Implementasi Algoritma Pencocokan String pada Fitur Filter Pesan Secara *Real-Time*

Raden Haryosatyo Wisjnunandono - 13520070
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520070@std.stei.itb.ac.id

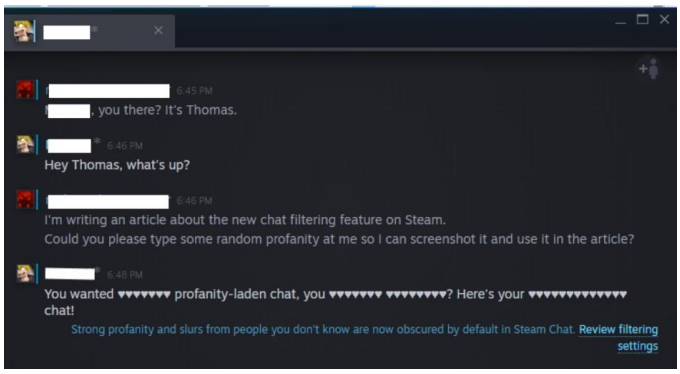
***Abstract*—Aktivitas bersosial manusia turut berevolusi seiring dengan berkembangnya teknologi, salah satunya adalah dalam aktivitas tukar menukar pesan di aplikasi pesan singkat. Selain memudahkan komunikasi tanpa perlu khawatir batasan waktu dan tempat, aplikasi pesan singkat juga memberikan kemudahan untuk menjaga anonimitas penggunanya. Sayangnya, kemudahan untuk menjaga anonimitas di aplikasi pesan singkat ini juga memudahkan seseorang dalam mengirimkan pesan yang sarat akan ujaran kebencian yang biasanya tabu untuk dibicarakan secara umum. Untuk mengatasi hal itu, beberapa aplikasi pesan singkat memiliki fitur filter pesan. Fitur filter pesan ini memungkinkan aplikasi untuk terlebih dahulu menyensor kata-kata yang dinilai tidak pantas dalam sebuah pesan sebelum nantinya dikirim kepada penerima pesan. Dalam pembuatannya, fitur filter pesan ini dapat memanfaatkan algoritma pencocokan string. Makalah ini akan mengkaji implementasi algoritma pencocokan string dalam pembuatan filter pesan secara *real time*. Dari hasil percobaan yang dilakukan, didapati bahwa algoritma pencocokan string dapat digunakan dalam pembuatan fitur fiter pesan. Lebih dari itu dari percobaan juga disimpulkan bahwa algoritma pencocokan string KMP lebih baik dalam pembuatan fitur filter pesan ketimbang algoritma BM.**

Kata kunci-Filter Pesan; Pesan Singkat; Pencocokan String

I. PENDAHULUAN

Sebagai makhluk sosial, bersosialisasi merupakan aktivitas yang esensial bagi manusia. Seiring dengan perkembangan teknologi, bentuk dari aktivitas ini pun juga turut berevolusi. Mulai dari komunkasi mulut ke mulut hingga tukar-menukar pesan di media sosial. Media sosial memungkinkan manusia untuk menjadi lebih leluasa dalam bersosialisasi tanpa perlu khawatir halangan waktu dan jarak. Media sosial juga memungkinkan manusia untuk dapat bersosialisasi tanpa perlu memberikan identitas aslinya. Singkatnya, media sosial memungkinkan manusia untuk bersosialisasi secara anonim.

Anonimitas dalam bersosialisasi melalui media sosial memungkinkan seseorang untuk menuliskan sesuatu tanpa perlu khawatir akan konsekuensinya. Ujaran kebencian, intoleransi, dan rasisme merupakan topik pembicaraan sehari-hari di media sosial. Oleh karena itu, beberapa aplikasi pesan singkat sekarang memiliki fitur filter pesan secara *real time* yang terlebih dahulu memproses pesan yang dikirimkan pengguna dan menyensor kata-kata yang dinilai tidak pantas sebelum dikirimkan kepada penerima pesan.



***Gambar 1** Ilustrasi Fitur Filter Pesan*
(Sumber: https://www.geekwire.com/2020/valve-debuts-chat-filtering-feature-steam-wait-long/)

Kemajuan teknologi yang memungkinkan terjadinya evolusi dalam bersosialisasi ini tak lepas dengan kemajuan ilmu komputasi. Kemajuan ilmu komputasi diiringi pula dengan semakin banyaknya algoritma untuk pemecahan masalah. Salah satu dari masalah yang dapat dipecahkan secara algoritmik adalah masalah pencocokan string. Umumnya pada algoritma pencocokan string diberikan sebuah teks dan sebuah pola. Kemudian, algoritma akan mencari lokasi pertama di dalam teks yang bersesuaian dengan pola.

Salah satu implementasi algoritma pencocokan string ini adalah dalam pembuatan fitur filter pesan. Makalah ini akan mencoba untuk mengaplikasikan beberapa algoritma pencocokan string dalam pembuatan fitur filter pesan dengan merepresentasikan pesan pengguna sebagai teks dan kata-kata tidak pantas yang ingin difilter sebagai pola. Harapannya, pengimplementasian algoritma pencocokan string ini dalam fitur filter pesan dapat membantu mewujudkan pengalaman bermedia sosial yang positif.

II. LANDASAN TEORI

A. *Algoritma Pencocokan String Knuth-Morris-Pratt (KMP)*

Algoritma pencocokan string Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencocokan string dengan ide utama menghindari pengulagan iterasi dari teks uji. Untuk dapat melakukan hal ini algoritma akan mempelajari pola prefix dan suffix dari pola yang akan diuji. Algoritma KMP melakukan pencarian sebuah pola dalam teks dari kiri ke kanan. Apabila terjadi *mismatch* maka penelusaran akan digeser berdasarkan *border function* dari pola yang sebelumnya telah diinisialisasikan. Dengan tidak menggeser pencarian ke elemen awal pola, jumlah pergeseran dapat diminimalisasi.

Sebelum menggunakan algoritma utama KMP perlu diinisialisasikan sebuah *border function* terlebih dahulu. Algoritma KMP melakukan praproses terhadap string pola untuk menentukan suatu *border function*. Dengan P merupakan pola uji, j merupakan posisi terjadinya *mismatch*, dan k merupakan posisi sebelum terjadinya *mismatch*, sebuah *border function b*(k) didefinisikan sebagai ukuran terbesar dari prefiks P[0..k] yang juga merupaan suffiks dari P[1..k] [1]. Contoh dari implementasi *pseudocode* dari *border function* adalah sebagai berikut.

```
function borderFunction(input pola : string) → array of integer
  KAMUS
      border: array [0..length of pola - 1] of integer
      m,i,j : integer
  ALGORITMA
      border[0] ← 0
      i ← 1
      j ← 0
      m ← length of pola
      while (i < m) do
          if pola[j] = pattern[i] then
              border[i] ← j + 1
              i ← i + 1
              j ← j + 1
          else if (j > 0) then
              j ← border[j-1]
          else
              border[i] ← 0
              i ← i + 1
    -> border
```

***Pseudocode 1** Border Function*

Adapun contoh ilustrasi dari pembuatan *border function* dari pola “abababababca” dapat dilihat pada gambar 2 di bawah ini.

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

***Gambar 2** Ilustrasi Border Function dari Pola “abababababca”*
(Sumber: https://www.geekwire.com/2020/valve-debuts-chat-filtering-feature-steam-wait-long/)

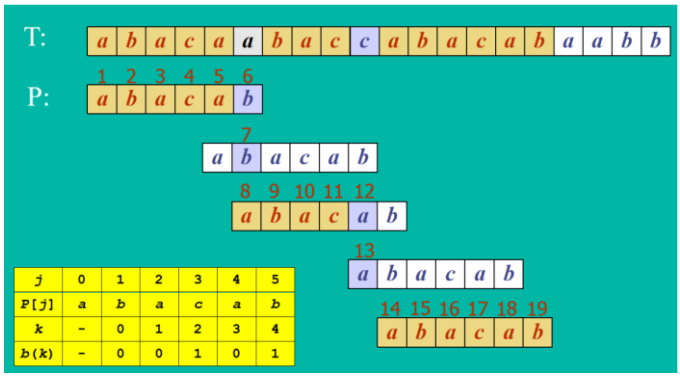
Setelah menentukan *border function* dari sebuah pola, selanjutnya fungsi tersebut dapat digunakan pada algoritma utama KMP. Algoritma KMP secara formal didefinisikan sebagai berikut. Misal sebuah pola P ingin dicari pada sebuah teks T, dengan i sebagai lokasi iterasi pencarian pada teks, b sebagai *border function* dari teks T, dan j adalah iterator pencarian pada P maka apabila T[i] != P[j] penelusran iterator j akan diulang dari b[j]. Untuk lebih lengkapnya diberikan *pseudocode* dari algoritma KMP sebagai berikut:

```
function KMP(input teks , pola : string) → integer
  KAMUS
      i,j,m,n : integer
      borderFunction : function (input : string) → array of integer
      b : array[0..m-1] of integer
  ALGORITMA
      i ← 0
      j ← 0
      n ← length of teks
      m ← length of pola
      b ← borderFunction(pola)
      while (i < n) do
          if pola[j]=teks[i] then
              if (j=m-1) then
                  -> i-m+1
              i ← i+1
              j ← j+1
          else if (j>0) then
```

```
                j ← b[j-1]
          else
                i ← i+1
    -> -1 {pola tidak terdapat pada teks}
```

Pseudocode 2 *Algoritma Utama KMP*

Adapun contoh penggunaan algoritma KMP ini pada teks T "abacaabaccabacabaabb" dan pola P "abacab" dapat dilihat pada gambar 3 di bawah ini.



Gambar 3 *Contoh Implementasi Algoritma KMP*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Makalah2021/Makalah-Stima-2021-K2%20(45).pdf)

Dapat dilihat pada pseudocode di atas bahwa algoritma pencocokan string KMP tidak melakukan pengulangan iterasi apabila terjadi *mismatch*. Hal ini dapat terwujud dengan bantuan dari *border function* yang telah diinisialisasikan terlebih dahulu. Dengan memanfaatkan *border function*, pada algoritma ini tidak memerlukan *nested looping* sehingga kompleksitas waktu dapat diminimalisasi. Dengan panjang pola *m*, algoritma pembentukkkan *boder function* memiliki kompleksitas waktu O(m) dan dengan panjang teks *n*, algoritma KMP utama memiliki kompleksitas waktu O(n). Oleh karena itu, total kompleksitas waktu algoritma KMP ini adalah O(m+n).

B. Algoritma Pencocokan String Boyer-Moore (BM)

Berbeda dengan algoritma pencocokan string KMP, algoritma penococokan string Boyer-Moore (BM) melakukan penelusuran pencocokkan dari kiri ke kanan. Algorirma BM ini didasari oleh dua teknik. Teknik pertama adalah teknik *looking-glass*. Teknik ini mencari pola pada teks dengan cara mengiterasi mundur tiap elemen yang ada di pola, dimulai dari elemen terakhirnya [1]. Teknik yang kedua adalah teknik *character-jump*. Teknik ini adalah teknik yang mengatasi kejadian *mismatch*. Saat terjadi *mismatch*, misal pada pencarian pola P pada teks T yakni pada T[i] = x dan P[j] != T[i] maka akan terjadi tiga kemungkinan:

- Kemungkinan pertama:

  Jika pola P mengandung x, maka coba untuk geser P ke kanan untuk mensejajarkan kemunculan terakhir x pada P dengan T[i].

- Kemungkinan kedua:

  Jika pola P mengandung x, tetapi maka percobaan untuk menggeser P ke kanan untuk mensejajarkan kemunculan terakhir x pada P dengan T[i] tidak memungkinkan, maka geser P ke kanan sebanyak 1 elemen ke T[i+1]

- Kemungkinan ketiga:

  Jika kemungkinan pertama dan kedua tidak memenuhi (jika x tidak terdapat pada P) maka geser P sehingga P[0] sejajar dengan T[i+1].

Serupa dengan algoritma KMP, algoritma BM juga melakukan praproses kepada pola yang ingin dicari dengan membuat fungsi pembantu yaitu *last occurrence function*. *Least occurrence function* memetakan tiap karakter A yang terdapat pada pola P kepada sebuah bilangan bulat. *Least occurrence function* didefinisikan sebagai L(x) dengan x adalah sebuah karakter pada A dan index terbesar adalah i sedemikian sehingga P[i] = x atau -1 jika tidak terdapat indeks. Untuk memperjelas, berikut dilampirkan *pseudocode* dari pembuatan *least occurrence function*.

```
function buildLast(input pola : string) → array of integer
  KAMUS
    last: array [0..127] of integer
  ALGORITMA
    i traversal [0..127] {128 ASCII Character}
      last[i] = -1 {inisialisasi array}
    i traversal [0..length of pola]
      last[pola[i]] ← i
  → last
```

Pseudocode 3 *Last Occurencec Function*

Adapun contoh ilustrasi dari penerapan *last occurrence function* pada pola "abacab" dengan x adalah "abcd" dapat dilihat pada gambar 4 di bawah ini.

x	***a***	***b***	***c***	***d***
L(x)	4	5	3	-1

Gambar 4 *Implementasi Least Occurrence Fucntion*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Makalah2021/Makalah-Stima-2021-K2%20(45).pdf)

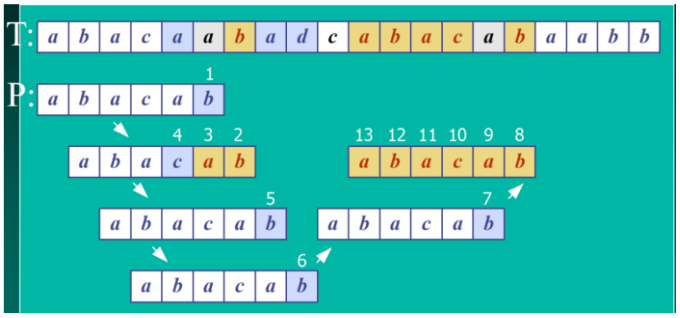
Setelah menerapkan *last occurrence function* kepada pola yang ingin dicari dalam teks, kita dapat menggunakan array hasilnya pada algoritma utama BM. Berikut dilampirkan *pseudocode* dari algoritma utama BM.

```
function BM(input teks, pola : string) → integer
  KAMUS
```

```
    n,m,i,j,lo : integer
    last : array[0..m-1] of integer
ALGORITMA
    n ← length of teks
    m ← length  of pola
    i ← m-1
    last ← buildLast(pola)
    if (i>n-1) then
        → -1 {tidak akan ditemukan jika pola lebih panjang dari teks}
    j ← m-1
    while (i<=n-1) do
        if pola[j]=teks[i] then
            if j=0 then
                → i {match}
            else {teknik looking-glass}
                i ← i – 1
                j ← j – 1
        else {teknik charater-jump}
            lo ← last[teks[i]]
            i ← i + m –min(j, 1+lo)
            j ← m-1
→ false
```

Pseudocode 4 *Algoritma Utama BM*

Adapun contoh penggunaan algoritma BM pada pencarian pola P "abacab" di dalam teks T "abacaabadcabacabaabb" dapat dilihat pada gambar 5 berikut ini.



Gambar 5 *Contoh Implementasi Algoritma BM*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Makalah2021/Makalah-Stima-2021-K2%20(45).pdf)

Dapat dilihat pada pseudocode di atas bahwa algoritma pencocokan string BM melakukan 2 kali iterasi pada saat mengeksekusi *least occurencfe function.* Lebih dari itu, BM juga melakukan iterasi sekali pada algoritma utama. Oleh karna itu, untuk panjang teks n, panjang pola m, dan banyaknya karakter unik pada pola A maka kompleksitas waktu algoritma BM adalah O(nm + A)

C. Filter Pesan

Seiring dengan kemajuan zaman, beberapa platform penyedia layanan pesan singkat mulai menyadari bahwa bahaya dapat muncul dalam bentuk pesan yang tidak pantas atau menyakitkan [2]. Oleh karena itu dibuatlah fitur penyensoran kata-kata tertentu atau yang biasa disebut fitur filter pesan. Fitur ini memproses tiap masukan pesan dari pengguna untuk kemudian mengganti kata-kata yang mengandung unsur kebencian dan hinaan dengan karakter-karakter netral seperti ****. Fitur filter pesan ini biasanya muncul pada platform permainan daring yang mengharuskan program untuk melakukan penyaringan secara *real-time* karena komunikasi yang dilakukan para pemain dalam permainan daring tersebut juga dilakukan secara *real-time*. Contoh dari penggunaan fitur filter pesan ini dapat dilihat pada gambar 1.

III. IMPLEMENTASI ALGORITMA PENCOCOKAN STRING DALAM PROGRAM FILTER PESAN

Dalam mengimplementasikan algoritma-algoritma yang sudah ditulis pada bagian sebelumnya, penulis menggunakan bahasa pemrograman Python. Garis besar penulisan program dibuat sama dengan yang telah dituliskan pada pseudocode 1-4 dengan beberapa perubahan untuk mengitkuti standar sintaks dari bahasa pemrograman Python.

Selain mengimplementasikan pseudocode 1-4, dalam membuat filter pesan sederhana juga diperlukan sebuah *driver* dan *main program* untuk mengetes algoritma-algoritma yang telah dibuat. Adapun driver dan main program yang dibuat dapat dilihat pada gambar 6 dan 7 berikut.

```python
def driverprogram(texts, patterns, method, isPhrase):
    if not isPhrase:
        texts = texts.split()
    else:
        texts = [texts]

    starttime = time.time()
    for text in texts:
        toPrint = []
        for elem in text:
            toPrint.append(elem)
        for pattern in patterns:
            idx = -1
            if method == "kmp":
                idx = kmp(pattern.lower(), text)
            elif method == "bm":
                idx = BM(pattern.lower(), text)
            else:
                print("method tidak valid")
                return 0
            if idx != -1:
                for jix in range(len(pattern)):
                    if pattern[jix] != " ":
                        toPrint[idx+jix] = "*"
```

```
    for element in toPrint:
        print(element, end="")
    print(" ", end="")
endtime = time.time()
print()
time = (endtime - starttime)*1000
print("elapsed time:", time, "ms")
return time
```

Gambar 6 *Snippet Implementasi Driver pada Program*
(Sumber: gubahan pribadi penulis)

```
userMethod = input("masukkan method: ")
userMethod = userMethod.lower()
if userMethod == "kmp":
    userMethod = "kmp"
else:
    userMethod = "bm"
searchPhrase = input("apakah ingin mengecek frasa? (Y/N)
: ")
if searchPhrase == "Y":
    isPhrase = True
else:
    isPhrase = False
userText = input("masukkan text: ")
userText = userText.lower()
patterns = csv_reader("words.csv")

driverprogram(userText, patterns, userMethod, isPhrase)
```

Gambar 7 *Snippet Implementasi Main Program*
(Sumber: gubahan pribadi penulis)

Pada gambar 7 terlihat bahwa program juga dapat diatur untuk pengaturan pemeriksaan frasa. Pemeriksaan frasa dalam hal ini ialah memeriksa keberadaan pola yang berbentuk frasa (dipisahkan oleh spasi) dalam teks. Pembedaan penanganan kasus untuk frasa dan kata ini disebabkan oleh adanya karakter spasi pada frasa. Untuk kasus penanganan kata, mula-mula teks inputan pengguna akan diubah menjadi sebuah larik yang elemennya merupakan kata-kata yang terdapat dalam teks masukkan pengguna. Sementara itu, untuk penanganan kasus frasa, teks masukkan pengguna akan langsung diproses dalam algoritma KMP/BM dan dijadikan parameter teks pada fungsi yang mengimplementasikan algoritma tersebut.

Fungsi *driver* pada program menerima larik yang berisi kumpulan kata-kata yang tidak pantas. Selanjutnya pada fungsi *driver*, akan dilakukan iterasi pada tiap elemen larik kumpulan kata-kata yang tidak pantas ini sehingga penggunakan fungsi algoritma KMP/BM dilakukan pada tiap iterasi dari larik kata-kata tidak pantas ini. Adapun kumpulan kata-kata tidak pantas yang ingin disensor ini didapat dari https://github.com/okkyibrohim/id-multi-label-hate-speech-and-abusive-language-detection. Bagian yang dimanfaatkan oleh makalah ini dari pranala tersebut ialah file abusive.csv. Untuk memproses file csv ini diperlukan sebuah fungsi csv reader. Pada program implementasi ini, fungsi csv reader dapat dilihat pada gambar 8 berikut ini.

```
def csv_reader(file_name):
    filename = file_name
    f = open(filename, "r")
    length = len(f.readlines())
    f.close()
    z = []
    for l in range(length):
        f = open(filename, "r")
        a = f.readlines()[l]
        c = 0
        for i in a:
            if i == ';':
                c += 1
        d = 1
        start = 0
        while (d <= c):
            for i in range(len(a)):
                if a[i] == ";":
                    stop = i
                    z.append(a[start:stop])
                    start = i + 1
                    d += 1
        if l == length - 1:
            z.append(a[start:(len(a))])
        else:
            z.append(a[start:(len(a) - 1)])

        f.close()
    return z
```

Gambar 8 *Snippet Implementasi CSV Reader pada Program*
(Sumber: gubahan pribadi penulis)

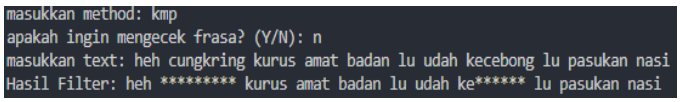
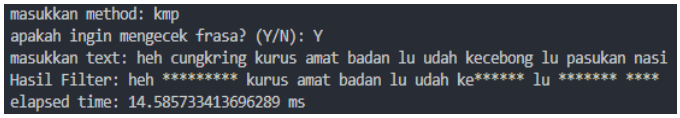
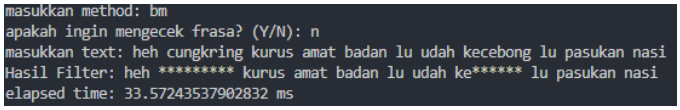
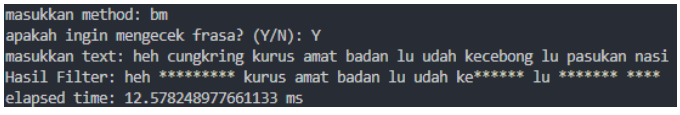
Setelah melalui *csv reader* kemudian kita dapat kata-kata yang ingin disensor. Kata-kata yang ingin difilter dalam program ini adalah sebagai berikut

['ABUSIVE', 'alay', 'ampas', 'buta', 'keparat', 'anjing', 'anjir', 'babi', 'bacot', 'bajingan', 'banci', 'bandot', 'buaya', 'bangkai', 'bangsat', 'bego', 'bejat',
'bencong', 'berak', 'bisu', 'celeng', 'jancuk', 'bodoh', 'berengsek', 'budek', 'burik', 'jamban', 'cocot', 'congor', 'culun', 'cupu', 'dongok', 'dungu', 'edan', 'tai', 'ngewe', 'geblek', 'gembel', 'gila', 'goblok', 'iblis', 'idiot', 'jablay', 'jembud', 'jembut', 'jijik', 'kacrut', 'kafir', 'modar', 'kampang', 'kampret', 'kampungan', 'kimak', 'kontol', 'kunti', 'tuyul', 'kunyuk', 'mampus', 'memek', 'monyet', 'najis', 'nete', 'ngentot', 'noob', 'pecun', 'perek', 'sampah', 'sarap', 'setan', 'silit', 'bokong', 'sinting', 'sompret', 'sontoloyo', 'terkutuk', 'titit', 'pantat', 'tolol', 'udik', 'antek', 'asing', 'ateis', 'sitip', 'autis', 'picek', 'ayam kampus', 'bani kotak', 'bispak', 'bisyar', 'bokep', 'bong', 'cacat', 'cct', 'cebong', 'taplak', 'cungkring', 'gay', 'gembrot', 'gendut', 'hina', 'homo', 'komunis', 'koreng', 'krempeng', 'lengser', 'lesbi', 'lgbt', 'lonte', 'mucikari', 'munafik', 'ngaceng', 'nista', 'kejam', 'onta', 'panastak', 'panasbung', 'bani', 'pasukan nasi', 'porno', 'seks', 'rejim', 'rezim', 'sange', 'serbet', 'sipit', 'transgender']

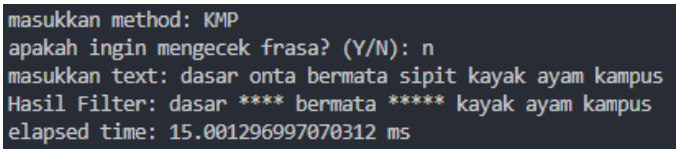
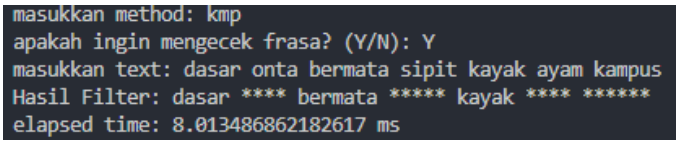
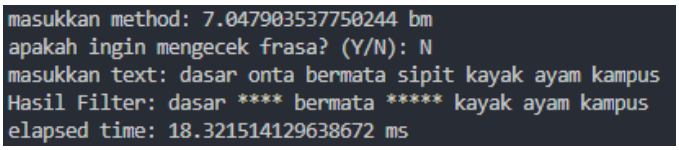
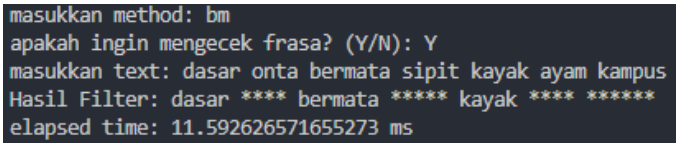
IV. PENGUJIAN, HASIL, DAN PEMBAHASAN

Untuk menguji efektivitas masing-masing algoritma dalam memfilter kata-kata yang diinginkan maka dalam program ini akan diberikan 3 buah teks percobaan. Prakondisi dari tes ini ialah masing-masing teks memiliki pola kata dan frasa yang

ingin difilter. Hasil dari pengujian ini ialah waktu yang dibutuhkan untuk memfilter dan mengganti kata-kata yang diinginkan. Untuk masing-masing teks akan dilakukan 4 macam pengujian yakni untuk kasus frasa menggunakan BM, frasa menggunakan KMP, kata menggunakan BM, dan kata menggunakan KMP. Untuk meminimalisasi efek perangkat keras dalam pengjuian, maka akan dilakukan sebanyak 20 kali pengujian untuk masing-masing jenis pengujian. Hasil dari pengujian dapat dilihat pada tabel 1-3 sebagai berikut. (Untuk menghemat tempat pada bagian tangkapan layer pengujian hanya akan diberikan gambar pada pengujian iterasi pertama)

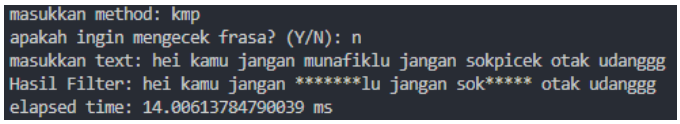
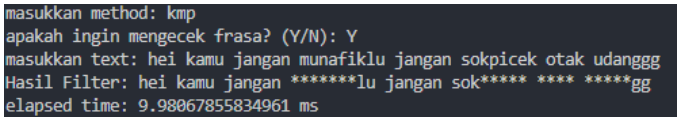
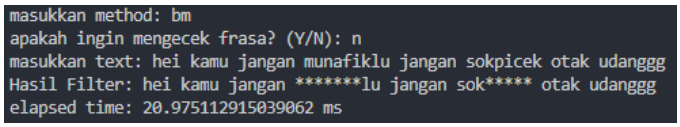
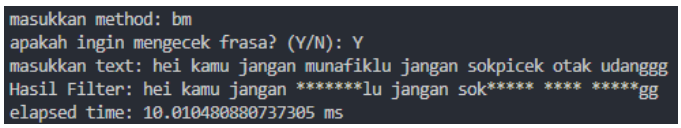
Teks Uji	heh cungkring kurus amat badan lu udah kecebong lu pasukan nasi
Kata yang ingin difilter	Cungkring, kecebong
Frasa yang ingin difilter	Pasukan nasi
Pengujian Filter Kata Menggunakan KMP	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 16.724002361297607 ms	
Pengujian Filter Kata dan Frasa Menggunakan KMP	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 9.72607135772705 ms	
Pengujian Filter Kata Menggunakan BM	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 29.827332496643066 ms	
Pengujian Filter Kata dan Frasa Menggunakan BM	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 11.528027057647705 ms	

Tabel 1 *Pengujian Filter Kata Teks 1*
(Sumber: gubahan pribadi penulis)

Teks Uji	dasar onta bermata sipit kayak ayam kampus
Kata yang ingin difilter	Onta, sipit
Frasa yang ingin difilter	Ayam kampus
Pengujian Filter Kata Menggunakan KMP	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 10.499215126037598 ms	
Pengujian Filter Kata dan Frasa Menggunakan KMP	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 7.047903537750244 ms	
Pengujian Filter Kata Menggunakan BM	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 17.815053462982178 ms	
Pengujian Filter Kata dan Frasa Menggunakan BM	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 8.778667449951172 ms	

Tabel 2 *Pengujian Filter Kata Teks 2*
(Sumber: gubahan pribadi penulis)

Teks Uji	hei kamu jangan munafiklu jangan sokpiceklu otak udanggg
Kata yang ingin difilter	Munafik, picek

Frasa yang ingin difilter	Otak udang
Pengujian Filter Kata Menggunakan KMP	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 14.048147201538086 ms	
Pengujian Filter Kata dan Frasa Menggunakan KMP	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 8.66483449935913 ms	
Pengujian Filter Kata Menggunakan BM	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 22.415149211883545 ms	
Pengujian Filter Kata dan Frasa Menggunakan BM	
Tangkapan layar pengujian pertama	
	
Waktu rata-rata 20 pengujian: 10.451650619506836 ms	

Tabel 3 *Pengujian Filter Kata Teks 3*
(Sumber: gubahan pribadi penulis)

Dari ketiga teks yang diuji dibuktikan bahwa fitur filter pesan secaara *real time* dapat dibuat menggunakan algoritma pencocokan string KMP dan Boyer-Moore. Dari pengujian juga didapati bahwa pemfilteran menggunakan algoritma KMP lebih cepat dibanding BM. Hal ini selaras dengan dasar teori yang menyatakan bahwa kompleksitas waktu KMP adalah O(n + m) dan kompleksitas waktu BM adalah O(mn + A) jadi sudah sewajarnya waktu eksekusi program lebih singkat jika menggunakan algoritma KMP. Cepatnya algoritma KMP ini juga didukung oleh fakta bahwa kebanyakan teks yang diujikan (dan tentunya juga yang ada di dunia nyata) mengandung pola yang dimulai dari awal tek. Hal ini menyebabkan meskipun di dalam teks terdapat prefiks atau suffiks yang mendahului maupun mengakhiri teks, tetapi hal tersebut tidak terlalu berpengaruh karena panjang prefiks/suffiks tersebut jauh lebih pendek ketimbang panjang pola. Hal ini menyebabkan minimnya pergeseran yang harus dilakukan selama penelusuran menggunakan algoritma pencocokan string.

V. KESIMPULAN DAN SARAN

Dari data percobaan yang dilakukan dapat disimpulkan bahwa algoritma pencocokan string KMP dan Boyer-Moore dapat digunakan dalam pembuatan fitur filter pesan secara *real-time*. Jika dibandingkan antara algoritma KMP dan Boyer-Moore maka algoritma KMP lebih unggul dalam pengaplikasian fitur filter pesan ini. Dari pengujian juga diketahui pemfilteran yang meliputi frasa jauh lebih cepat dibanding pemfilteran yang eksklusif hanya kata saja. Hal ini disebabkan oleh pengubahan teks input menjadi larik kata per kata di pemfilteran kata sehingga dibutuhkan *nested looping* dalam pengecekannya.

Saran dari penulis adalah supaya mengembangkan program yang garis besarnya sudah dilampirkan pada makalah ini dengan memperbanyak kasus pola uji. Lebih dari itu, penulis berharap makalah ini dapat menjadi referensi pembaca dalam pembuatan fitur filter pesan apabila nantinya pembaca ingin membuat aplikasi pesan singkat agar pengalaman pengguna dalam bersosialisasi menjadi menyenangkan.

PRANALA VIDEO YOUTUBE

Video penjelasan lebih lanjut mengenai makalah ini dapat diakses melalui pranala berikut:

https://youtu.be/AcOfeK7_pFY

UCAPAN TERIMAKASIH

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat mengerjakan makalah ini. Kemudian, tidak lupa penulis juga mengucapkan terima kasih kepada. Bapak Rinaldi Munir, Ibu Masayu Leylia Khodra, dan Ibu Ulfa Nur Maulidevi selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma yang telah bersedia membagikan ilmunya sehingga saya dapat mengerjakan makalah ini dengan lancar. Tidak lupa penulis juga mengucapkan terima kasih sebesar-besarnya kepada keluarga dan teman-teman penulis yang telah mendukung secara penuh dalam proses pembelajaran dan pengerjaan makalah ini.

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. Diakses pada 21 Mei 2022.

[2] https://www.slashgear.com/steam-chat-filtering-lets-you-decide-which-words-to-bleep-out-08641509. Diakses pada 21 Mei 2022.

[3] Andrew Davidson. 2006. “Pattern Matching” dalam Munir, Rinaldi (E.d.). http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Pencocokan%20String%20(2015).ppt (diakses pada 23 Mei 2022)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 23 Mei 2022

Raden Haryosatyo Wisjnunandono 13520070