Laporan Tugas Kecil 4
IF 2211 Strategi Algoritma
Ekstraksi Informasi dari Artikel Berita dengan Algoritma Pencocokan String

Disusun Oleh:
Farras Mohammad Hibban Faddila
13518017

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2019/2020

Daftar Isi

A. Algoritma Pencocokan String

a. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP merupakan sebuah algoritma pencocokan string yang memiliki kompleksitas O(N), dengan N merupakan panjang dari string teks pencarian. Urutan pencariannya adalah dari kiri ke kanan, serupa dengan algoritma *brute-force* biasa, namun terdapat sebuah optimisasi pada algoritma ini. Algoritma ini memanfaatkan nilai dari fungsi border_function, yakni untuk sebuah string S, border_function(i) didefinisikan sebagai panjang prefix terpanjang dari S[0..i] yang juga merupakan suffix dari S[1..i]. Jika dalam pengecekan terdapat sebuah *mismatch* pada saat melakukan pengecekan dengan karakter ke j dari pattern, maka program tidak akan melakukan pengecekan dari awal *pattern*, namun akan melakukan *shift* pattern sebanyak border_function(j), sehingga pointer pada teks tidak perlu diubah (cukup string pattern saja yang di-*shift*), sehingga hanya diperlukan satu kali iterasi saja, dan akibatnya kompleksitas algoritmanya adalah O(N).

b. Algoritma Boyer-Moore

Algoritma Boyer-Moore merupakan algoritma pencocokan string yang memiliki kompleksitas waktu terburuk O(MN+A), di mana A merupakan banyak alfabet, dan M,N merupakan panjang dari *pattern* yang dicari dan teks pencarian. Seperti KMP, iterasi starting index juga dilakukan dari kiri ke kanan, namun untuk setiap starting index, pencocokan string dimulai dari kanan. Hal ini karena Boyer-Moore menggunakan fungsi last_occurence dalam algoritmanya, yakni untuk sebuah karakter c, last_occurence(c) merupakan indeks kemunculan c yang terakhir atau yang paling kanan. Jika terjadi *mismatch* saat pengecekan string saat karakter teks bernial c, *pattern* akan di-*shift* bergantung pada apakah ada c di sebelah kiri *pattern*, atau jika c terletak di sebelah kanan, atau jika c tidak ada sama sekali pada *pattern*.

c. Regular Expression

Regular Expression merupakan notasi standar yang mendeskripsikan suatu *pattern* string. Penggunaan regex untuk pencocokan string cukup efisien karena kita dapat mendefinisikan pattern dengan sistematis. Sebagai contoh, terdapat karakter spsial ‘.’ yang dapat dicocokkan dengan karakter apapun. Selain itu, tipe-tipe spesial seperti huruf, angka, maupun *whitespace* juga dapat dicocokkan dengan mudah karena ada karakter spesial lain seperti ‘\d’,

‘\w’, dan lain-lain. Jumlah kemunculan tiap karakter juga dapat ditentukan rangenya.

B. Kode Program

a. Algoritma KMP

```
def prefix_function(pattern):
    length = len(pattern)
    prefix = [0 for i in range(length)]
    for i in range(1,length):
       j = prefix[i-1]
       while (j>0 and pattern[i]!=pattern[j]):
             j = prefix[j-1]
       if pattern[i] == pattern[j]:
             j += 1
       prefix[i] = j
    return prefix

def kmp(pattern, text):
    #Change null_char to character not appearing in the text
    NULL_CHAR = '#'
    n = len(pattern)
    m = len(text)
    meta_text = pattern + NULL_CHAR + text

    prefix_meta = prefix_function(meta_text)
    matching_position = []

    for i in range(n+1, len(prefix_meta)):
       if prefix_meta[i] == n:
             matching_position.append(i-2*n)

    return matching_position
```

Tabel 2.1. Algoritma KMP

b. Algoritma Boyer-Moore

```
def build_last(pattern):
    #don't forget to fill unexisting character of the text with
-1
    last = {}
    length = len(pattern)
    for i in range(length):
       last[pattern[i]] = i
```

```
    return last

def boyer_moore(pattern, text):

    last = build_last(pattern)
    for i in text:
       if i not in last:
             last[i] = -1

    matching_position = []

    n = len(pattern)
    m = len(text)
    if n > m:
       return -1

    start = 0
    while start <= (m-n):
       j = n-1

       while (j >= 0) and (pattern[j] == text[j+start]):
             j -= 1

       if j < 0:
             matching_position.append(start)
             if (start+n) < m:
                   start += n - last[text[start+n]]
             else:
                   start += 1
       else:
             start += max(1, j - last[text[start + j]])

    return matching_position
```

Tabel 2.2. Algoritma Boyer-Moore

c. Regular Expression

i. Pencarian Waktu

```
def find_time(sentence):

    #currently only support date formatting
    time_pattern =
re.compile(r"[0-9]{1,2}[-/][0-9]{1,2}[-/][0-9]{2,4}")
    times = time_pattern.findall(sentence)
    return times
```

Tabel 2.3. Regex untuk pencarian tanggal pada kalimat

ii. Pencarian Jumlah

```
def find_amount_info(sentence):
# mengembalikan lokasi beserta string yang cocok dengan pattern
# lokasi digunakan untuk mencari angka yang terdekat
    amounts = [(m.start(0),m.group()) for m in re.finditer(r"[0-9]+\.*[0-9]*",
sentence)]
    return amounts
```

Tabel 2.4. Regex untuk pencarian nilai jumlah pada kalimat

d. Extractor

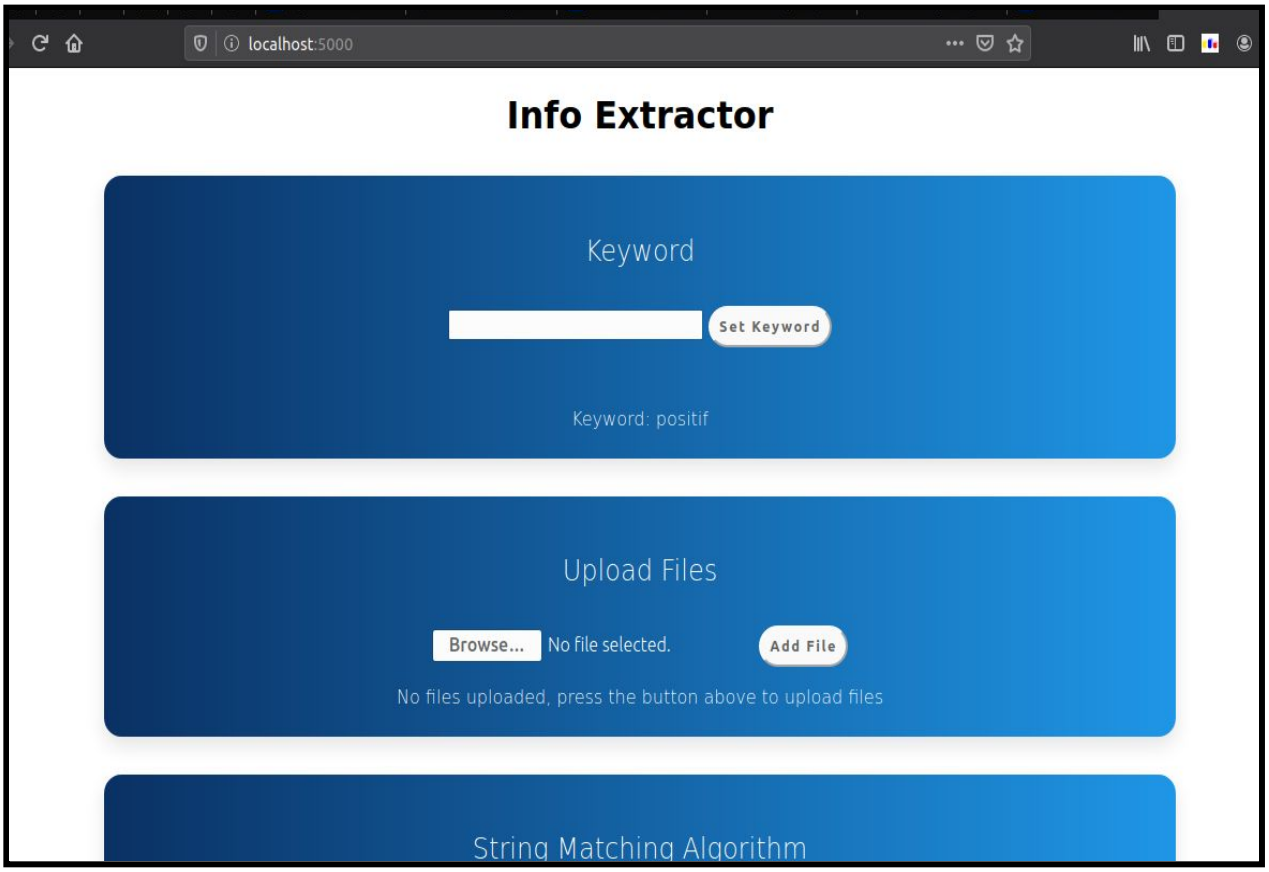
```
def extractor(keyword, sentences, option):

    informations = []
    for sentence in sentences[1]:
       if option == 'kmp':
             positions = kmp(keyword, sentence)
       elif option == 'bm':
             positions = boyer_moore(keyword, sentence)
       elif option == 'regex':
             positions = [m.start(0) for m in re.finditer(keyword, sentence)]
       if positions:
             time = find_time(sentence)
             if len(time) == 0:
                   for other_sentence in sentences[1]:
                         time = find_time(other_sentence)
                         if time:
                               break
             amount = find_amount_info(sentence)
             closest_amount = ''
             if amount:
                   closest_amount = amount[0][1]
                   if len(amount) > 1:
                         # find the closest one with the keyword
                         minimal = len(sentence)
                         for amount_candidate in amount:
                               if abs(amount_candidate[0] - positions[0]) < minimal:
                                     closest_amount = amount_candidate[1]
                                     minimal = abs(amount_candidate[0] - positions[0])
             informations.append([keyword, sentences[0], time[0], closest_amount, sentence])
```

```
    return informations
```

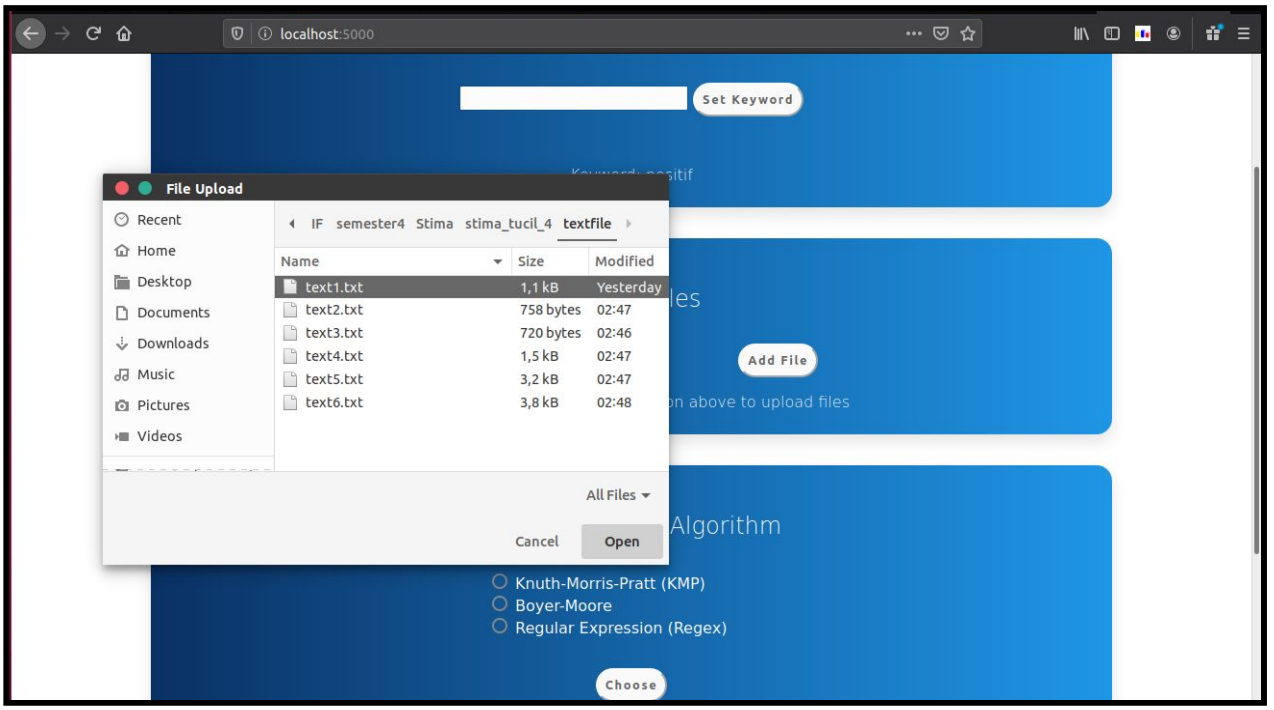
Tabel 2.5. Fungsi ekstraktor informasi utama yang dijalankan pada backend untuk menghasilkan data yang akan ditampilkan

C. Screenshot Program

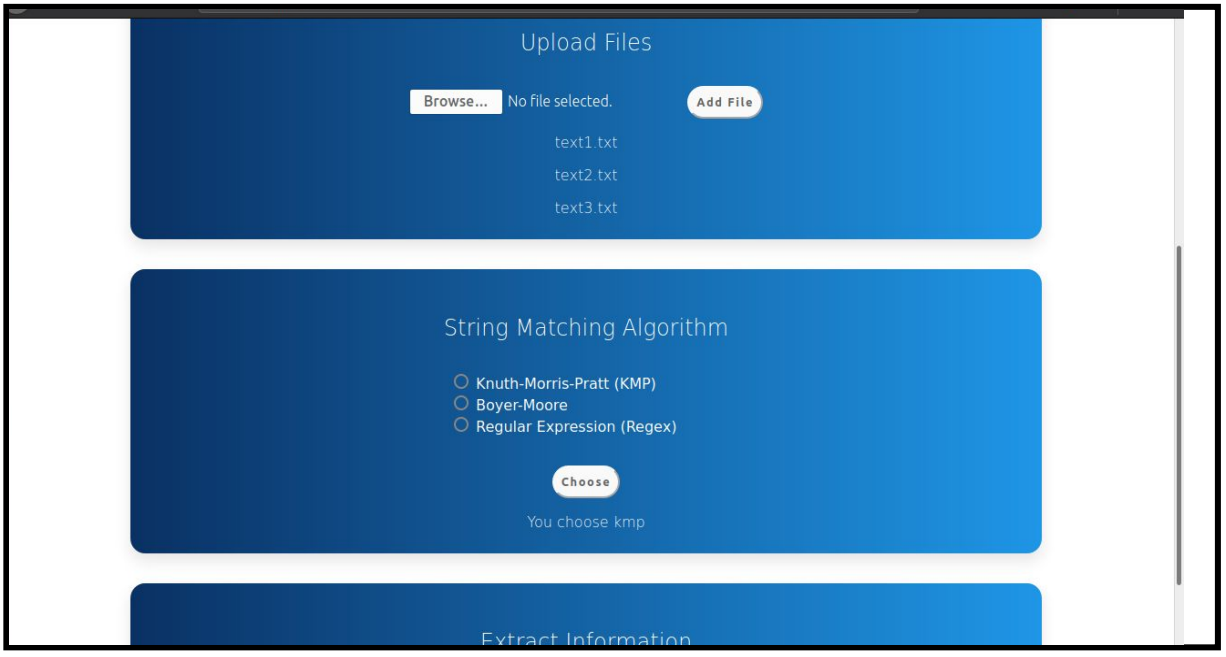
Program merupakan aplikasi berbasis web, dengan menggunakan Flask sebagai back-end nya, dan HTML-CSS sebagai front-end dan stylingnya. Program hanya terdiri atas satu halaman saja. Berikut ini merupakan beberapa *screenshot* laman aplikasi tersebut.



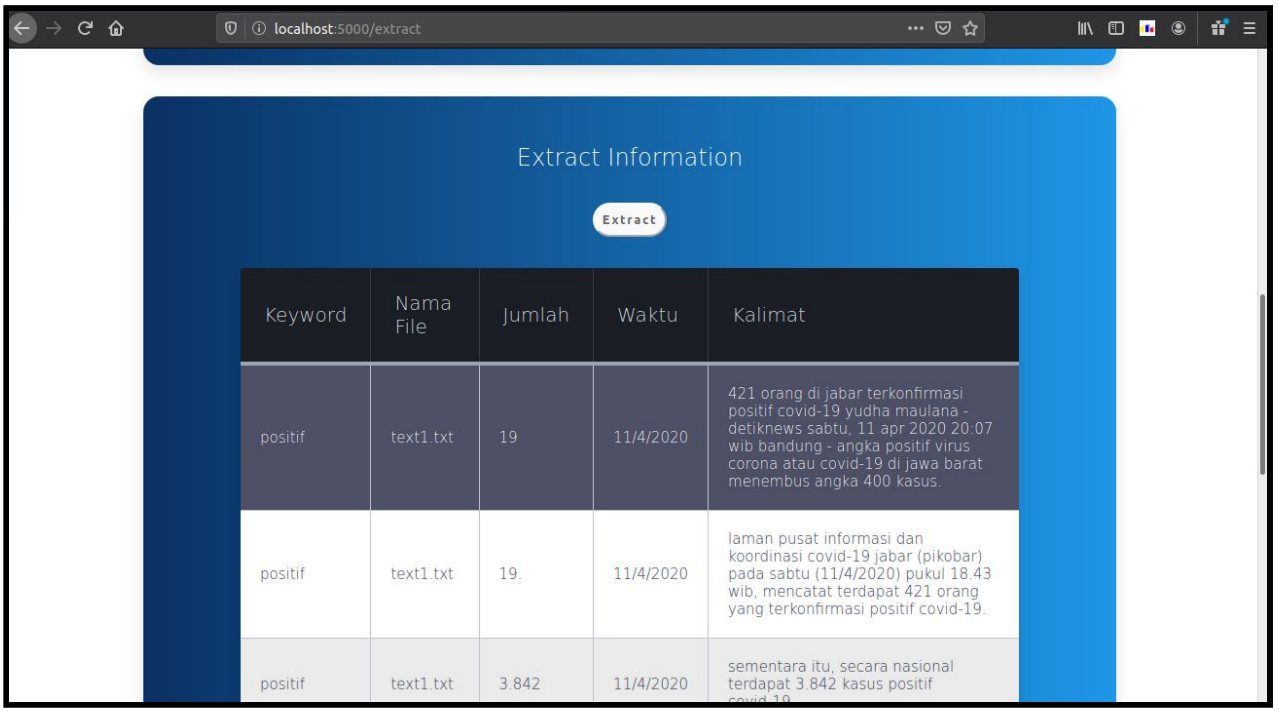
Gambar 3.1. Aplikasi Utama (Sumber: Penulis)



Gambar 3.2. Tampilan input file (Sumber: Penulis)



Gambar 3.3. Tampilan pemilihan algoritma pencocokan string (Sumber: Penulis)



Gambar 3.4. Hasil query dengan keyword ‘positif’ (Sumber: Penulis)

Gambar 3.5. Hasil query dengan keyword ‘positif’ (Sumber: Penulis)

D. Cek List

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil running	✓	
3. Program dapat menerima input dan menuliskan output.	✓	
4. Luaran sudah benar untuk data uji		✓

*Catatan: Seluruh kemungkinan data waktu yang ada belum dimasukkan pada regexnya, sehingga kemungkinan besar ada data uji yag tidak menghasilkan jawaban yang lengkap.