TUGAS KECIL 4
IF2211 STRATEGI ALGORITMA

STRING MATCHING ALGORITHM
KNUTH-MORRIS-PRATT
BOYER-MOORE
REGULAR-EXPRESSION

Disusun oleh:

Jones Napoleon
13518086

Mata Kuliah Strategi Algoritma
Semester 2 Tahun Ajaran 2019/2020

Prodi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung 2020

BAB I

STRING MATCHING DAN DASAR TEORI ALGORITMA

1.1. *OVERVIEW*

Algoritma pencarian string (*string matching algorithm*) atau sering disebut juga pencocokan string adalah algoritma untuk melakukan pencarian semua kemunculan string pendek (*pattern*) di string yang lebih panjang (*text*).

Contoh aplikatif dari pencarian teks biasa dapat ditemukan di bidang bioinformatika seperti mencari apakah ada tidaknya bagian DNA tertentu misalnya ACGT pada DNA seorang individu.

Pada aplikasi ini, program yang dijalankan berlaku untuk algoritma Knuth-Morris-Pratt, Boyer-Moore, dan Regular Expression. Agar, program dapat berjalan sesuai spek, ingat untuk selalu mencentang checkbox pada **TubesMode**

Secara dasar, program menerima *input* berupa:
- Algoritma yang akan digunakan
- Input teks (apakah berupa file atau teks)
- String pendek (*pattern*) yang akan digunakan untuk mencocokkannya dengan teks

Secara opsional, program juga dapat melakukan:
- Case-sensitive (tidak berlaku untuk Regex ataupun Tubes mode)
- Web-scraping dasar (sangat dasar, dan tidak berlaku untuk SSR)

Kemudian dari sejumlah *input* tersebut, program akan melakukan pencarian *string* sesuai dengan algoritma yang dipilih.

Note: PROGRAM HANYA MENCARI DAN MENELUSURI KALIMAT YANG MANA MERUPAKAN KALIMAT ADANYA KEYWORD HASIL INPUT USER

1.3. ALGORITMA KNUTH-MORRIS-PRATT

Knuth-Morris-Pratt adalah algoritma yang akan mengecek setiap karakter hanya sekali dari kiri ke kanan, maka kompleksitasnya adalah O(n).

Awalnya, kita perlu mencari sebuah *prefixOrder* atau *border function* pada *pattern* dahulu, dan *prefixOrder* ini berfungsi untuk menentukan perlu lompat kemanakah indeks karakter di *pattern* jika suatu indeks karakter di *pattern* telah berbeda dengan

indeks karakter di teks. Jika panjang karakter *pattern* adalah **m**, maka kompleksitas fungsi *border* ini adalah O(**m**). Algoritma detailnya ada di bagian kedua

Setelah itu, akan dilakukan pengecekan string di teks secara linear dengan mengamati apakah sama tidaknya suatu karakter di *pattern* dengan di *text* sampai habis.

Jika menemukan sama, maka akan di-*append* ke suatu *output_array*, jika menemukan sampai setengah dan ada yang salah, maka kembali ke nilai dari karakter di *border_function* dan dilanjutkan lagi tanpa kembali ke teks sebelumnya.

1.3. ALGORITMA BOYER-MOORE

Boyer-Moore adalah algoritma yang akan mengecek setiap karakter hanya sekali dari kiri ke kanan, maka kompleksitasnya adalah O(n).

Awalnya, kita perlu mencari sebuah *prefix function* pada *pattern* dahulu, dan *prefix function* ini berfungsi untuk menentukan perlu lompat kemanakah indeks karakter di *pattern* jika suatu indeks karakter di *pattern* telah berbeda dengan indeks karakter di teks. Jika panjang karakter *pattern* adalah **m**, maka kompleksitas fungsi *border* ini adalah O(**m**). Algoritma detailnya ada di bagian kedua

Setelah itu, program akan melakukan pengecekan string di teks secara linear dengan mengamati apakah sama tidaknya suatu karakter di *pattern* dengan di *text* sampai habis dengan. Caranya, disamakan dari karakter terbelakang pattern hingga yang terdepan.

Jika menemukan sama, maka akan di-*append* ke suatu *output_array*, jika menemukan sampai setengah dan ada yang salah, maka kembali ke nilai dari karakter di *border_function* dan dilanjutkan lagi tanpa kembali ke teks sebelumnya.

BAB II

IMPLEMENTASI KODE PROGRAM DALAM BAHASA PYTHON

2.1. KNUTH-MORRIS-PRATT

Di bawah ini merupakan fungsi untuk mencari *border function* dan implementasi dasar algoritma KMP

```python
def get_prefix(pattern, is_case_sensitive):
    if not is_case_sensitive:
        pattern = pattern.lower()

    prefix = [0 for i in range(len(pattern))]
    j = 0

    for i in range(1, len(pattern)):
        val = prefix[i - 1]
        if prefix[i - 1] == 0:
            j = 0
            val = 0

        if pattern[i] == pattern[j]:
            prefix[i] = val + 1
        else:
            while j < i and pattern[i] != pattern[j
                j += 1
            prefix[i] = 0 if j == i else 1
        j += 1

    return prefix
```

Gambar 1. Fungsi mencari *prefix function* pada algoritma *Knuth-Morris-Pratt*

```python
def knuth_morris_pratt(data):
    pattern = data['keywords']
    text = data['text']
    is_case_sensitive = data['case-sensitive']

    border = get_prefix(pattern, is_case_sensitive)
    pattern = '0' + pattern
    border.insert(0, 0)
    response = []
    j = 0

    if is_case_sensitive:
        for i in range(len(text)):
            while j != 0 and pattern[j + 1] != text[i]:
                j = border[j]
            if text[i] == pattern[j + 1]:
                i += 1
                j += 1
            if j == len(border) - 1:
                response.append(i- len(pattern) + 1)
                j = 0
    else:
        for i in range(len(text)):
            while j != 0 and pattern[j + 1].lower() != text[i].lowe
                j = border[j]
            if text[i].lower() == pattern[j + 1].lower():
                i += 1
                j += 1
            if j == len(border) - 1:
                response.append(i- len(pattern) + 1)
                j = 0

    return render_result(response, text, pattern[1:])
```

Gambar 2. Fungsi implementasi *Knuth-Morris-Pratt function* secara dasar.

Tampak pada fungsi dasar Knuth-Morris-Pratt bahwa setiap menemukan *text* yang sesuai dengan *pattern*, indeks awal dari *text* tersebut akan di-*append* ke *array* **response**. Input dan output program akan menjadi dasar jika hanya algoritma dasarnya yang diimplementasikan.

Akan tetapi, karena diperlukan dicarinya identifikasi tanggal ataupun jumlah yang ada, dan di-*render*nya dalam *valid* HTML, maka masih diperlukan pengecekan adanya tanggal ataupun jumlah pada teks tersebut dan jika iya, mengubahnya menjadi ber-HTML, sehingga ketika di-*render* secara "*dangerous"*, akan langsung berbeda.

```python
def tubes_knuth_morris_pratt(data):
    pattern = data['keywords']
    text = data['text']
    sentences = extract_text_to_sentence(text)

    response = []
    statements = []

    border = get_prefix(pattern, False)
    pattern = '0' + pattern
    border.insert(0, 0)

    j = 0

    for i in range(len(sentences)):
        sentence = sentences[i]

        for position in range(len(sentence)):
            while j != 0 and pattern[j + 1].lower() != sentence[position].lower():
                j = border[j]
            if sentence[position].lower() == pattern[j + 1].lower():
                position += 1
                j += 1
            if j == len(border) - 1:
                response.append({"sentence-order": i, "relative-pos": position - len(pattern) + 1 })
                statements.append(parse_information(sentence))
                j = 0

    return generator(response, statements, sentences)
```

Gambar 3. Fungsi kompleks KMP untuk spek tubes.

Tahap pertama yang dilakukan adalah mempartisi suatu *text* (string) menjadi *sentences* (array of *sentence*) – **extract_text_to_sentence**, dan dari sana melakukan sesuai dengan algoritma KMP biasa. Tetapi, jika telah menemukan *pattern* yang sesuai, tidak hanya **response** yang akan di-*append* indeks string pada *sentence* itu saja, tetapi juga dibuat **statements** yang akan di-*append* identifikasi tanggal maupun jumlah yang ada. Untuk mengidentifikasi ada tidaknya, tanggal ataupun jumlah digunakan fungsi **parse_information.** Setelah semuanya, akan dilakuakan **generator** untuk menghasilkan output yang dapat di-*paste* secara *dangerous* oleh HTML.

```python
def parse_information(one_sentence):
    numbers_pattern = get_number()
    moment_pattern = get_time()
    numbers = []
    moments = []

    for matched_number in re.finditer(numbers_pattern, one_sentence):
        numbers.append({'span': matched_number.span(), 'group': matched_number.group(0).strip()})

    for matched_moment in re.finditer(moment_pattern, one_sentence):
        moments.append({'span': matched_moment.span(), 'group': matched_moment.group(0).strip()})

    final = set_final(numbers, moments)
    copy_number = [f for f in final['final_numbers']]
    copy_moment = [f for f in final['final_moments']]

    html = set_html(one_sentence, final['final_numbers'], final['final_moments'])

    return {
        "innerHTML": html,
        "number": copy_number,
        "moment": copy_moment,
    }
```

Gambar 4. Implementasi *function* **parse_information**

Fungsi **get_number** dan **get_time** merupakan kumpulan *Regex* yang dilakukan untuk menentukan pola *numbers* dan *moments* (*time*) secara berturut-turut. Fungsi **set_final** berguna untuk menghilangkan duplikat ataupun fungsi yang terdeteksi dua kali di *moment* dan *number*; sedangkan fungi **set_html** bertugas untuk me-*replace* "string" dengan "<mark>string</mark>" agar tertanda ketika di-*paste* di HTML.

Setelah disini, perlu diketahui bahwa *array* **response** dan *array* **statements** akan berkorespondensi satu satu dengan indeks sebagai *id FULL JOIN*.

Jadi, di fungsi generator, akan dilakukan operasi yang menggabung hasil **response** dan **statements** membentuk suatu *renderable* HTML.

```
def generator(response, statements, sentences):
    temp = []
    plain_ouput = []
    final_inner_HTML = ""
    for res in response:
        temp.append(res['sentence-order'])

    if len(temp) > 0:
        count = temp.pop(0)
        for i in range(len(sentences)):
            if count == i:
                final_inner_HTML += statements[0]['innerHTML'] + '\n'
                plain_ouput.append(manage(statements[0]))
                if len(temp) > 0:
                    count = temp.pop(0)
                    statements.pop(0)
            else:
                final_inner_HTML += sentences[i] + '\n'
    else:
        final_inner_HTML = sentences

    return {
        "raw_output": plain_ouput,
        "inner_HTML": final_inner_HTML
    }
```

Gambar 5. Implementasi fungsi **generator**

2.2. BOYER_MOORE

Di bawah ini merupakan fungsi untuk mencari *prefix jump* pada *pattern* di *array*

```
def get_prefix(pattern, is_case_sensitive):
    if not is_case_sensitive:
        pattern = pattern.lower()
    length = len(pattern)
    prefix = {}

    for i in range(len(pattern)):
        char = str(pattern[i])
        prefix[char] = (length - i - 1) if i != (length - 1) else length
    prefix['*'] = length

    return prefix
```

Gambar 6. Implementasi fungsi **get_prefix** pada Boyer-Moore

```python
def boyer_moore(data):
    pattern = data['keywords']
    text = data['text']
    is_case_sensitive = data['case-sensitive']

    response = []
    border = get_prefix(pattern, is_case_sensitive)
    length = len(pattern) - 1
    last = length
    i = 0

    if not is_case_sensitive:
        while last < len(text):
            while i < length and text[last].lower() == pattern[length - i].lower():
                last -= 1
                i += 1
            if i == length and text[last].lower() == pattern[length - i].lower():
                response.append(last)
                last += length
            i = 0

            last += border[text[last].lower()] if text[last].lower() in border else border['*']

    else:
        while last < len(text):
            while i < length and text[last] == pattern[length - i]:
                last -= 1
                i += 1
            if i == length and text[last] == pattern[length - i]:
                response.append(last)
                last += length
            i = 0

            last += border[text[last]] if text[last] in border else border['*']

    return render_result(response, text, pattern)
```

Gambar 7. Implementasi **boyer_moore** secara dasar

```python
def tubes_boyer_moore(data):
    pattern = data['keywords']
    text = data['text']
    sentences = extract_text_to_sentence(text)
    response = []
    statements = []

    border = get_prefix(pattern, False)
    length = len(pattern) - 1

    for j in range(len(sentences)):
        sentence = sentences[j]
        last = length
        i = 0
        while last < len(sentence):
            while i < length and sentence[last].lower() == pattern[length - i].lower():
                last -= 1
                i += 1
            if i == length and sentence[last].lower() == pattern[length - i].lower():
                response.append({"sentence-order": j, "relative-pos": last})
                statements.append(parse_information(sentence))
                last += length
            i = 0
            last += border[sentence[last].lower()] if sentence[last].lower() in border else border['*']
    return generator(response, statements, sentences)
```

Gambar 8. Implementasi **boyer_moore** sesuai spek tubes

Sama dengan pada pola pemograman yang ada pada KMP, Boyer-Moore juga terbagi atas yang tubes dan yang tidak.

Fungsi-fungsi yang ada di **boyer_moore** seperti **generator, parse_information**, **extract_text_to_sentence** merupakan fungsi yang sama dengan saat KMP.

2.3. *REGULAR EXPRESSION*

Pada Regular Expression, program tidak banyak memiliki algoritma karena pada dasarnya hanya melakukan regex (library *built-in* yang ada di Python). Pada regular expression pula, tidak dilakukan pencarian indeks pendukung seperti *border function* pada *pattern*.

```python
import re

def regular_expression(data):
    pattern = data['keywords']
    text = data['text']

    try:
        solution = re.findall(pattern, text)
        saved_solution = [sol for sol in solution]
        string = ""

        if len(solution) > 0:
            to_be_checked = solution.pop(0)
            index = 0
            while index < len(text):
                while text[index : index + len(to_be_checked)] != to_be_checked:
                    string += text[index]
                    index += 1
                string += "<mark>" + to_be_checked + "</mark>"
                index += len(to_be_checked)
                if len(solution) > 0:
                    to_be_checked = solution.pop(0)
                else:
                    string += text[index: index + len(text)]
                    break
        else:
            string = text

        response = {
            "inner_HTML": string.strip(),
            "raw_output": saved_solution,
        }

    except:
        response = {
            "inner_HTML": "<h4 class='highlight'>Keywords error</h4>",
            "raw_output": [],
        }

    return response
```

Gambar 9. Implementasi pencarian pada **regular_expression**

Dari tadi, dapat dilihat implementasi algoritma utama selalu dimulai dengan **data** yang kemudian diambil bagian **keywords** dan **text**-nya. Untuk mengetahui darimana **data** dapat diambil, sedikit bagian HTML semantik dan aplikasi backend akan diperlihatkan.

2.3. FORM HTML AND BACKEND

```
<form class="row" action="/" method="POST" enctype="multipart/form-data">

            <section class="col-sm-12 col-md-6 col-xl-4">

                <legend class="lead">Choose algorithm: </legend>

                <div>

                    <input type="radio" id="regex" name='algorithm' value="regex" {% if previous["
algorithm"] == 'regex' %} checked {% endif %} >

                    <label for="regex">Regular Expression</label>
```

```
                </iv>
                <div>
                    <input type="radio" id="kmp" name='algorithm' value="kmp" {% if previous["algo
rithm"] == 'kmp' %}checked{% endif %} >
                    <label for="kmp">Knuth-Morris-Pratt</label>
                </div>
                <div>
                    <input type="radio" id="bm" name='algorithm' value="bm" {% if previous["algori
thm"] == 'bm' %}checked{% endif %} >
                    <label for="bm">Boyer-Moore</label>
                </div>
            </section>

            <section class="col-sm-12 col-md-6 col-xl-4">
                <legend class="lead">Choose input method: </legend>
                <div>
                    <input type="radio" id="text" name='input-
method' value="text" {% if previous["input-method"] == 'text' %} checked {% endif %}>
                    <label for="text">Text</label>
                </div>
                <div>
                    <input type="radio" id="file" name='input-
method' value="file" {% if previous["input-method"] == 'file' %} checked {% endif %}>
                    <label for="file">File</label>
                </div>
                <div>
                    <input type="radio" id="web" onclick="window.location.href = '/web';" {% if pr
evious["input-method"] == 'web' %} checked {% endif %}>
                    <a href='/web' id='cta-web'>
                        <label for="web">Web (Beta)</label>
                    </a>
                </div>
            </section>

            <section class="mt-3 col-xs-12 col-md-12 col-xl-4">
                <legend class="lead">
```

```
                    <a href="#" data-toggle="tooltip" data-
html="true" title="For regex,<br><b class='lead'>Do: <code>[^bcr]at</code></b><br><b>Don't: <code>
/[^bcr]at/g</code></b>">
                        <i class="fas fa-info-circle"></i>
                    </a>
                    Keywords:
                </legend>
                <div>
                    <input type="text" id="keywords" name='keywords' placeholder="Type here..." va
lue='{{ previous["keywords"] }}' required>
                </div>
                <div class="form-check">
                    <input type="checkbox" class="form-check-input" name='case-
sensitive' id="case-sensitive" value='{{ previous["case-sensitive"]}}' {% if previous["case-
sensitive"] == 'true' %} checked {% endif %}>
                    <label class="form-check-label" for="case-sensitive">
                        Case sensitive
                        <a href="#" data-toggle="tooltip" data-
html="true" title="Note that this case match is not applicable for Regular Expression">
                            <i class="fas fa-info-circle"></i>
                        </a>
                    </label>
                </div>
                <div class="form-check">
                    <input type="checkbox" class="form-check-input" name='tubes-mode' id="tubes-
mode" value='{{ previous["tubes-mode"]}}' {% if previous["tubes-
mode"] == 'true' %} checked {% endif %}>
                    <label class="form-check-label" for="case-sensitive">
                        Tubes mode
                        <a href="#" data-toggle="tooltip" data-
html="true" title="For general purposes, don't check this checkbox<br>Read more at my official REA
Dme to understand the usage of tubes mode">
                            <i class="fas fa-info-circle"></i>
                        </a>
                    </label>
                </div>
                <button type='submit' class="mt-1 btn btn-primary btn-md">Find</button>
            </section>
```

```
            <section class="mt-3 col-xs-12 col-md-12">
                <div id='text-chosen' {% if previous["input-
method"] != 'text' %} style="display: none;" {% endif %}>
                    <legend class="lead">Input text: </legend>
                    <textarea id='input-text' name='input-
text' rows='3' placeholder="Copy your text here..." required>{{ previous["text"] }}</textarea>
                </div>
                <div id='file-chosen' {% if previous["input-
method"] != 'file' %} style="display: none;" {% endif %}>
                    <legend class="lead">Input file: </legend>
                    <input type="file" id='input-file' accept="text/plain" name='input-
file' onchange="this.files[0].text().then(t => process(t))">
                </div>
            </section>

            <aside id="before-parsed" class="mt-3 col-xs-12 col-lg-12">
                <!-- AREA FOR INPUT FILE -->
            </aside>
        </form>
```

Gambar 10. Form POST

Kemudian, dari form yang telah di-*POST*, akan teresktrak di backend dan akan di-**preprocess** sehingga kemudian dapat langsung di-*pass* sebagai *argument* pada fungsi utama algoritma **knuth_morris_pratt** atau **regular_expression** atau **boyer_moore**.

```
def preprocess(data):
    temp = {}
    temp['algorithm'] = data['algorithm'][0]
    temp['keywords'] = data['keywords'][0]
    temp['text'] = data['input-text'][len(data['input-text']) - 1]
    try:
        if data['case-sensitive']:
            temp['case-sensitive'] = True
    except:
        temp['case-sensitive'] = False
    try:
        if data['tubes-mode']:
            temp['tubes-mode'] = True
    except:
        temp['tubes-mode'] = False

    return temp
```

Gambar 11. Fungsi **preprocess**

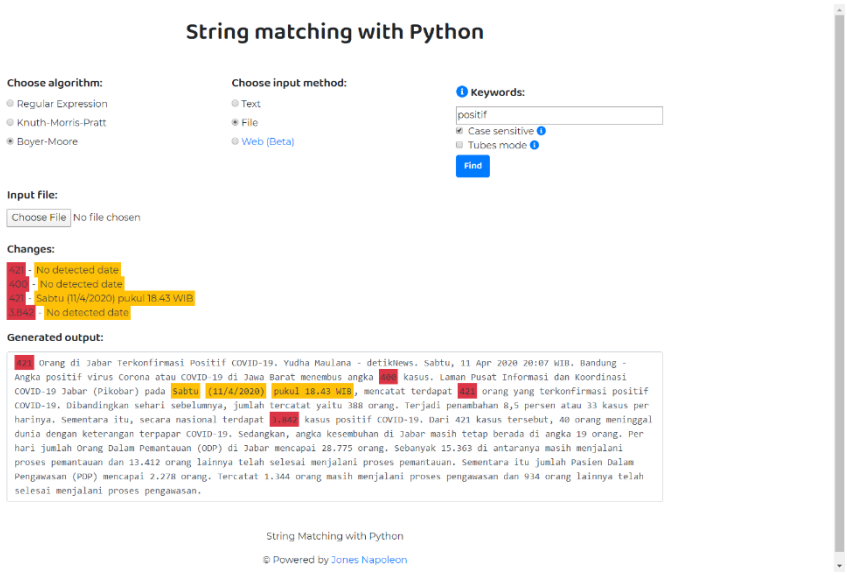
```
if data['algorithm'] == 'regex':
    result = regular_expression(data)
elif data['algorithm'] == 'kmp':
    result = knuth_morris_pratt(data)
else:
    result = boyer_moore(data)
```

Gambar 12. Proses sederhana backend

BAB III

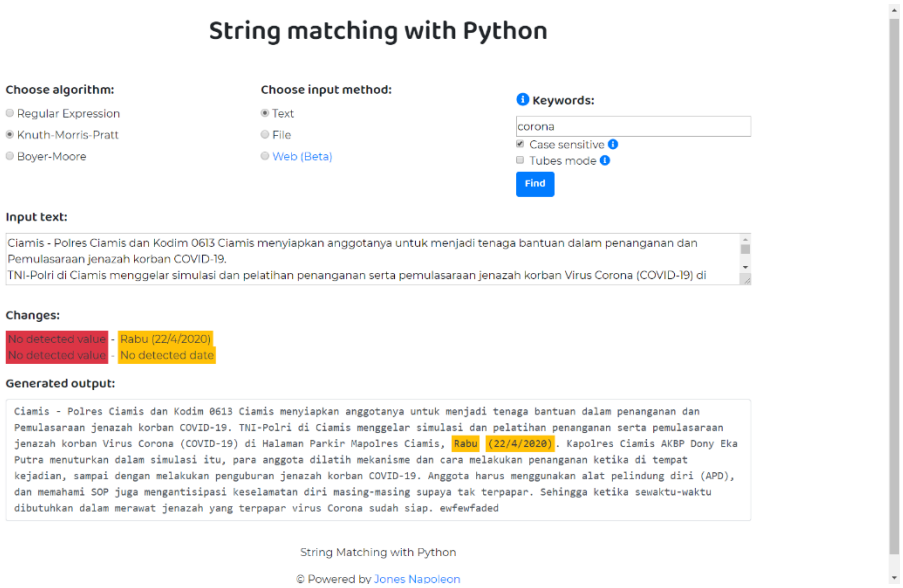
SCREENSHOT HASIL PROGRAM

3.1. INPUT 1 (SPEK.TXT)



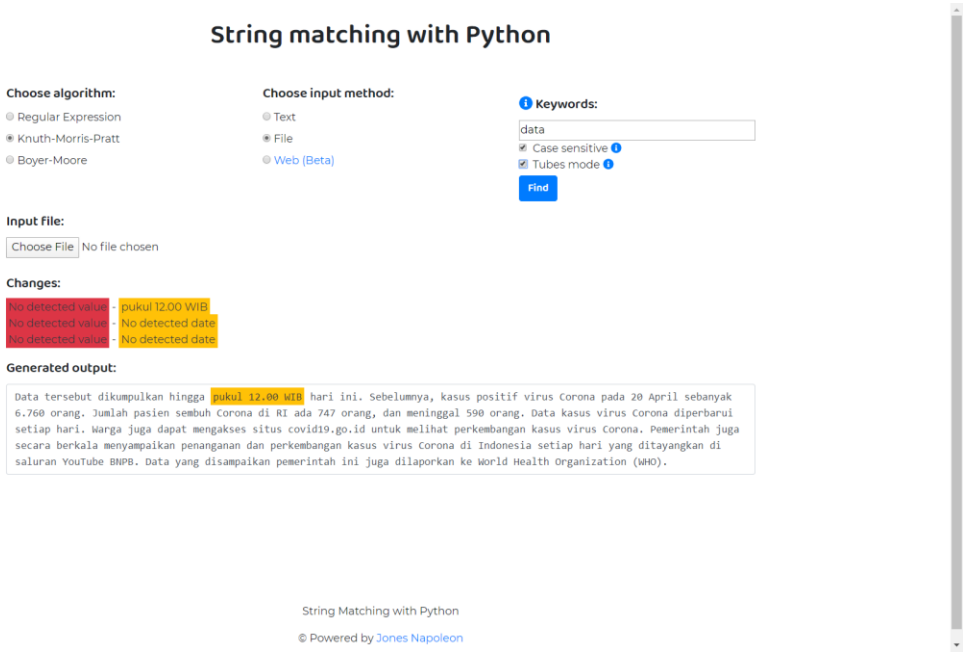
Karena program hanya mencari di *sentence* yang ada *keyword* **positif** dan ada 4 kalimat, maka pada masing-masing kalimat ditulis jumlah beserta harinya jika terdeteksi.

3.2. INPUT 2 (CORONA.TXT)



Karena hanya terdapat 2 kata Corona ataupun corona pada teks tersebut, maka tertampilkanlah 2 **changes.** Pada *sentence* pertama yang terdapat kata [Cc]orona, terdapat identifikasi tanggal yakni: **Rabu (22/4/2020)**, tetapi tidak terdapat angka, maka baris pertama *changes* menjadi: **No detected value - Rabu (22/4/2020).** Karena pada *sentence* kedua adanya kata [Cc]orona (*Sehingga ketika sewaktu-waktu dibutuhkan dalam merawat jenazah yang terpapar virus Corona sudah siap.*), tidak terdapat identifikasi angka maupun tanggal, maka baris kedua *changes* menjadi: **No detected value – No detected date.**

3.3. INPUT 3 (CORONA_LAGI.TXT)



Penjelasan input 3 sama dengan input 2

This project's website**:** **https://string-matching.herokuapp.com**

Developer's official website**:** **https://jonesnapoleon.web.app**

BAB IV

LAIN-LAIN

4.1. PENILAIAN INDIVIDU

No	Poin	Ya	Tidak
1.	Program berhasil dikompilasi	√	
2.	Program berhasil running	√	
3.	Program dapat menerima input dan menuliskan output.	√	
4.	Luaran sudah benar untuk semua n	√	

4.2. SPESIFIKASI PERANGKAT KERAS YANG DIGUNAKAN