

LAPORAN
TUGAS BESAR III
DETEKSI SPAM PADA MEDIA SOSIAL ATAU *CHAT-MESSENGER*
DENGAN ALGORITMA PENCOCOKAN STRING
IF2211 STRATEGI ALGORITMA
SEMESTER II 2017-2018



Disusun oleh:

Rahmat Nur Ibrahim Santosa	13516009
Haifa Fadhila Ilma	13516076
Rinda Nur Hafizha	13516151

BAB I

A. Deskripsi Tugas



Electronic-spam merupakan pesan elektronik yang tidak diinginkan penerimanya, bisa dalam bentuk surat elektronik, SMS, *posting* atau komentar di media sosial yang muncul di *timeline* kita, ataupun pesan pada *chat-messenger*. *Spammer* melakukan *spamming* untuk tujuan tertentu, paling banyak untuk menyebarkan iklan. Penentuan *spam* sangat bersifat subjektif, artinya *spam* untuk kita, belum tentu *spam* untuk pengguna lain. Gambar 1-2 merupakan contoh *spam* pada *twitter*, *facebook*, dan *line*.

- (a) 
- (b) 

<https://econsultancy.com/blog/4218-the-seven-twitter-sins-a-comprehensive-list-of-twitter-spam-techniques>

Gambar 1. Contoh spam pada sosial media *twitter*:

(a) hashtag/trending topic spam; (b) autoresponder spam



<https://m.facebook.com/security/photos/a.36604031885.58084.31987371885/36604061885/?type=3&source=43>

Gambar 2. Contoh *spam* pada media sosial *facebook*

Algoritma pencocokan string (*pattern*) Knuth-Morris-Pratt (KMP) dan Algoritma Boyer-Moore merupakan algoritma yang lebih baik daripada *brute force*. Pada Tugas Besar III kali ini diminta membuat aplikasi sederhana deteksi *spam* pada media sosial dengan kedua algoritma tersebut, plus menggunakan *regular expression (regex)*. Teks yang akan Anda proses adalah *posting* berbahasa Indonesia. Pengguna aplikasi ini akan memberikan masukan berupa *keyword spam*, dan menandai daftar *posting* yang dikategorikan *spam* menurut berdasarkan tanggal.

Pencocokan string yang dibuat adalah *exact matching* (untuk KMP dan BM) jadi *posting* yang diproses mengandung string yang tepat sama dengan keyword spam dari pengguna. Sedangkan bila menggunakan *regex* maka tidak selalu *exact matching*. Pencarian juga tidak bersifat *case sensitive*, jadi huruf besar dan huruf kecil dianggap sama.

BAB 2

DASAR TEORI

A. Algoritma Pencarian String (String Matching)

Pencarian *string* di dalam teks disebut juga pencocokan string (*string matching* atau *pattern matching*). Persoalan pencarian *string* dirumuskan sebagai berikut:

Diberikan:

1. Teks (*text*), yaitu (*long*) *string* yang panjangnya n karakter
2. *Pattern*, yaitu *string* dengan panjang m karakter ($m < n$) yang akan dicari di dalam teks.

Lalu persoalannya adalah mencari lokasi pertama di dalam teks yang bersesuaian dengan *pattern*. Kita mengasumsikan teks berada di dalam memori. Jika *pattern* muncul lebih dari sekali di dalam teks, maka pencarian hanya memberikan keluaran berupa lokasi *pattern* ditemukan pertama kali. *Pattern* yang berada di dalam teks kita namakan target.

B. Algoritma Knuth-Morris-Pratt (KMP)

Bila pada algoritma *brute force* ditemukan ketidakcocokan *pattern* dengan teks, maka *pattern* digeser satu karakter ke kanan, maka pada algoritma KMP kita memelihara informasi yang digunakan untuk melakukan jumlah pergeseran. Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya per satu karakter, sehingga waktu pencarian dapat dikurangi secara signifikan. Algoritma ini mencari *pattern* pada teks dari kiri ke kanan, seperti algoritma *brute force*.

Misalkan A adalah alfabet dan $x = x_1x_2\dots x_k$, $k \in \mathbf{N}$, adalah *string* yang panjangnya k yang dibentuk dari karakter-karakter di dalam alfabet A .

Awalan (*prefix*) dari x adalah upa-string (*substring*) u dengan

$$u = x_1x_2\dots x_{k-1}, k \in \{1, 2, \dots, k-1\}$$

Dengan kata lain, x diawali dengan u .

Akhiran (*suffix*) dari x adalah upa-string (*substring*) u dengan

$$u = x_{k-b}x_{k-b+1}\dots x_k$$

$$, k \in \{1, 2, \dots, k-1\}$$

Dengan kata lain, x diakhiri dengan v .

Awalan u dari x atau akhiran u dari x disebut awalan atau akhiran sebenarnya (*proper*) jika $u \neq x$, yaitu panjangnya, b , lebih kecil daripada k .

Pinggiran (*border*) dari x adalah upa-string r sedemikian sehingga

$$r = x_1x_2\dots x_{k-1} \text{ dan } r = x_{k-b}x_{k-b+1}\dots x_k, \mathbf{k} \in \{1, 2, \dots, k-1\}$$

Dengan kata lain, pinggiran dari x adalah upa-string yang keduanya awalan dan juga akhiran sebenarnya dari x .

Algoritma ini melakukan proses awal (*preprocessing*) terhadap *pattern* P dengan menghitung **fungsi pinggiran** (beberapa literatur menyebutnya fungsi *overlap*, fungsi *failure*, fungsi awalan, dsb) yang mengindikasikan pergeseran s terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian *string*. Dengan adanya fungsi pinggiran ini, dapat dicegah pergeseran yang tidak berguna, seperti halnya pada algoritma *brute force*.

Fungsi pinggiran hanya bergantung pada karakter-karakter di dalam *pattern*, dan bukan pada karakter-karakter di dalam teks. Oleh karena itu, kita dapat melakukan perhitungan fungsi awalan sebelum pencarian *string* dilakukan.

Fungsi *pinggiran* $b(j)$ didefinisikan sebagai ukuran awalan terpanjang dari P yang merupakan akhiran dari $P[1..j]$

C. Algoritma Boyer-Moore

Boyer-Moore String Matching merupakan sebuah cara untuk mencocokkan sebuah String dari kanan ke kiri. Sebuah teks dicocokkan dengan pattern tertentu untuk menentukan apakah dalam teks yang dicocokkan terdapat pattern tersebut.

Perbedaan pencocokan String Boyer-Moore dengan pencocokan String secara Brute Force adalah pada Algoritma Boyer-Moore tidak semua String dicocokkan seperti pada cara Brute Force. Ketika Ada text dan pattern yang terjadi Mismatch, maka pattern akan mencocokkan text meloncat menurut nilai pada tabel delta atau sebanyak jumlah karakter yang telah dicocokkan. Hal ini tergantung nilai maksimum yang terdapat dari keduanya.

Langkah-langkah algoritma Boyer-Moore :

1. Buat tabel pergeseran string yang dicari (S) dengan pendekatan Match Heuristic (MH) dan Occurrence Heuristic (OH), untuk menentukan jumlah pergeseran yang akan dilakukan jika mendapat karakter tidak cocok pada proses pencocokan dengan string (T).
2. Jika dalam proses perbandingan terjadi ketidakcocokan antara pasangan karakter pada S dan karakter pada T, pergeseran dilakukan dengan memilih salah satu nilai pergeseran dari dua tabel analisa string, yang memiliki nilai pergeseran paling besar.
3. Dua kemungkinan penyelesaian dalam melakukan pergeseran S, jika sebelumnya belum ada karakter yang cocok adalah dengan melihat nilai pergeseran hanya pada tabel occurrence heuristic : Jika karakter yang tidak cocok tidak ada pada S maka pergeseran adalah sebanyak jumlah karakter pada S. dan jika karakter yang tidak cocok ada pada S, maka banyaknya pergeseran bergantung dari nilai pada tabel.
4. Jika karakter pada teks yang sedang dibandingkan cocok dengan karakter pada S, maka posisi karakter pada S dan T diturunkan sebanyak 1 posisi, kemudian lanjutkan dengan pencocokan pada posisi tersebut dan seterusnya. Jika kemudian terjadi ketidakcocokan karakter S dan T, maka pilih nilai pergeseran terbesar dari dua tabel analisa pattern yaitu nilai dari tabel match heuristic dan nilai tabel occurrence heuristic dikurangi dengan jumlah karakter yang telah cocok.
5. Jika semua karakter telah cocok, artinya S telah ditemukan di dalam T, selanjutnya geser pattern sebesar 1 karakter.
6. Lanjutkan sampai akhir string T.

D. *Regular Expression*

Regular Expression, atau sering disebut juga Regex/Regexp, dan juga sering disebut *pattern*, dalam teori bahasa formal, adalah deretan karakter spesial yang mendefinisikan sebuah *pattern* dalam pencarian teks. Regex menyederhanakan pencarian *pattern* dalam suatu teks. Regex digunakan pada *search engine*, *search and replace* pada *word processors* dan teks editor. Banyak bahasa pemrograman menyediakan kemampuan regex yang terpasang tetap pada bahasa tersebut atau melalui *library*.

Susunan kata regex sering digunakan untuk maksud sintaks tekstual standar untuk mewakili *pattern* untuk pencocokan teks. Tiap karakter pada regex adalah metakarakter atau karakter reguler yang mempunyai arti secara harfiah. Contohnya, pada regex *a . , a* adalah

karakter literal yang cocok dengan ‘a’ dan . adalah metakarakter yang cocok pada tiap karakter kecuali *newline*. Metakarakter dan karakter literal bisa digunakan untuk mengidentifikasi teks dari *pattern* yang diberikan, atau memproses sejumlah instansinya.

E. *Spam*

Spam adalah penggunaan perangkat elektronik untuk mengirimkan pesan secara bertubi-tubi tanpa dikehendaki oleh penerimanya. Orang yang melakukan *spam* disebut *spammer*. Tindakan *spam* dikenal dengan nama *spamming*.

Bentuk *spam* yang dikenal secara umum meliputi : *spam* surat elektronik, *spam* pesan instan, *spam* jejaring sosial, dan lain-lain. Sebagian besar *spammer* melakukan *spamming* dengan tujuan promosi atau mengiklankan produk atau jasanya, diikuti dengan tujuan penipuan dan tujuan lain-lain.

Pada tugas besar ini, *spam* digunakan sebagai *pattern* pada pencarian di dalam teks menggunakan ketiga cara pencarian yang telah disebutkan. Penentuan kata yang tergolong *spam* merupakan input pengguna.

F. *Twitter*

Twitter adalah layanan jejaring sosial dan mikroblog daring yang memungkinkan penggunaanya untuk mengirim dan membaca pesan berbasis teks hingga 140 karakter, yang dikenal dengan sebutan kicauan (*tweet*). Sejak diluncurkan, Twitter telah menjadi salah satu dari sepuluh situs yang paling sering dikunjungi di Internet, dan dijuluki dengan “pesan singkat dari Internet.”

Tingginya popularitas Twitter menyebabkan layanan ini telah dimanfaatkan untuk berbagai keperluan dalam berbagai aspek, misalnya sebagai sarana protes, kampanye politik, sarana pembelajaran, dan sebagai media komunikasi darurat. Karena popularitasnya ini, Twitter dihadapkan pada berbagai masalah dan kontroversi, salah satunya adalah seorang/sekelompok orang yang melakukan *spamming* untuk tujuan tertentu.

Pada tugas besar ini, pencarian *spam* yang dilakukan oleh salah satu dari tiga algoritma di atas akan dilakukan pada media sosial ini, di mana akan dievaluasi per-*tweet*-nya apakah *tweet* tersebut tergolong spam atau bukan.

G. *Python*

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. *Python* diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. *Python* juga didukung oleh komunitas yang besar.

Pada tugas besar ini, algoritma KMP, Boyer-Moore, dan *Regular Expression* yang digunakan untuk deteksi *spam* pada media sosial diimplementasikan menggunakan bahasa *Python*. Pada bahasa ini sudah tersedia *library* untuk regex, yaitu *re*.

H. *PHP*

PHP (*Hypertext Preprocessor*) adalah bahasa skrip yang dapat ditanamkan atau disisipkan ke dalam HTML. *PHP* banyak dipakai untuk memprogram situs web dinamis. Pada tugas besar ini, program yang kami buat merupakan aplikasi berbasis web dan kami menggunakan *PHP* sebagai kakas pengembangan programnya karena Web Server yang mendukung *PHP* dapat ditemukan di mana-mana dari mulai *apache*, *IIS*, dan lainnya. Selain itu juga karena lebih mudah dalam sisi pengembangannya, dan *PHP* adalah bahasa open source yang dapat digunakan di berbagai mesin dan dapat dijalankan secara runtime.

BAB 3

ANALISIS PEMECAHAN MASALAH

Masalah yang akan diselesaikan merupakan masalah pencarian *pattern*, dalam hal ini adalah *spam*, pada teks, dalam hal ini adalah pada *tweet*, yang dapat diselesaikan dengan metode pencarian algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer Moore, atau dengan Regular Expression (regex). Dari masukan pengguna yang diterima melalui aplikasi web program ini, akan dihasilkan daftar *tweet* dilengkapi dengan penanda apakah *tweet* tersebut tergolong *spam* atau bukan.

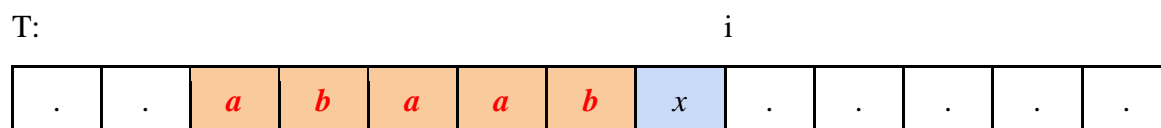
Masukan pengguna berupa *username* pengguna *twitter* yang akan digunakan untuk melihat *timeline* dari user tersebut dan diambil *tweet*-nya, yang berperan sebagai teks dalam pencarian *pattern*. Lalu, terdapat masukan pilihan metode pencarian yang digunakan untuk mencari *pattern* pada teks yaitu di antara ketiga metode yang telah disebutkan. Yang terakhir, masukan yang diterima adalah *keyword spam* yang berperan sebagai *pattern*.

Dari masukan tersebut, *pattern* dicari pada *tweet* user yang memiliki *username* hasil masukan menggunakan metode pencarian yang dipilih. Setelah itu, hasil pencarian akan ditampilkan pada aplikasi web program di mana akan ditampilkan semua *tweet* yang ada di *timeline user* tersebut dan *tweet* yang tergolong sebagai *spam* akan diberi warna merah pada tampilan tweetnya.

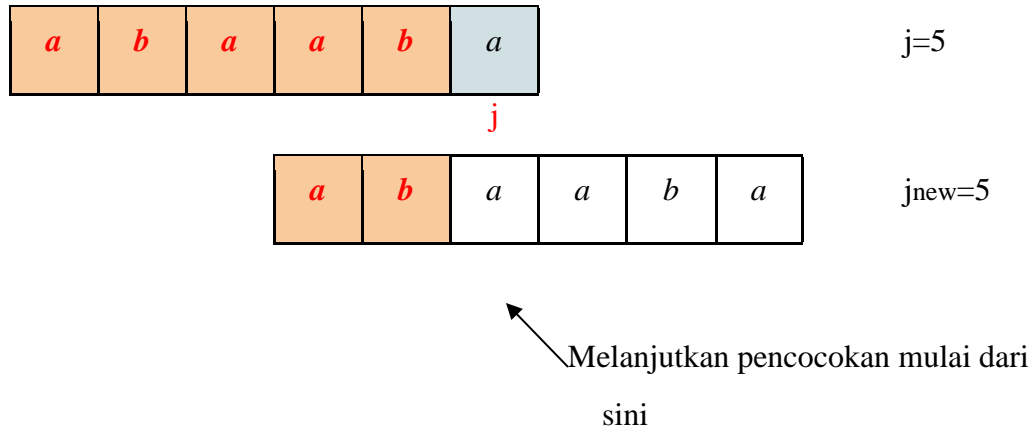
1. Langkah pencarian string menggunakan algoritma KMP

Algoritma ini mencari *pattern* pada teks dengan urutan dari kiri ke kanan. Bila terdapat ketidakcocokan di antara teks T dengan *pattern* P pada $P[j]$, misalnya $T[i] \neq P[j]$, algoritma akan menggeser ke kanan paling banyak yaitu prefiks terbesar dari $P[0 .. j-1]$ yang merupakan suffiks dari $P[1 .. j-1]$

Ilustrasi:



P:



1. Cari prefiks terbesar dari

“**a**baab” (P[0..4])

Yang merupakan suffiks dari

“aba**a**b” (P[1..4])

2. Jawaban no 1 adalah “ab” yang memiliki panjang 2
3. Set j menjadi 2. Ini adalah j baru untuk memulai pencocokan
4. Jumlah pergeseran:

$$S = \text{panjang}(\text{abbab}) - \text{panjang}(\text{ab}) = 5 - 2 = 3$$

Fungsi pinggiran $b(k)$ didefinisikan sebagai ukuran dari prefiks terbesar dari $P[0..k]$ yang juga merupakan suffiks dari $P[1..k]$.

j = posisi tidak cocok pada $P[]$

k = posisi sebelum tidak cocok ($k=j-1$)

Contoh

$P = \text{abaaba}$

$j = 012345$

j	0	1	2	3	4	5
P[j]	a	b	a	a	b	a
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

Dengan adanya fungsi pembatas ini, maka

- Bila terjadi ketidakcocokan pada $P[j]$, maka
 $k = j-1$
 $j = b(k)$

Didapatkan nilai j baru

2. Langkah pencarian string menggunakan algoritma BM

Dalam implementasinya, algoritma Boyer-Moore akan memilih nilai pergeseran terbesar dari 2 tabel pergeseran (Occurence Heuristic dan Match Heuristic). Berikut adalah contoh implementasi algoritma Boyer-Moore:

posisi :	1	2	3	4	5	6	7	8
string :	a	n	p	a	n	m	a	n
pergeseran (OH) :	1	0	5	1	0	2	1	0
pergeseran (MH) :	6	6	6	6	6	3	8	1

teks : berikut ini anpamman bukan anpanman

pada tabel OH, \rightarrow selain karakter “a”, “n”, “p”, “m” nilai pergeseran sebesar panjang string yaitu 8

tahap 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b	e	r	i	k	u	r		i	n	i		a	n	p	a	m	m	a	n		b	u	k	a	n		a	n	p	a	m	a	n	
a	n	p	a	n	m	a	n																											

spasi tidak cocok dengan “n”

-tabel OH : karakter spasi nilai pergeserannya = 8 belum ada karakter yang cocok

-tabel MH : ketidakcocokan pada posisi 8 (karakter “n”) nilai pergeserannya = 1

-sehingga geser string sebesar 8 posisi (nilai maksimal dari kedua tabel pergeseran)

tahap 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b	e	r	i	k	u	r		i	n	i		a	n	p	a	m	m	a	n		b	u	k	a	n		a	n	p	a	m	a	n	
								a	n	p	a	n	m	a	n																			

-”a” tidak cocok dengan “n”

-tabel OH : karakter “a” nilai pergeserannya = 1 belum ada karakter yang cocok

- tabel MH : ketidakcocokan pada posisi 8 (karakter “n”) nilai pergeserannya = 1
- sehingga geser string sebesar 1 posisi (nilai maksimal dari kedua tabel pergeseran)

tahap 3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b	e	r	i	k	u	t		i	n	i		a	n	p	a	m	a	n		l	u	k	a	n		a	n	p	a	n	m	a	n	

m” tidak cocok dengan “n”

- tabel OH : karakter “m” nilai pergeserannya = 2 belum ada karakter yang cocok
- tabel MH : ketidakcocokan pada posisi 8 (karakter “n”) nilai pergeserannya = 1
- sehingga geser string sebesar 2 posisi (nilai maksimal dari kedua tabel pergeseran)

tahap 4

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b	e	r	i	k	u	t		i	n	i		a	n	p	a	m	m	a	n		l	u	k	a	n		a	n	p	a	n	m	a	n
												a	n	p	a	n	m	a	n															

-”a” tidak cocok dengan “n”

- tabel OH : karakter “a” nilai pergeserannya = 1 belum ada karakter yang cocok
- tabel MH : ketidakcocokan pada posisi 8 (karakter “n”) nilai pergeserannya = 1
- sehingga geser string sebesar 1 posisi (nilai maksimal dari kedua tabel pergeseran)

tahap 5

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
b	e	r	i	k	u	t		i	n	i		a	n	p	a	m	m	a	n		l	u	k	a	n		a	n	p	a	n	m	a	n
												a	n	p	a	n	m	a	n															

”n” cocok dengan “n”

-”a” cocok dengan “a”

-”m” cocok dengan “m”

-”m” cocok dengan “n”

- tabel OH : karakter “m” nilai pergeserannya = 2, sudah ada 3 karakter cocok, nilai pergeseran = 2-3=-1 (pergeseran tidak mungkin dilakukan, hal ini merupakan kekurangan tabel occurrence heuristic, ada kemungkinan nilai pergeseran menjadi negatif)

- tabel MH : ketidakcocokan pada posisi 5 (karakter “n”) nilai pergeserannya = 6
- sehingga geser string sebesar 6 posisi (nilai maksimal dari kedua tabel pergeseran)

tahap 6

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
h	e	r	i	k	u	r		i	n	i		a	n	p	a	m	m	a	n		b	u	k	a	n		a	n	p	a	m	a	n	
																		a	n	p	a	m	a	n										

- ”n” cocok dengan “n”
- ”a” cocok dengan “a”
- ”k” tidak cocok dengan “m”
- tabel OH : karakter “k” tidak ada dalam string, sudah ada 2 karakter cocok, sehingga nilai pergeserannya = $8-2=6$
- tabel MH : ketidakcocokan pada posisi 6 (karakter “m”) nilai pergeserannya = 3
- sehingga geser string sebesar 6 posisi (nilai maksimal dari kedua tabel pergeseran)

tahap 7

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
h	e	r	i	k	u	r		i	n	i		a	n	p	a	m	m	a	n		b	u	k	a	n		a	n	p	a	m	a	n		
																									a	n	p	a	m	a	n				

- ”n” cocok dengan “n”
- ”a” cocok dengan “a”
- ”p” tidak cocok dengan “m”
- tabel OH : karakter “p” nilai pergeseran 5, sudah ada 2 karakter cocok, sehingga nilai pergeserannya = $5-2=3$
- tabel MH : ketidakcocokan pada posisi 6 (karakter “m”) nilai pergeserannya = 3
- sehingga geser string sebesar 3 posisi (nilai maksimal dari kedua tabel pergeseran)

tahap 8

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
h	e	r	i	k	u	r		i	n	i		a	n	p	a	m	m	a	n		b	u	k	a	n		a	n	p	a	m	a	n	
																											a	n	p	a	m	a	n	

semua karakter cocok, string yang dicari telah ditemukan.

3. Langkah pencarian string menggunakan Regex

Pada program ini, kami memanfaatkan library yang telah disediakan python untuk menyelesaikan string matching dengan regular expression. Berikut adalah langkah pemanfaatannya:

1. Impor modul regular expression yaitu dengan perintah `import re`
2. Buat objek Regex menggunakan metode `re.compile()` menggunakan raw string `r`
3. Lewatkan string ke metode `finditer(pattern, text, re.IGNORECASE)` yang mengembalikan list kosong (jika tidak ada match) dan list berisi hasil semua string matching pada teks yang ada.

Berikut adalah tinjauan simbol-simbol regex:

- `?` mencocokkan nol atau satu kali pola
- `*` mencocokkan nol, satu atau beberapa kali pola
- `+` mencocokkan satu atau beberapa kali pola
- `{n}` mencocokkan n kali pola
- `{n,}` mencocokkan n kali atau lebih pola
- `{,m}` mencocokkan 0 sampai m kali pola
- `{n,m}` mencocokkan n sampai m kali pola
- `{n,m}?` atau `*?` atau `+?` mencocokkan pola dengan mode non-greedy
- `^spam` berarti string harus diawali dengan spam
- `spam$` berarti string harus diakhiri dengan spam
- `.` cocok dengan karakter apapun kecuali karakter newline
- `\d`, `\w`, `\s` masing – masing cocok dengan digit, word, dan karakter kosong (spasi)
- `\D`, `\W`, `\S` kebalikan dari `\d`, `\w`, `\s` yaitu selain digit, word, dan karakter kosong
- `[abc]` cocok dengan karakter yang ada dalam tanda `[]`. Di contoh ini adalah karakter a, b, dan c
- `[^abc]` cocok dengan karakter selain yang ada dalam tanda `[]`. Di contoh ini adalah selain a, b, dan c

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4. Implementasi

Dalam pembuatan tugas aplikasi sederhana deteksi spam ini, kami menggunakan bahasa pemrograman python untuk mengimplementasikan algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore, dan juga Regex. Selain itu, kami menggunakan kakas PHP dan HTML-CSS sebagai pengembangan program berbasis web dan sebagai media interaksi program dan pengguna (*user interface*). Dalam penggabungan program python dan kakas PHP tersebut, kami juga menggunakan beberapa perantara file eksternal untuk transfer data dalam bentuk .txt dan .json. Berikut elemen-elemen yang diperlukan pada program kami:

a. File “home.php”

File ini berisi program HTML dan PHP untuk tampilan awal saat membuka aplikasi web kami. Program ini akan menampilkan halaman utama, nama aplikasi, serta beberapa *textbox* (form yang dapat diisi) yaitu form username, pilihan algoritma, serta *keyword* spam yang ingin dicari. Pada program ini, hasil pengisian username, pilihan algoritma, dan *keyword* akan dimasukkan ke file eksternal “input.txt” untuk kemudian diproses dengan memanggil execute pada program utama “ori.py”. Setelah didapatkan hasilnya, halaman ini akan melakukan *redirect* ke halaman “form.php” untuk menampilkan hasil tweet dan tanda spam.

b. File “form.php”

File ini berisi program HTML dan PHP untuk menampilkan hasil tweet berikut tanda spam yang sudah diproses program utama “ori.py”. Jadi, setelah mengeksekusi program ori.py, akan dihasilkan file json “twitterfeeds.json” yang kemudian akan di decode, lalu konten tiap tweet ditampilkan (berikut foto profil, username, teks tweet, dan waktu) dan tweet yang mengandung spam akan ditandai merah saat ditampilkan.

c. File “about.php”

File ini berisi program HTML dan PHP untuk menampilkan deskripsi para anggota pembuat aplikasi yang berisi foto dan deskripsi singkat per anggota. Halaman ini akan dibuka saat pengguna meng-klik pilihan “about us” pada halaman home.

d. Program python “ori.py”

Program ini merupakan program utama yang berisi implementasi algoritma KMP, Boyer-Moore, dan Regex. Program ini juga berisi penggunaan Twitter API untuk mengakses tweet berdasarkan username. Berikut penjelasan mengenai fungsi-fungsi yang terdapat pada ori.py:

- Fungsi **kmp(text, pattern)** dan **computeFail(pattern)**

Fungsi kmp() dan computeFail() ini berisi implementasi dari algoritma KMP. Parameter masukan text adalah teks yang akan dicek dan pattern adalah keyword spam yang akan dicari. Fungsi kmp() akan menggunakan fungsi computeFail() dan akan

mengembalikan indeks mulai dimana keyword spam ditemukan (jika spam) atau mengembalikan integer -1 jika tidak spam.

- Fungsi **bmMatch(text,pattern)** dan **buildLast(pattern)**

Fungsi `bmMatch()` dan `buildLast()` ini berisi implementasi dari algoritma Boyer-Moore. Parameter masukan `text` adalah teks yang akan dicek dan `pattern` adalah keyword spam yang akan dicari. Fungsi `bmMatch()` akan menggunakan fungsi `buildLast()` dan akan mengembalikan indeks mulai dimana keyword spam ditemukan (jika spam) atau mengembalikan integer -1 jika tidak spam.

- Fungsi **regex(text,pattern)**

Fungsi `regex()` ini berisi implementasi dari regex. Parameter masukan `text` adalah teks yang akan dicek dan `pattern` adalah keyword spam yang akan dicari. Fungsi ini akan mengembalikan indeks mulai dimana keyword spam ditemukan (jika spam) dalam bentuk list (semua indeks mulai akan disimpan) atau mengembalikan -1 jika tidak spam.

- Fungsi **accessTweets(username,p, opt)**

Fungsi ini berisi algoritma pengaksesan tweet sekaligus pengecekan keyword spam di tiap tweet. Pada awalnya, program ini akan meng-*authorize* beberapa kode untuk mengakses twitter API dengan akun tertentu. Setelah itu, akan diambil tweet dari profile seseorang yang memiliki username sesuai masukan pengguna dan untuk tiap tweet, beberapa data akan disimpan ke *dictionary* untuk kebutuhan tampilan di web nantinya.

Selain itu, teks dari tweet tersebut akan diproses menggunakan algoritma terpilih (KMP, Boyer Moore, atau Regex) dengan memanggil fungsi `spamMatch()` dengan `p` adalah `pattern` dan `opt` adalah pilihan algoritma. Pada *dictionary* tweet tersebut akan ditambahkan data spam (true/false) serta indeks letak spam. Setelah itu, *dictionary* tersebut akan di-*append* kedalam sebuah list dan kemudian list tersebut akan dikonversi dalam bentuk JSON dan dituliskan kedalam file “*twitterfeeds.json*”.

- Fungsi **spamMatch(text,pattern,pil)**

Fungsi ini dipanggil oleh `accessTweets()` untuk pengecekan apakah suatu teks tweet spam atau tidak. Parameter teks merupakan tweet yang akan dicek, `pattern` adalah keyword spam dan `pil` adalah pilihan algoritma. Disinilah fungsi `kmp()`, `bmMatch()` dan `regex()` dipanggil.

- Main Program

Program utama akan melakukan pembacaan file “*input.txt*” yang dihasilkan oleh program yang ada pada halaman *home.php* dan memanggil fungsi `accessTweets()` dengan parameter berasal dari pembacaan file tersebut.

e. File “*twitterfeeds.json*”

File ini merupakan hasil penyimpanan data JSON oleh program utama “ori.py”. File ini berisi list *dictionary* dari semua tweet yang diakses. Data yang terdapat pada dictionary adalah nama, username, link sumber foto profil, tweet, waktu tweet tersebut dibuat, keyword spam, posisi keyword spam, dan boolean apakah tweet tersebut spam atau tidak. File JSON ini kemudian akan diproses oleh program pada form.php untuk ditampilkan ke laman web.

f. File “input.txt”

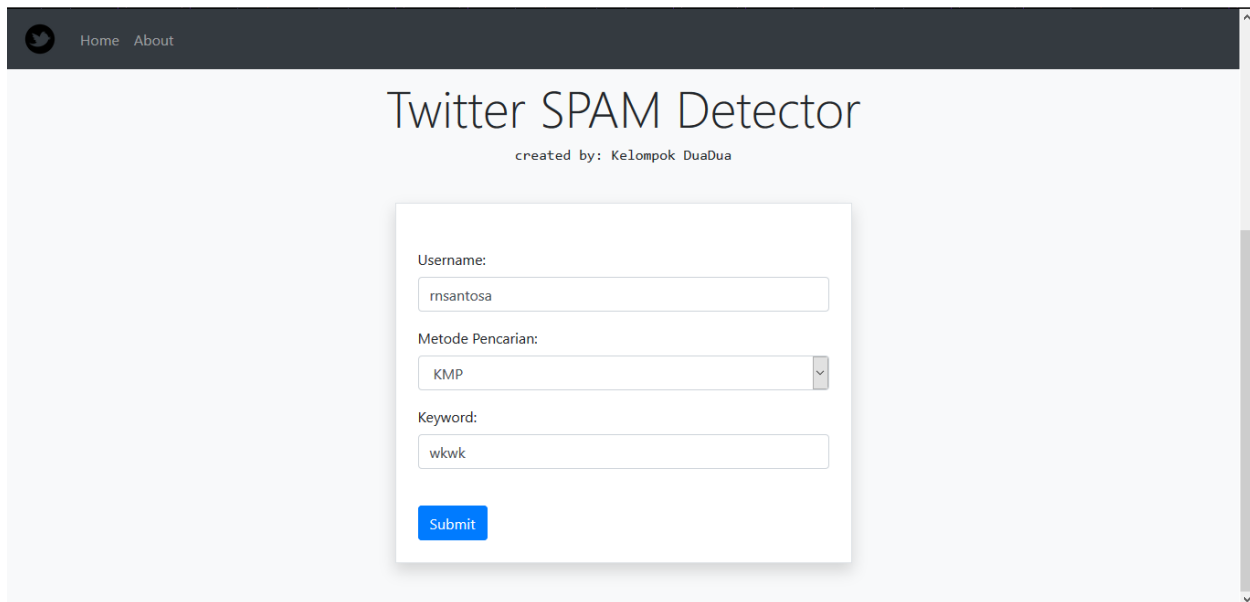
File ini cukup sederhana, hanya berisi 3 konten masukan pengguna dari halaman home pada web, yaitu username, pilihan algoritma, dan keyword spam. File ini akan dibaca oleh program utama “ori.py”

5. Pengujian

a. Menggunakan Algoritma KMP

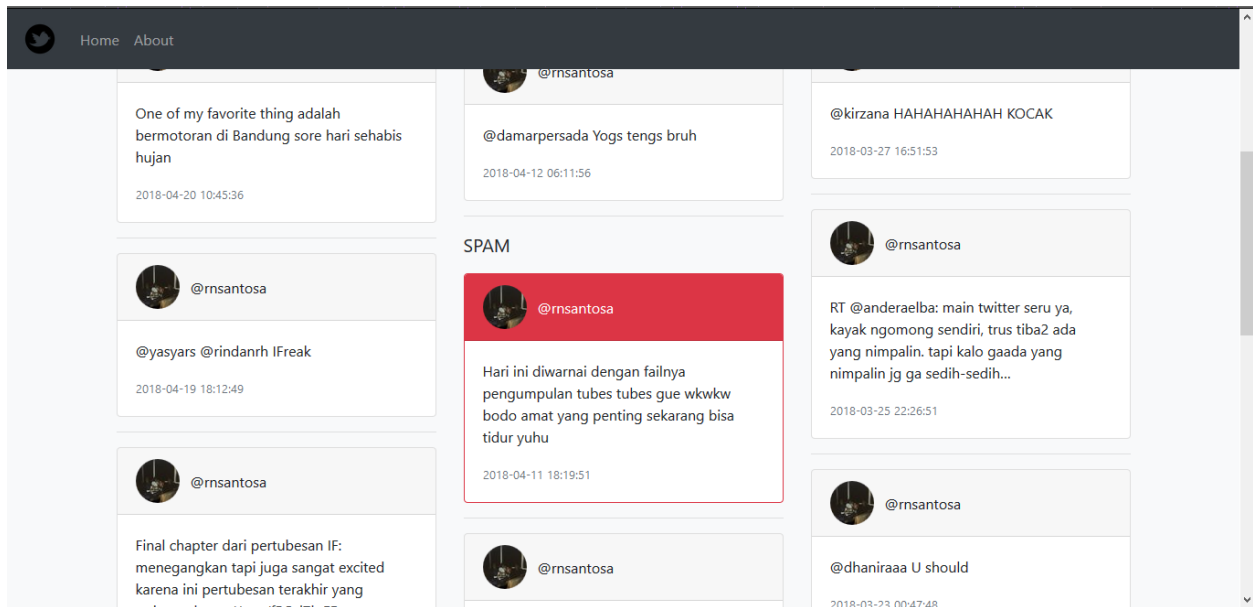
- **Kasus *keyword spam* semua lowercase**

Input adalah username twitter rnsantosa, pilihan algoritma yaitu KMP, dan keyword yaitu “wkwk”.

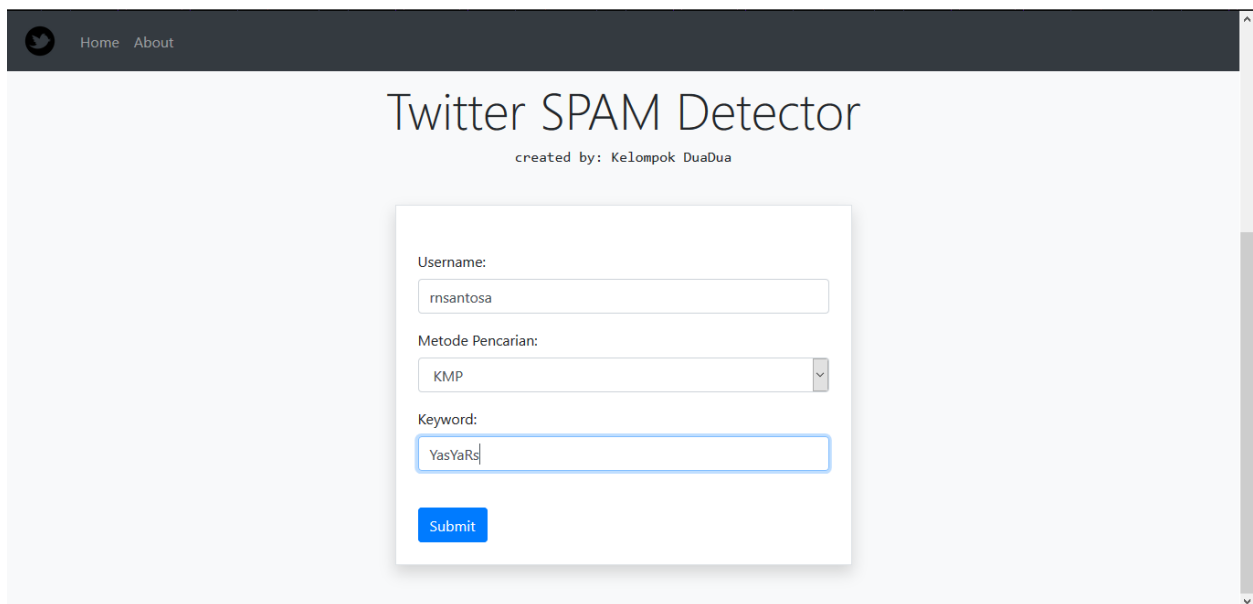


The screenshot shows a web application titled "Twitter SPAM Detector" with a subtitle "created by: Kelompok DuaDua". The interface includes a navigation bar with a Twitter logo and links for "Home" and "About". The main content area features a form with three input fields: "Username:" containing "rnsantosa", "Metode Pencarian:" with a dropdown menu set to "KMP", and "Keyword:" containing "wkwk". A blue "Submit" button is located at the bottom of the form.

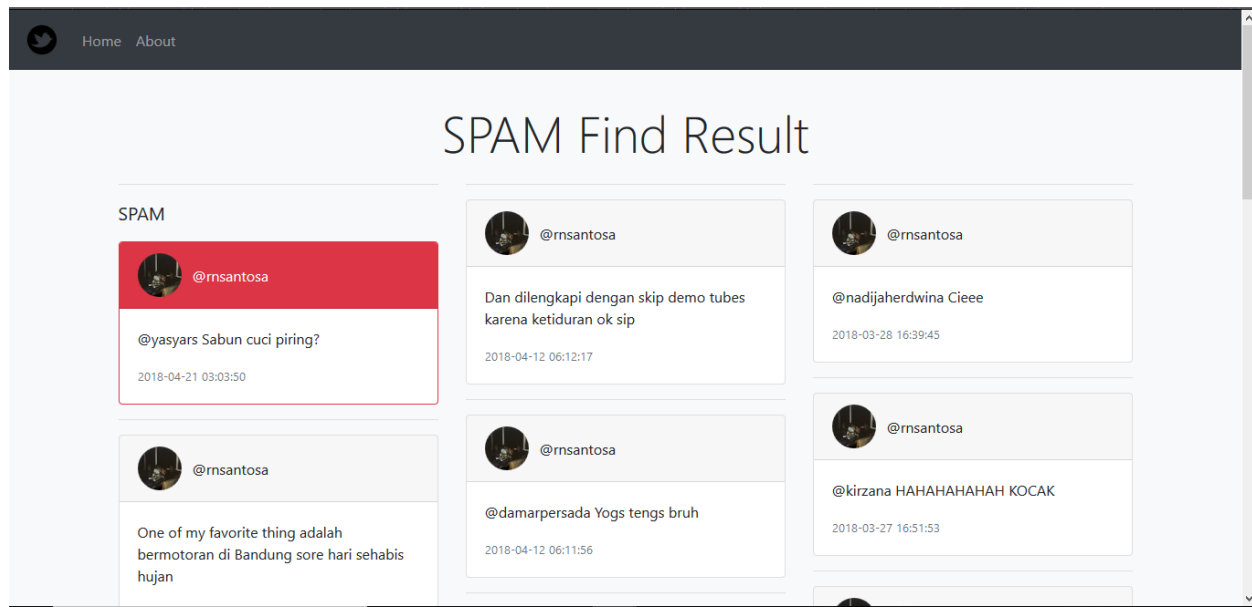
Output adalah 20 tweet terbaru pada timeline rnsantosa di mana terdapat satu buah tweet spam yaitu tweet yang mengandung keyword “wkwk” yang ditemukan pada kata “wkwkwk”



- **Kasus *keyword spam* semua campuran antara lower case dengan upper case**
Input adalah username twitter rnsantosa, pilihan algoritma yaitu KMP, dan keyword yaitu “YasYaRs”.



Output adalah 20 tweet terbaru pada timeline rnsantosa di mana terdapat dua buah tweet spam yaitu tweet yang mengandung keyword “YasYaRs” yang ditemukan pada kata “@yasyars”



b. Menggunakan Algoritma BM

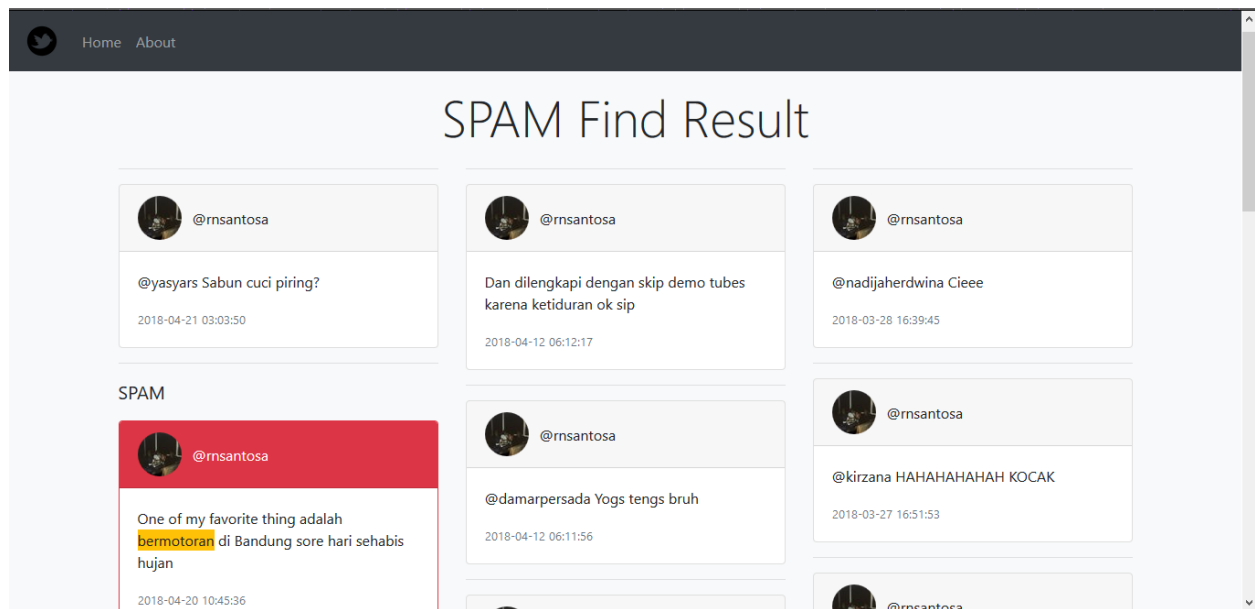
- **Kasus *keyword spam* semua lowercase**

Input dalah username twitter, pilihan algoritma, dan keyword spam yang akan dicari menggunakan algoritma terpilih di timeline user terpilih.

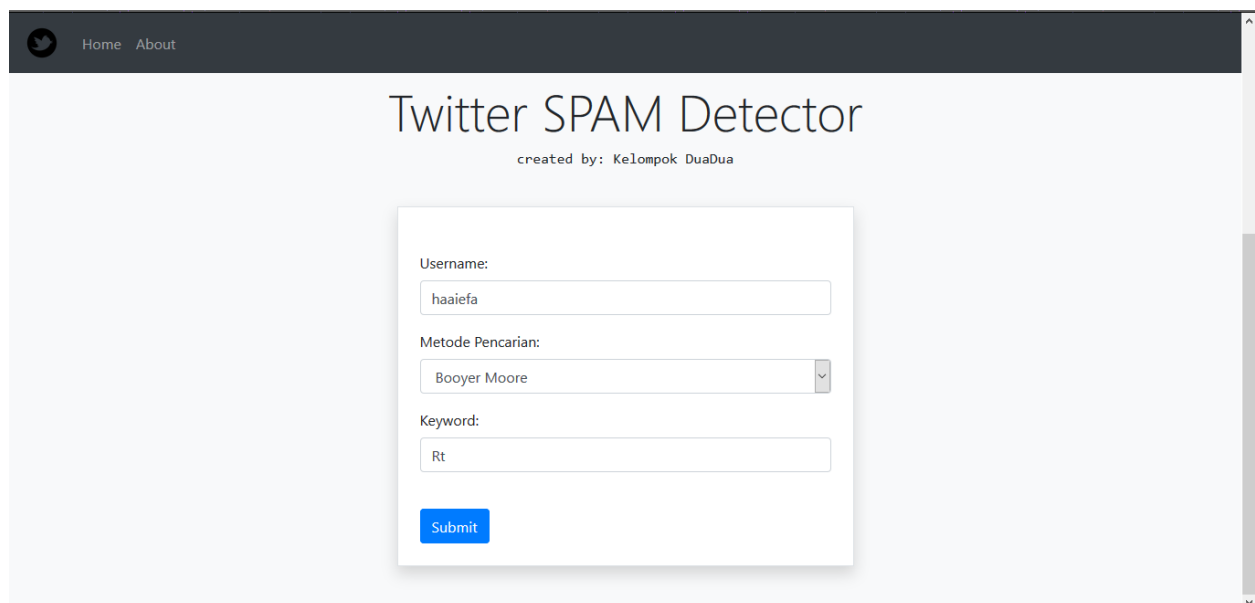
The screenshot shows a web application titled "Twitter SPAM Detector" created by Kelompok DuaDua. It features a form for searching tweets with the following fields:

- Username:** rnsantosa
- Metode Pencarian:** Booyer Moore (selected from a dropdown menu)
- Keyword:** motor
- Submit Button:** A blue button labeled "Submit".

Output adalah 20 tweet terbaru pada timeline rnsantosa di mana terdapat satu buah tweet spam yaitu tweet yang mengandung keyword “motor” yang ditemukan pada kata “bermotoran”.



- **Kasus *keyword spam* semua campuran antara lower case dengan upper case**
Input adalah username twitter haaiefa, pilihan algoritma yaitu BM, dan keyword yaitu “Rt”.



Output adalah 20 tweet terbaru pada timeline haaiefa di mana terdapat sepuluh buah tweet spam yaitu tweet yang mengandung keyword “Rt” yang ditemukan pada berbagai kata.

SPAM Find Result


 @dhaniraaa

@dhaniraaa @jonathanend Lu kan naik lift

2018-04-22 21:14:57


SPAM

RT @prastards: A Quiet Place (2018) memiliki pesan mendalam bagi kehidupan manusia agar senantiasa tidak banyak bacot anjenk ntar mati.

 @haaiefa

@miraristym Wait what


2018-04-18 13:30:55

 @haaiefa

@kitkatcranci WKWKWKWKWK

2018-04-18 05:35:11

SPAM

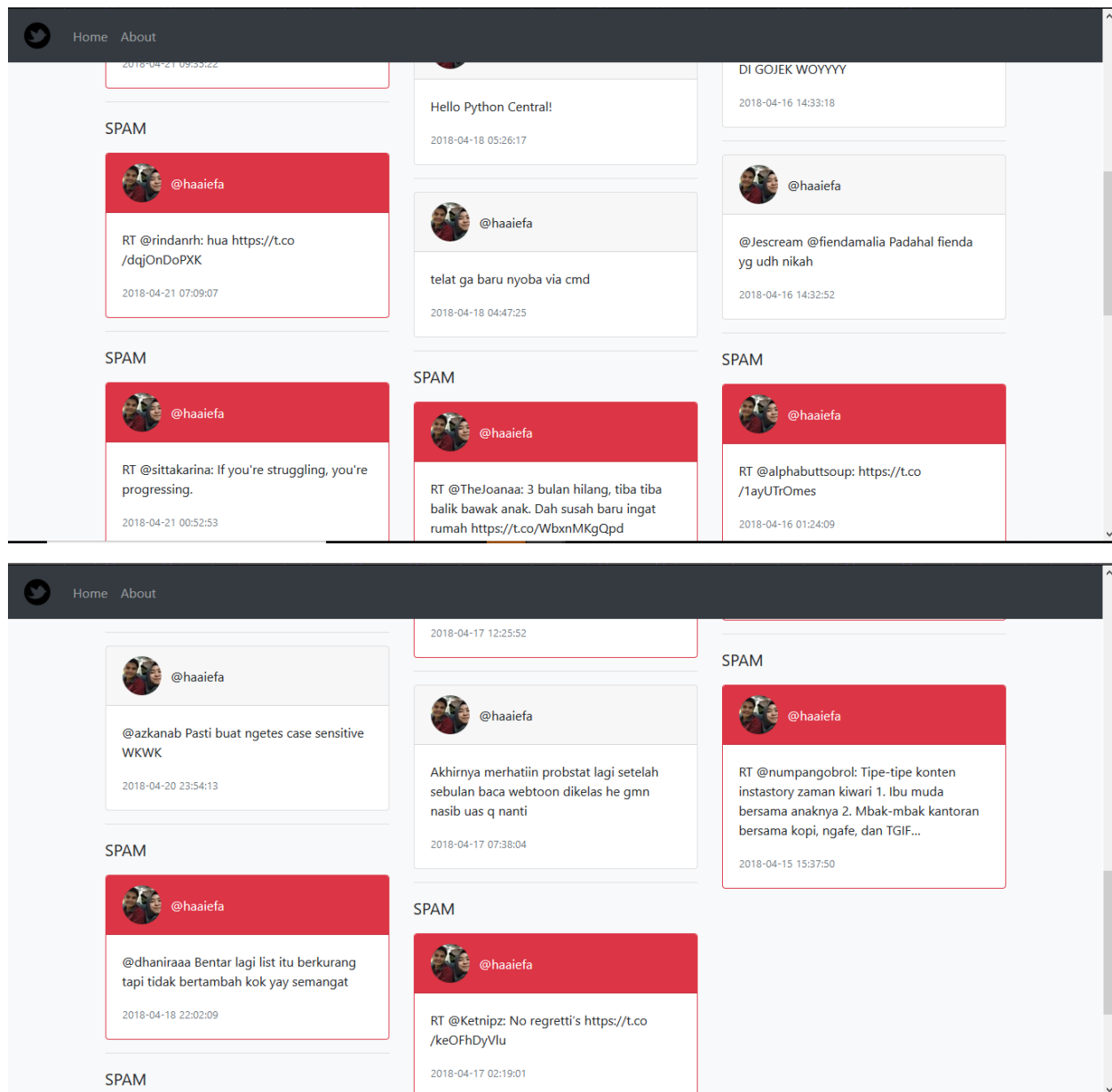
 @haaiefa

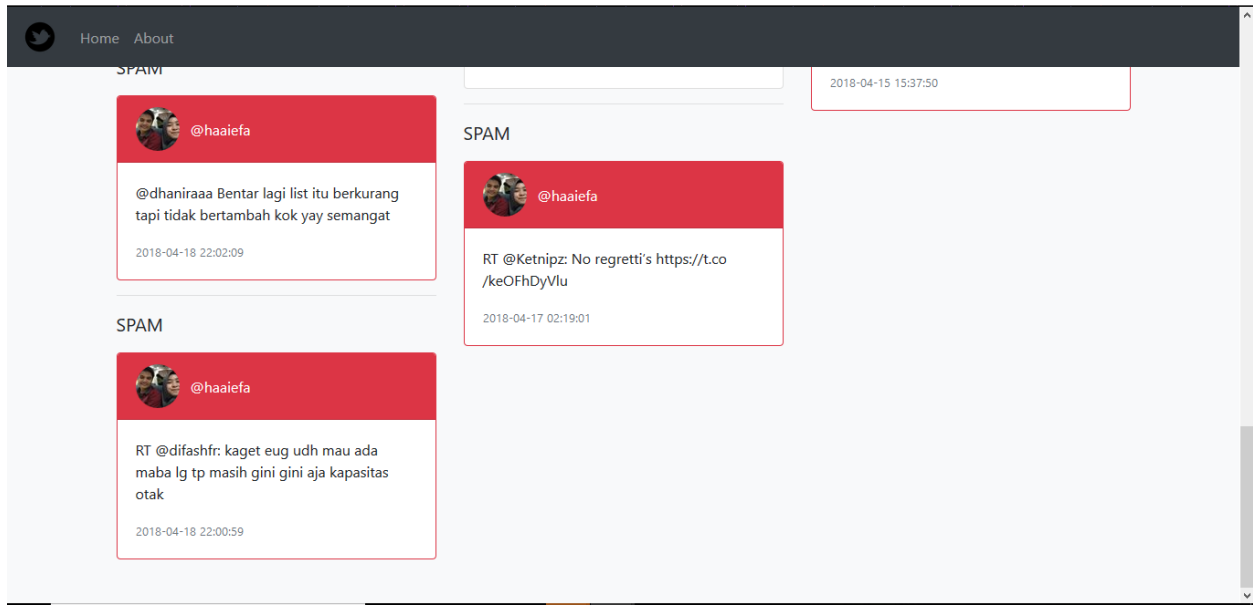
RT @mairanotmaria2: My worst fear is that I'll fail my finals and have a shitty GPA and not be able to return all the sacrifices my parents...

2018-04-16 14:52:39

@haaiefa

@fiendamalia @kitkatcranci
HAHAHAHAHAHAHAHA NGAKAK GUA LAGI

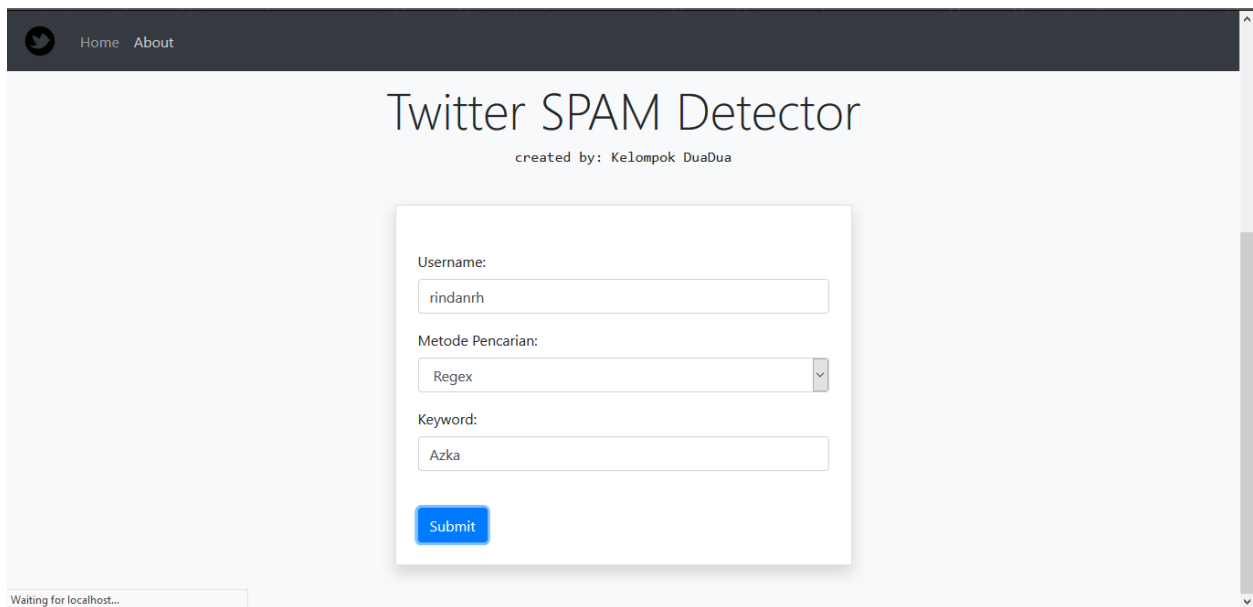




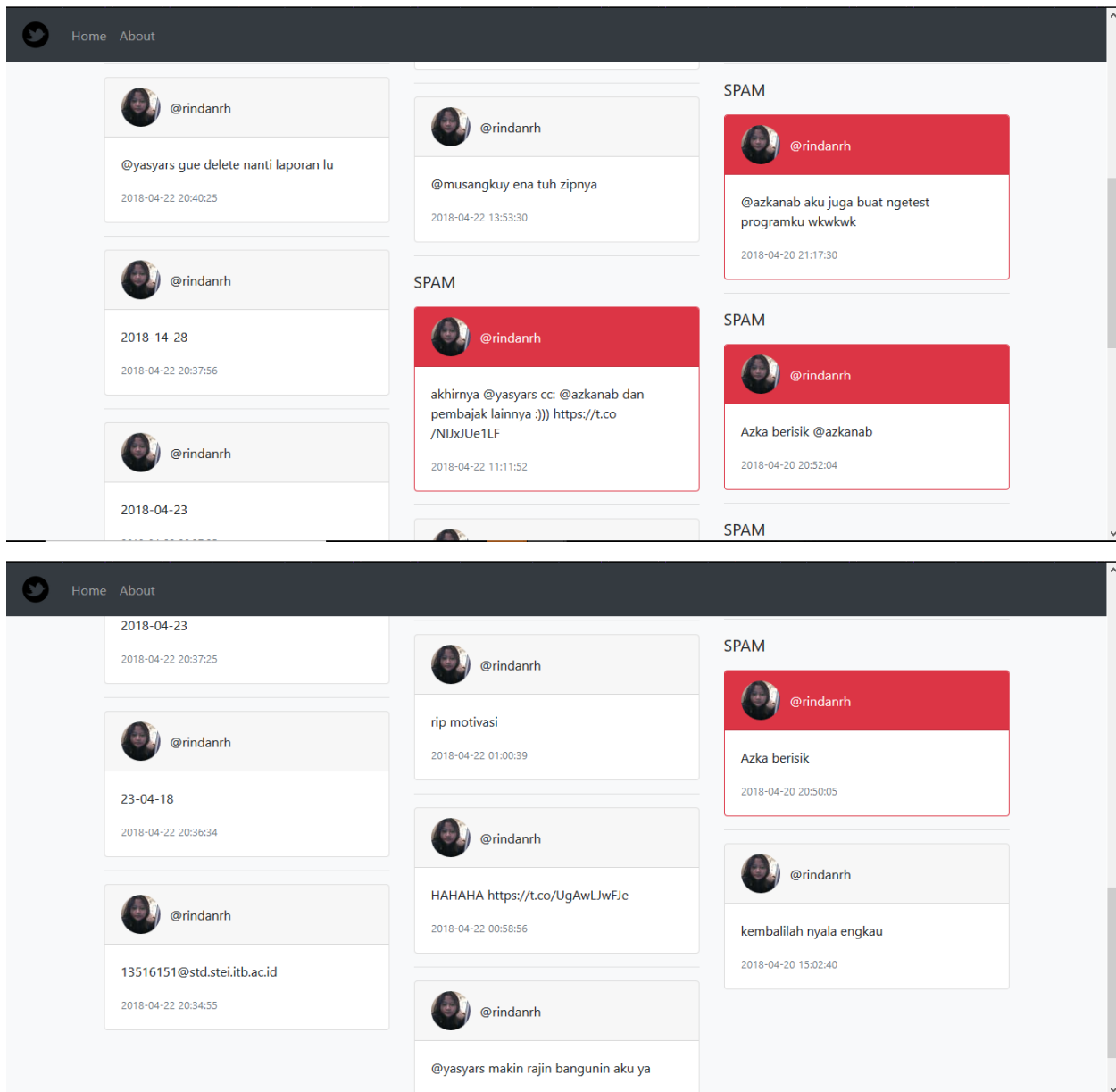
c. Menggunakan Regex

- **Kasus *keyword string normal***

Input adalah username twitter rindanrh, pilihan algoritma yaitu regex, dan keyword yaitu “Azka”.



Output adalah 20 tweet terbaru pada timeline rindanrh di mana terdapat empat buah tweet spam yaitu tweet yang mengandung keyword “Azka” yang ditemukan pada berbagai kata.

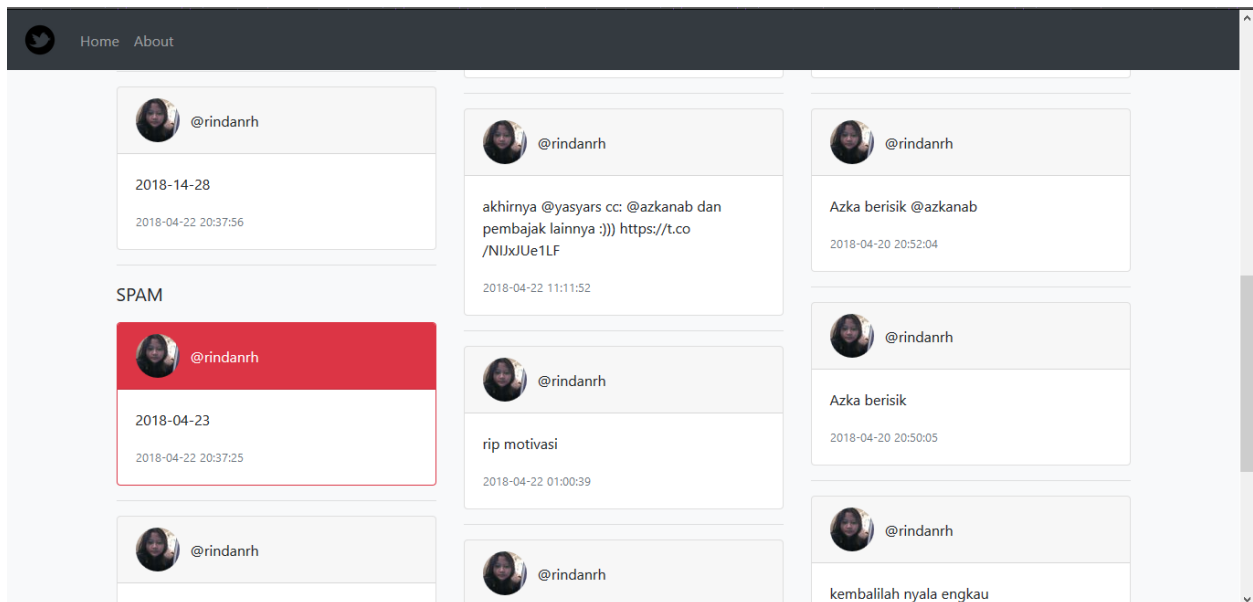


- **Kasus *keyword* adalah *pattern regex* yang kompleks - tanggal**


Input adalah username twitter rindanrh, pilihan algoritma yaitu regex, dan keyword yaitu “`^(19|20)\d\d[-/.] (0[1-9]|1[012])[-/.] (0[1-9]|12)[0-9]3[01])$`”. Keyword tersebut digunakan untuk mencari *pattern* berupa tanggal dengan format *yyyy-mm-dd* di mana batasan dari bulannya adalah 12, batasan dari tanggal adalah 31, dan batasannya adalah tanggal 1900-01-01 hingga 2099-12-31.

The screenshot shows the Twitter SPAM Detector interface. At the top, there is a navigation bar with a Twitter logo and links for 'Home' and 'About'. The main heading is 'Twitter SPAM Detector' with a subtitle 'created by: Kelompok DuaDua'. Below this is a search form with three fields: 'Username:' containing 'rindanrh', 'Metode Pencarian:' with a dropdown menu set to 'Regex', and 'Keyword:' containing a complex regex pattern: `^(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])$`. A blue 'Submit' button is located at the bottom of the form.

Output adalah 20 tweet terbaru pada timeline rindanrh di mana terdapat satu buah tweet spam yaitu tweet yang mengandung keyword “`^(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])$`”.



- **Kasus *keyword* adalah *pattern* regex yang kompleks - e-mail**
 Input adalah username twitter rindanrh, pilihan algoritma yaitu regex, dan keyword yaitu “`\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b`”. Keyword tersebut digunakan untuk mencari *pattern* berupa email.



[Home](#)
[About](#)

Twitter SPAM Detector

created by: Kelompok DuaDua

Username:


Metode Pencarian:

Regex
 


Keyword:


Submit


Output adalah 20 tweet terbaru pada timeline rindanrh di mana terdapat dua buah tweet spam yaitu tweet yang mengandung keyword “\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b”.



[Home](#)
[About](#)


SPAM Find Result



 @rindanrh
 @inibukanyoko ganteng
 2018-04-22 20:52:16



 @rindanrh
 @yasyars gadeng luv
 2018-04-22 20:40:35



 @rindanrh

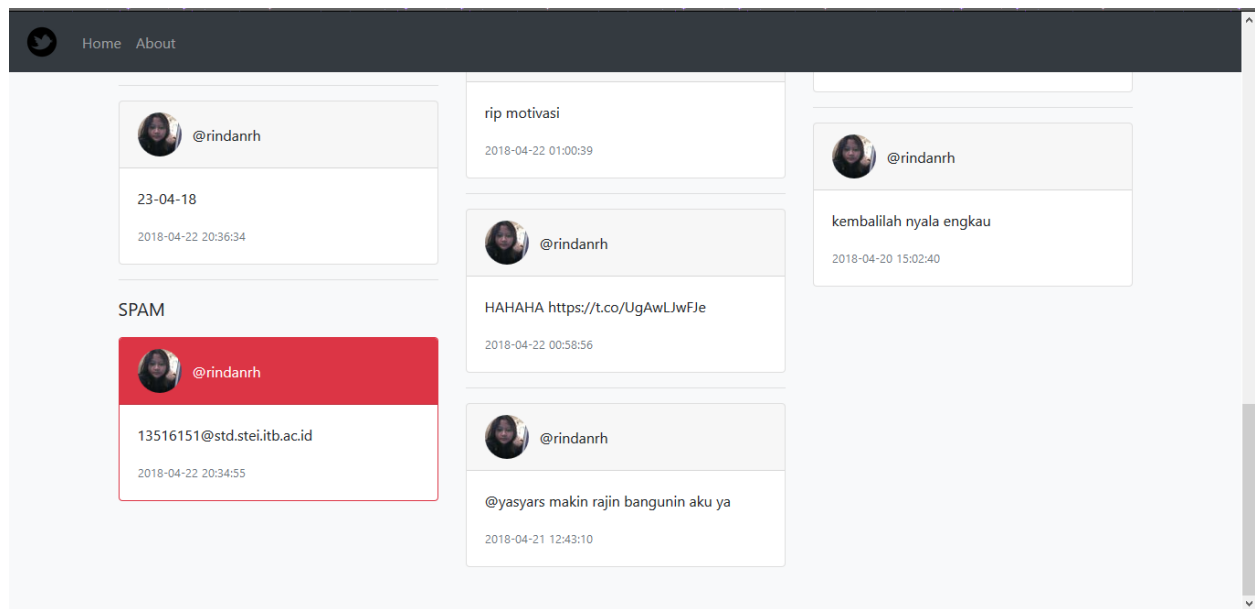
SPAM

 @rindanrh
 rindanurh@gmail.com
 2018-04-22 20:34:12


 @rindanrh
 @ShintaAyuCK @faizzunS amin aku mendukung kalian
 2018-04-22 19:17:57


 @rindanrh
 @Sabdalam tugaskuu hehe
 2018-04-21 06:25:08


 @rindanrh
 hua <https://t.co/dqjOnDoPXX>
 2018-04-21 06:18:49


 @rindanrh



BAB 5

KESIMPULAN DAN SARAN

A. Kesimpulan

Penyelesaian masalah sampah elektronik (*electronic spam*) dapat dilakukan dengan pendekatan strategi algoritma pattern-matching (string matching), karena algoritma ini dapat mendeteksi keberadaan suatu potongan kata pada suatu teks panjang beserta lokasi potongan kata tersebut, hal ini dapat digunakan untuk mengurangi konten yang mengandung sampah elektronik dengan mendeteksi konten menggunakan *keyword* tertentu. Algoritma pattern matching dapat mengaplikasikan metode KMP, Boyer-Moore, atau Regex. Pada program web yang kami buat, pengguna akan mengisi field username, pilihan algoritma, dan keyword spam lalu kemudian diproses dan tweet yang diakses ditampilkan dan ditandai jika spam. Secara umum, program telah berhasil berjalan dengan baik untuk semua jenis algoritma.

B. Saran

Program dapat dikembangkan untuk tidak hanya mengakses tweet dari user tertentu, tetapi juga tweet berdasarkan lokasi, hashtag, maupun faktor lainnya. Selain itu, pengaksesan data teks untuk deteksi spam juga dapat diperluas dari sumber selain twitter, misalnya line, facebook, atau sumber konten lainnya agar program web yang kami buat dapat lebih aplikatif dan bermanfaat.

DAFTAR PUSTAKA

- [1] Munir, Rinaldi. 2007. Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Penerbit Informatika, Palasari.
- [2] https://en.wikipedia.org/wiki/Regular_expression diakses pada tanggal 22 April 2018 pukul 23.28
- [3] <http://customations.blogspot.co.id/2014/10/apa-yang-di-maksud-spam-dan-apa-tujuannya.html> diakses pada tanggal 22 April 2018 pukul 23.54
- [4] <https://id.wikipedia.org/wiki/Twitter> diakses pada tanggal 23 April 2018 pukul 00.07
- [5] <https://id.wikipedia.org/wiki/PHP> diakses pada tanggal 23 April 2018 pukul 00.07
- [6] <https://fnsfind16.wordpress.com/2015/01/03/boyer-moore-string-matching/> diakses pada tanggal 23 April 2018 pukul 05.40
- [7] <http://technoprast.blogspot.co.id/2011/04/kasus-pada-algoritma-boyer-moore.html/> diakses pada tanggal 23 April 2018 pukul 05.48
- [8] <https://www.codepolitan.com/belajar-regex-dengan-python-584106394cd6a-59> diakses pada tanggal 23 April 2018 pukul 11.15