampilkan Kembali Artikel yang Diekstrak*

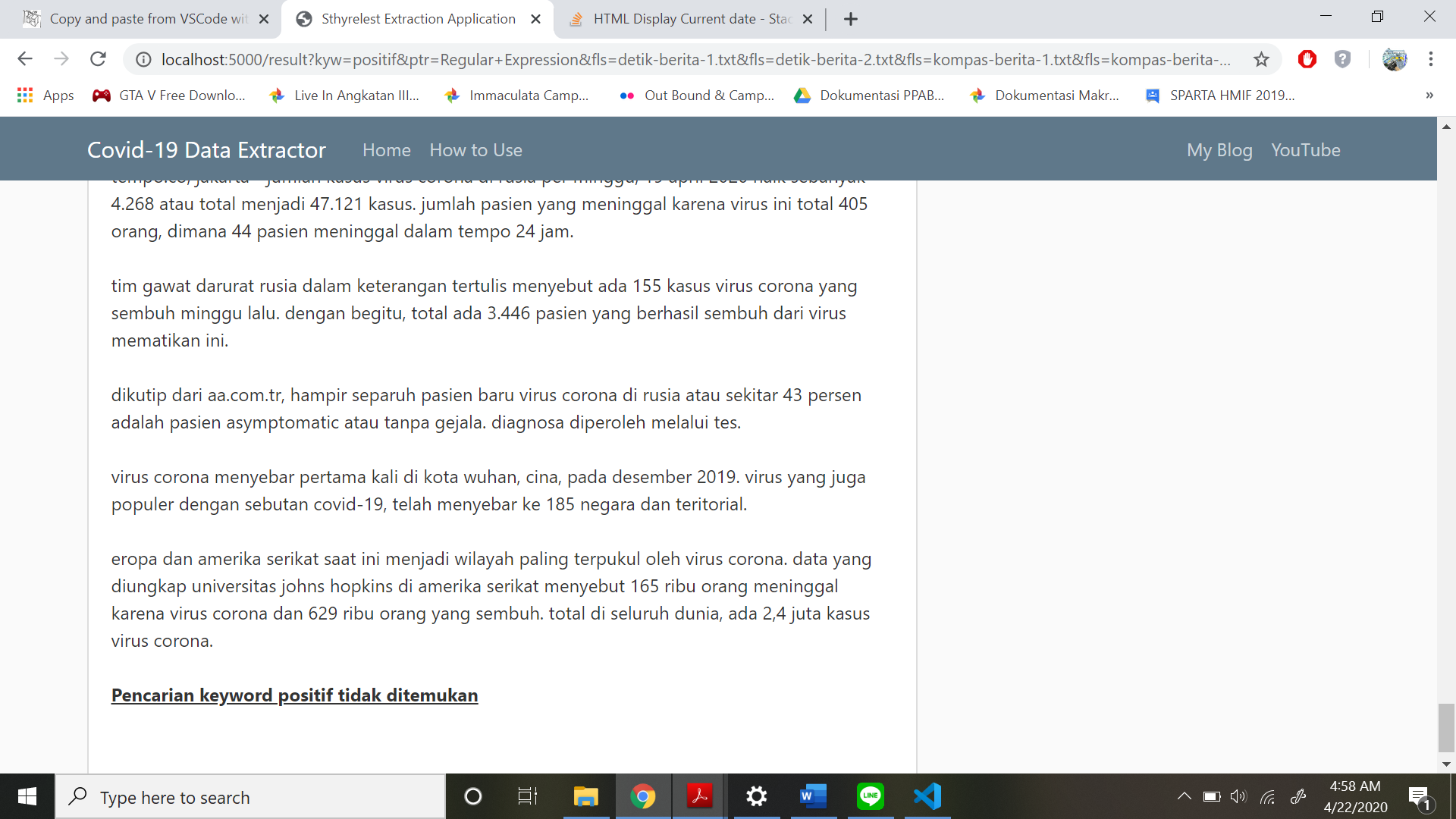
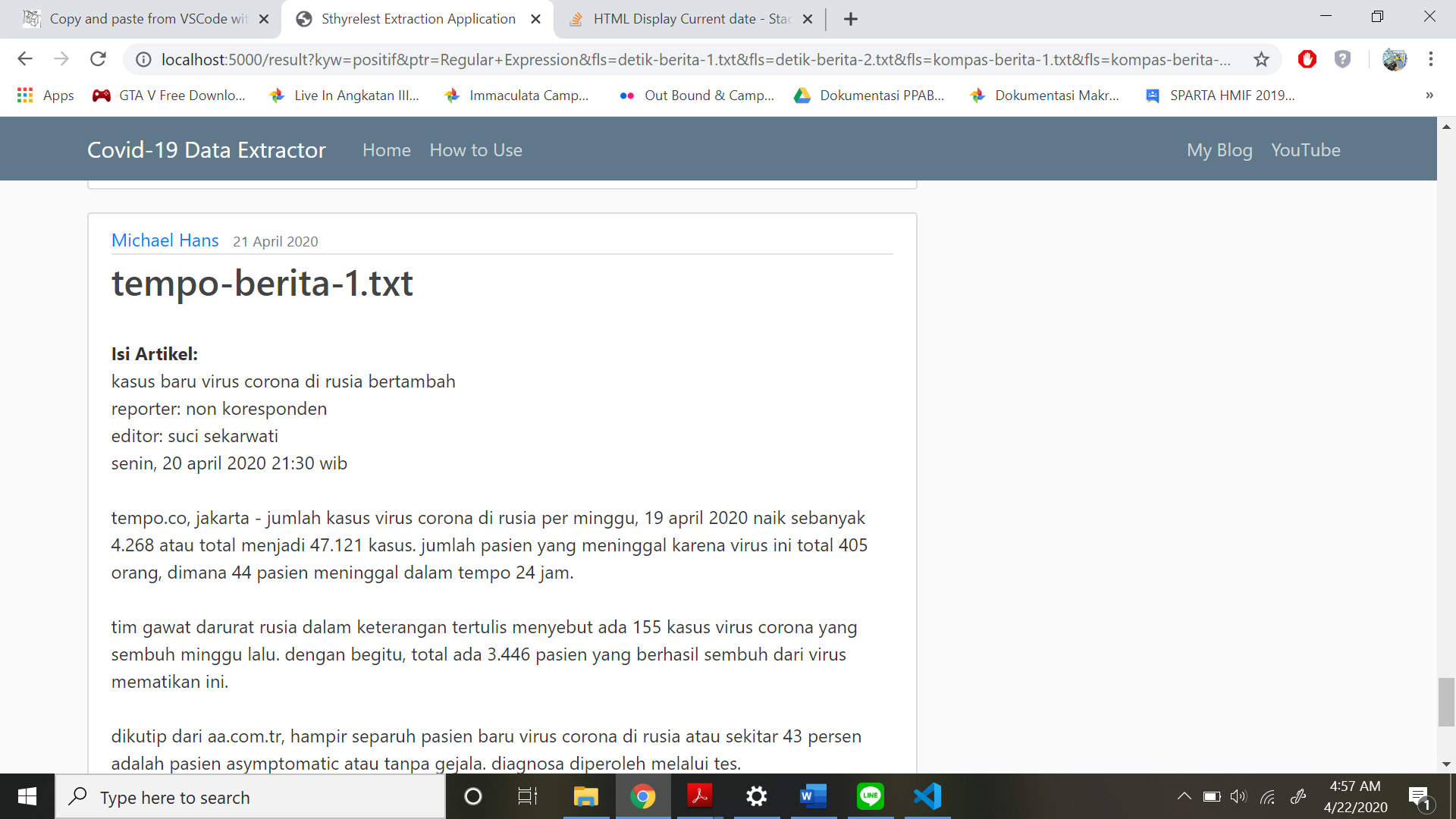


***Kasus 1****: Menampilkan Hasil Ekstraksi Data dari Artikel pada detik-beria-1.txt*



***Kasus 2****: Menampilkan isi artikel dari detik-berita-2.txt*

***Kasus 2****: Menampilkan hasil ekstraksi data dari detik-berita-2.txt*



***Kasus 3:*** *Menampilkan hasil ekstraksi data dari tempo-berita-1.txt: Keyword positif tidak ditemukan*

**3.3. Checklist Program**

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | ✓ |  |
| 1. Program berhasil running | ✓ |  |
| 1. Program dapat menerima input dan menuliskan output | ✓ |  |
| 1. Luaran sudah benar untuk semua data uji | ✓ |  |

**DAFTAR PUSTAKA**

Rinaldi Munir. “Pencocokan String”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf (diakses pada 15 April 2020)

Rinaldi Munir. “String Matching dengan Regex”https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf (diakses pada 20 April 2020)

Nur Ulfa Maulidevi. “Pencocokan String dengan Algoritma Knutt-Morris-Pratt”. <https://www.youtube.com/watch?v=5oSQnywqm0M&>.

Nur Ulfa Maulidevi. “Pencocokan String dengan Algoritma Boyer-Moore”. <https://www.youtube.com/watch?v=AvFx1vc32xY&>

Masayu Leylia Khodra, Yudi Wibisono. Regular Expression. <https://docs.google.com/document/d/1TUkkiQNL> nwbP0Vq46K5OL1\_7dACrJLrC/edit (diakses pada 20 April 2020)