LAPORAN TUGAS KECIL 4

IF2211 Strategi Algoritma

Ekstraksi Informasi dari Artikel Berita dengan Algoritma Pencocokan String

Oleh:

Annisa Ayu Pramesti

13518085

PENDAHULUAN

1. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencarian string, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977.

Perhitungan penggeseran pada algoritma 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+n-1] = pattern[0..j-1] dan a = teks[i+j] tidak sama dengan b = pattern[j] . Ketika kita menggeser, sangat beralasan bila ada sebuah awalan dari pattern akan sama dengan sebagian akhiran 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 posisi karakter pattern[j] setelah digeser, sehingga kita bisa menggeser pattern sebesar j-next[j] relatif terhadap teks. Secara sistematis, langkah-langkah yang dilakukan algoritma 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:
    1. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
    2. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

2. Kompleksitas Algoritma Knuth-Morris-Pratt

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 memory internal jika teks dibaca dari file eksternal. Semua besaran O tersebut tidak tergantung pada besarnya ruang alpabet

3. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah salah satu algoritma pencarian string, dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada tahun 1977.

Algoritma ini dianggap sebagai algoritma yang paling efisien pada aplikasi umum. Tidak seperti algoritma pencarian string yang ditemukan sebelumnya, algoritma Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritma 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 algoritma Boyer-Moore pada saat mencocokkan string adalah:

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

4. Kompleksitas Algoritma Boyer-Moore

Tabel untuk penggeseran bad-character dan good-suffix dapat dihitung dengan kompleksitas waktu dan ruang sebesar $O(n + \sigma)$ dengan σ adalah besar ruang alfabet. Sedangkan pada fase pencarian, algoritma ini membutuhkan waktu sebesar O(mn), pada

kasus terburuk, algoritma ini akan melakukan 3n pencocokkan karakter, namun pada performa terbaiknya algoritma ini hanya akan melakukan O(m/n) pencocokkan.

5. *Regular Expression*

Regular expression (disingkat regexp atau regex) adalah cara untuk menggambarkan sekumpulan karakter menggunakan aturan sintaksis. Banyak bahasa pemrograman menggunakan atau mendukung *regular expression*. *Regular expression* kemudian digunakan oleh program khusus atau bagian dari bahasa pemrograman. Program ini akan menghasilkan parser yang dapat digunakan untuk mencocokkan ekspresi atau cocok dengan ekspresi itu sendiri.

Prosesor *regular expression* digunakan untuk memproses pernyataan *regular expression* dalam hal tata bahasa dalam bahasa formal yang diberikan, dan dengan itu memeriksa string teks.

Beberapa contoh dari apa yang dapat dicocokkan dengan *regular expression*:

1. Urutan karakter "mobil" muncul berurutan dalam konteks apa pun, seperti dalam "mobil", "kartun", atau "bikarbonat"
2. Urutan karakter "mobil" yang terjadi dalam urutan itu dengan karakter lain di antara mereka, seperti dalam "Icelander" atau "kandil"
3. Kata "mobil" ketika muncul sebagai kata yang terisolasi
4. Kata "mobil" ketika didahului dengan kata "biru" atau "merah"
5. Kata "mobil" bila tidak diawali dengan kata "motor"
6. Tanda dolar segera diikuti oleh satu atau lebih digit, dan kemudian secara opsional satu periode dan tepat dua digit lebih (misalnya, "$ 10" atau "$ 245,99"). Ini tidak cocok dengan "$ 5", karena jarak antara tanda dolar dan angka, atau "€ 25", karena tidak ada tanda dolar.

IMPLEMENTASI PROGRAM

1. Spesifikasi Laptop

   Laptop yang digunakan untuk mengimplementasikan algoritma di atas adalah sebagai berikut.

Nama Laptop	: ASUS Vivobook A442U
OS	: Ubuntu 18.04
Memory	: 4GB / 8GB DDR4 2133MHz SDRAM
Processor	: Intel® Core™ i5 8250U Processor (6M Cache, up to 3.40 GHz)

2. Kode Program

   Berikut adalah implementasi algoritma yang sudah dijelaskan dalam bahasa python.

```
def get_index(word):

    idx = [0] * len(word)

    i = 1

    m = 0

    while i < len(word):

        if word[i].lower() == word[m].lower():

            m += 1

            idx[i] = m

            i += 1

        elif word[i].lower() != word[m].lower() and m != 0:

            m = idx[m-1]

        else:

            idx[i] = 0

            i += 1

    return idx
```

```
def search_keyword_kmp(data, keyword):
    ret = []
    for sentence in data:
        idx = get_index(keyword)
        i = 0
        j = 0
        while j < len(sentence):
            if keyword[i].lower() != sentence[j].lower():
                if i == 0:
                    j += 1
                else:
                    i = idx[i-1]
            else:
                i += 1
                j += 1
                if i == len(keyword):
                    ret.append(sentence)
                    break
    return ret
def init_char_value(word):
    init = [-1]*256
    for i in range(len(word)):
        init[ord(word[i])] = i;
```

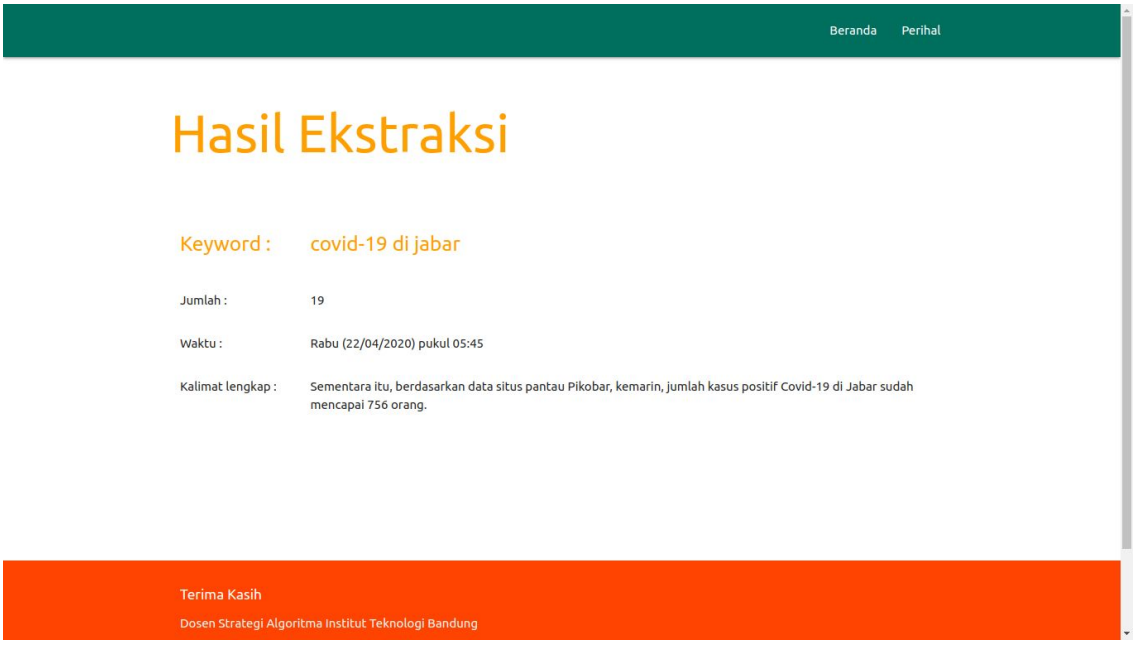
```
    return init


def search_keyword_bm(data, keyword):
    ret = []
    key_length = len(keyword)
    val = init_char_value(keyword)
    for sentence in data:
        sen_length = len(sentence)
        i = 0
        while(i <= sen_length - key_length):
            j = key_length - 1
            while j >= 0 and sentence[i+j].lower() ==
keyword[j].lower():
                j -= 1
            if j == -1:
                ret.append(sentence)
                if i + key_length < sen_length:
                    i += key_length -
val[ord(sentence[i+key_length])]
                else:
                    i += 1
            else:
                i += j - val[ord(sentence[i+j])]
                if i <= 0:
                    i = 1
```

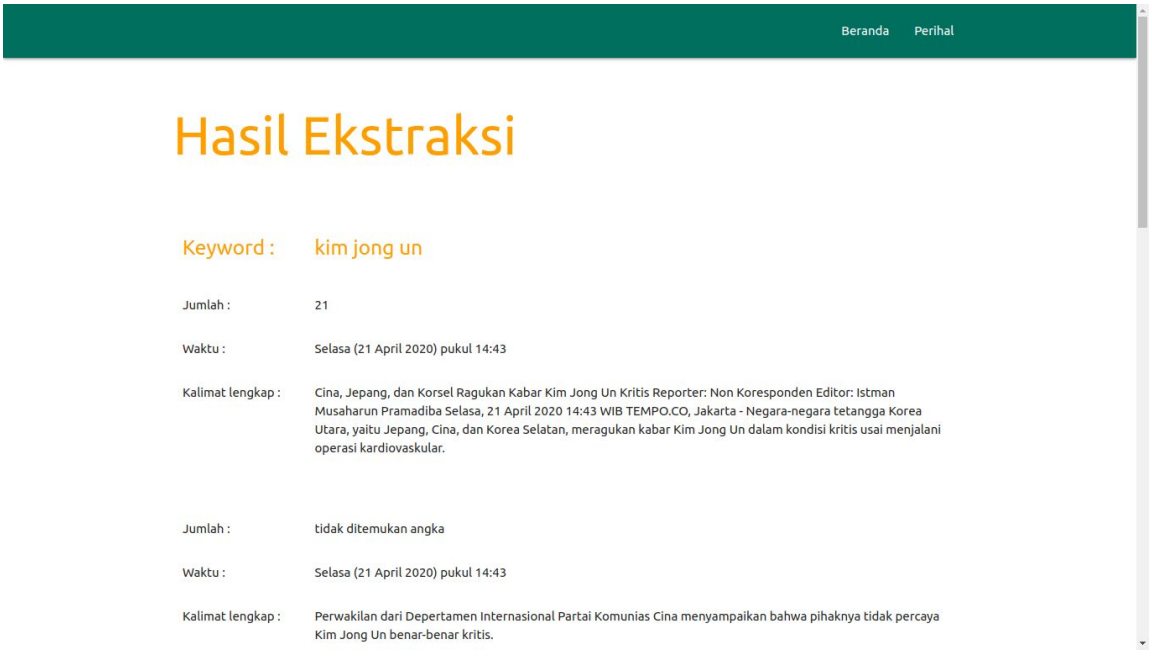
```
return ret
```

3. *Screenshot Test Cases*

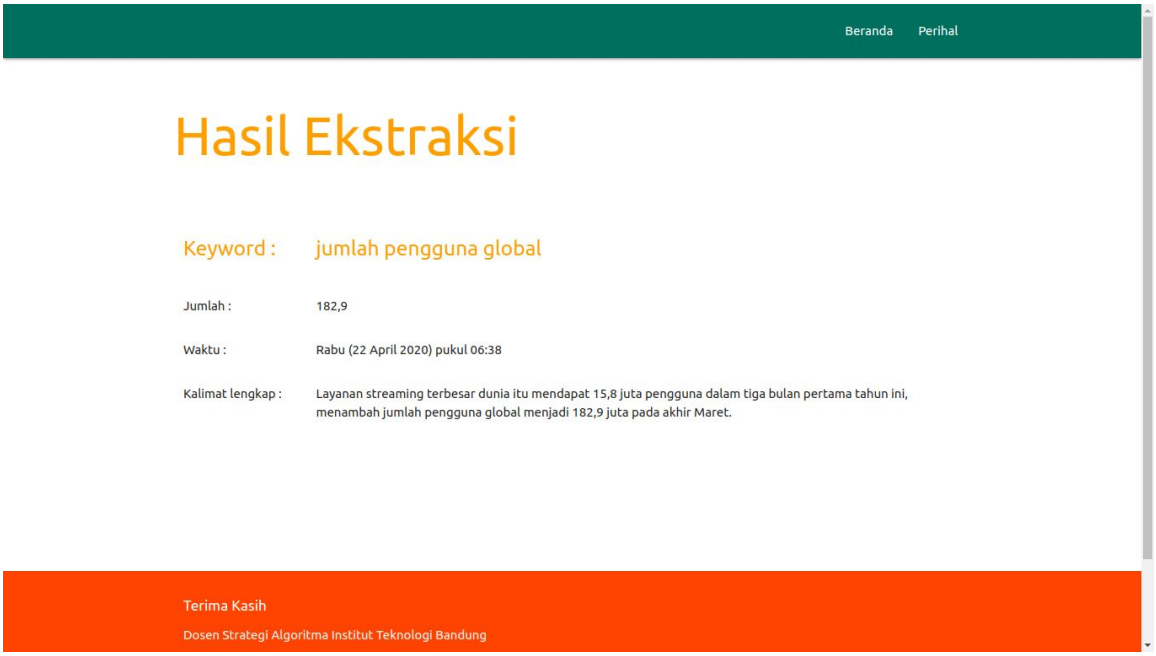
   a. Keluaran program untuk keyword = "covid-19 di jabar", teks = test1.txt



   b. Keluaran program untuk keyword = "kim jong un", teks = test5.txt



   c. Keluaran program untuk keyword = "jumlah pengguna global", teks = test8.txt



Tabel Poin

Poin	Ya	Tidak
1. Program berhasil dikompilasi	√	
2. Program berhasil running	√	
3. Program dapat menerima input dan menuliska output	√	
4. Luaran sudah benar untuk semua n	√	