Aplikasi *String Matching* pada *Plugin SMS Blocker* untuk Validasi Pesan

Mario Tressa Juzar 13512016[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512016@std.stei.itb.ac.id

***Abstrak* — Dewasa ini kehidupan manusia tidak dapat dipisahkan dari gadget, terutama telepon seluler. Telepon selular mayoritas dimiliki oleh setiap orang karena harganya yang terjangkau. Berinteraksi dengan telepon seluler antar manusia bisa dilakukan dengan telpon ataupun fasilitas SMS (*Short Message Service*). Namun, terkadang ada yang memanfaatkan SMS ini untuk hal yang kurang baik. Karena itulah muncul *plugin* pada aplikasi SMS untuk memfilter hal yang kurang baik tersebut. Makalah ini akan membahas bagaimana *string matching* pada *plugin SMS Blocker* untuk memvalidasi pesan agar pesan yang kurang baik tidak mengganggu pengguna SMS.**

***Index Terms* — *string matching*, validasi, *plugin, SMS blocker*.**

I. PENDAHULUAN

SMS (*Short Messager Service*) merupakan layanan dari penyedia layanan komunikasi untuk mengirim dan menerima suatu pesan singkat berupa teks melalui perangkat nirkabel, yaitu perangkat komunikasi yang dalam hal ini yang digunakan adalah telepon seluler.

Sebuah pesan SMS maksimal terdiri dari 140 bytes, dengan kata lain sebuah pesan bisa memuat 140 karakter 8-bit, 160 karakter 7-bit atau 70 karakter 16-bit untuk Bahasa Jepang, Bahasa Korea dan Bahasa Mandarin yang memakai Hanzi (Aksara Kanji/Hanja). Selain 140 bytes ini ada data-data lain yang termasuk. Adapula beberapa metode untuk mengirim pesan yang lebih dari 140 bytes, tetapi seorang pengguna harus membayar lebih dari sekali. Misalnya pesan yang dikirimkan terdiri dari 167 karakter, maka pesan ini akan dipecah menjadi 2 buah SMS (1 buah SMS dengan 160 karakter dan 1 SMS dengan 7 karakter).

Plugin adalah sebuah program komputer yang menambah fungsionalitas program utama. Program utama biasanya memberikan antar-muka (*interface*) agar *plugin* dapat berinteraksi dengan program utama. *Plugin* ditambahkan kemudian dan bisa menjadi bagian dari program utama.

Setiap orang tidak dapat dipisahkan dari komunikasi dalam kehidupan sehari-hari. Dengan zaman yang semakin maju, komunikasi tidak bisa dilepaskan dari gadget yang mempermudah komunikasi. Salah satu gadget yang paling banyak digunakan untuk berkomunikasi adalah telepon seluler.

Banyak fitur pada telepon seluler yang bisa dimanfaatkan untuk komunikasi antar sesama manusia. Dengan makin berkembangnya telepon seluler semakin berkembang pula aplikasi untuk mendukung komunikasi tersebut. Yang paling banyak dimanfaatkan oleh kebanyakan orang adalaha fitur SMS. Dengan semakin banyaknya dan beragamnya pengguna SMS ini maka semakin banyak pula yang menggunakannya dalam hal yang tidak wajar seperti menggunakan kata-kata yang tidak baik dalam kehidupan sehari-hari dalam SMS.

Pada makalah ini, penulis akan membahas tentang penggunaan string matching untuk memvalidasi pesan yang kurang baik ataupun pesan negatif yang dikirimkan oleh pengguna SMS (pengirim) kepada penerima SMS. Caranya adalah menggunakan SMS *blocker*, yaitu dengan melakukan validasi terhadap pesan yang diterima dengan unsur-unsur negatif yang telah didefinisikan oleh pengguna sebelumnya. Setelah divalidasi kemudian pesan dipisahkan dari kotak masuk SMS. Setelah itu baru ditentukan apa yang akan dilakukan pada pesan tersebut.

II. *STRING MATCHING*

String matching atau yang biasa dikenal dengan *pattern matching* adalah suatu cara/algoritma untuk melakukan pencarian semua kemunculan *string* pendek (*pattern*) di dalam string yang lebih panjang (teks). *String* yang bakal dijadikan pola yang akan dicari pada sebuah teks disebut dengan *pattern*. Untuk memperjelas maksud dari *string matching* ini, berikut diberikan sebuah contoh mengenai *string matching* :

1. T : Teks (*text*) yaitu *string* yang panjangnya n karakter.
2. P : *Pattern*, yaitu *string* dengan panjang m karakter (asumsi bahwa m < n) yang akan dicari pada teks.

Kemudian, carilah lokasi pertama kemunculan *pattern* pada teks.

Contoh :

T : "Budi dan teman-temannya sedang bermain bola dilapangan"

P : "main"

Dengan menggunakan algoritma *string matching*, *pattern* "main" akan ditemukan dalam kata "bermain".

Algoritma *string matching* ini memiliki banyak sekali pengaplikasiannya dalam ilmu pengetahuan. Contohnya adalah pencarian di dalam editor text, *web search engine* (misal : Google, Yahoo, dll), analisis citra, hingga ke aspek bioinformatika (misal: pencocokan rantai asam amino pada rantai DNA, penentuan senyawa hasil reaksi kimia, dll).

Sebuah *string* dapat kita pandang memiliki prefix dan suffix (awalan dan akhiran) yang dapat menjadi ide dasar dalam algoritma *string matching*. Asumsi S adalah sebuah *string* dengan panjang m.

$$S = x1x2...xm$$

Prefix dari S adalah sebuah *substring* S[1...k-1] .
Suffix dari S adalah sebuah *substring* S[k-1...m]
dengan :

- k adalah sebuah indeks di antara 1 dan m
- S[0] adalah sebuah karakter *null*, dan bersimbol Ø

Contoh:

S :

b	e	r	m	a	i	n

Semua kemungkinan prefix dari S adalah :
- "Ø", "b", "be", "ber", "berm", "berma", "bermai"

Semua kemungkinan suffix dari S adalah:
- "Ø", "n", "in", "ain", "main", "rmain", "ermain"

Algoritma untuk *string matching* diantaranya yaitu algoritma *brute force*, algoritma Knuth-Morris-Pratt (KMP), dan algoritma Boyer-Moore.

A. Algoritma *Brute Force*

Algoritma *Brute Force* merupakan algoritma pencocokan *string* yang ditulis tanpa memikirkan peningkatan performa. Algoritma ini mengecek setiap posisi di dalam *text* T untuk melihat apakah sebuah *pattern* P dimulai pada posisi tersebut. *Pattern* P bergeser satu karakter pada setiap waktu melalui T.

Secara sistematis, langkah-langkah yang dilakukan algoritma *brute force* pada saat mencocokkan string adalah:

1. Algoritma *Brute Force* 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 terus menggeser *pattern* sebesar satu ke kanan, dan mengulangi langkah ke-2 sampai *pattern* berada di ujung teks.

Algoritma *Brute Force* memiliki kompleksitas waktu rata-rata O(m+n). Algoritma ini akan cepat apabila karakter di dalam sebuah teks bernilai besar (misal : A..Z, a..z, 1...9, dll). Namun, algoritma ini akan lambat apabila karakter di dalam sebuah teks bernilai kecil (misal : 0,1 (data biner)).

B. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma *string matching* yang dikembangkan secara terpisah oleh Donald E. Knuth dan James H. Morris bersama Vaughan R. Pratt.

Algoritma KMP melihat *pattern* di dalam teks dari kiri ke kanan layaknya algoritma *brute force*. Namun, algoritma ini menggeser *pattern* secara ”lebih cerdas” dari pada algoritma *brute force*. Berbeda dengan algoritma *brute force*, informasi yang digunakan masih dipelihara untuk melakukan jumlah pergeseran. Pergeseran yang dilakukan membuat algoritma KMP lebih baik dari algoritma *brute force* karena bisa menghindari perbandingan yang tidak perlu.

Algoritma KMP menggunakan fungsi pinggiran KMP (KMP *Border Function*) sebagai praproses *pattern* untuk mencari kecocokan dari prefix dari *pattern* dengan *pattern*nya sendiri.

Berikut beberapa keterangan yang akan digunakan dalam algoritma KMP :

- j = posisi yang tidak sama di dalam P[]
- k = posisi sebelum ketidakcocokan (k = j-1)
- B*order function* b(k) didefiniskan sebagai panjang dari prefix yang paling panjang dari P[1..k] yang juga merupakan suffix dari P[1..k]
- Nama lainnya : *failure function* (disingkat: *fail*)

Contoh *border function* KMP :
P : “abaacaba”
j : 12345678

j	1	2	3	4	5	6	7	8
P[j]	a	b	a	a	c	a	b	a
b(j)	0	0	1	1	0	1	2	3

Tabel 1 : contoh KMP *border function*

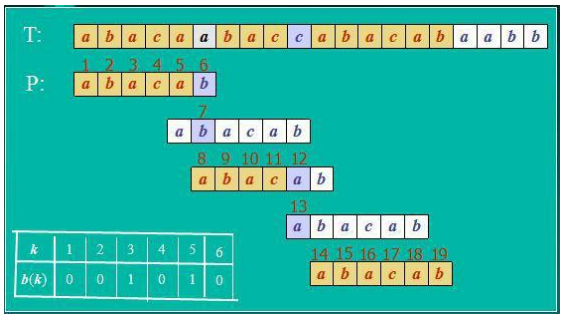
b(j) adalah ukuran dari border yang paling besar. Di dalam kode program, b() direpresentasikan dengan array, seperti yang di tabel.

Secara sistematis, berikut merupakan langkah-langkah yang dilakukan algoritma Knuth-Morris-Pratt pada saat mencocokkan *string* adalah :

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.

Kompleksitas waktu terbaik dari algoritma brute force adalah $O(n)$. Kasus terbaik terjadi jika pada operasi perbandingan, setiap huruf pattern yang dicocokkan dengan awal dari teks adalah sama. Kompleksitas waktu terburuk dari *brute force* adalah $O(mn)$. Kompleksitas waktu rata-rata algoritma KMP adalah O(m+n). Namun algoritma KMP tidak cocok pada *pattern* dan teks dengan ukuran yang besar (Alfabet, Numerik,dll).

Berikut merupakan ilustrasi dari cara kerja algoritma KMP:



Gambar 1 : ilustrasi cara kerja algoritma KMP

C. Algoritma Boyer-Moore

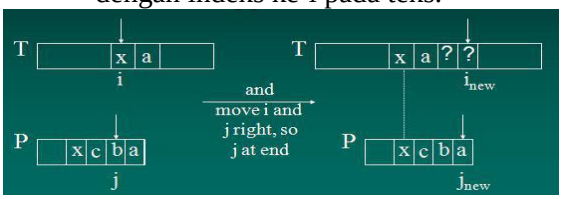
Algoritma Boyer-Moore adalah salah satu algoritma pencarian *string* yang dipublikasikan oleh Robert S. Boyer dan J. Strother Moore. Algoritma ini dianggap sebagai algoritma yang paling efisien pada aplikasi umum. Algoritma Boyer-Moore berbeda dari algoritma-algoritma *string matching* sebelumnya, algoritma Boyer-Moore mulai mencocokan karakter dari sebelah kanan *pattern*. Ide dibalik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan maka akan lebih banyak informasi yang didapat.

Algoritma Boyer-Moore didasarkan pada 2 teknik, yaitu :

1. *Looking-Glass technique*
- Teknik mencari *pattern* di dalam teks dengan cara bergeser mundur melalui *pattern* yang dimulai dari karakter paling terakhir / belakang
2. *Character-Jump technique*
- Teknik ini dilakukan apabila ketidakcocokan terjadi pada indeks ke-i pada teks dan karakter pada indeks ke-j pada *pattern* itu tidak sama dengan indeks ke-i pada teks.

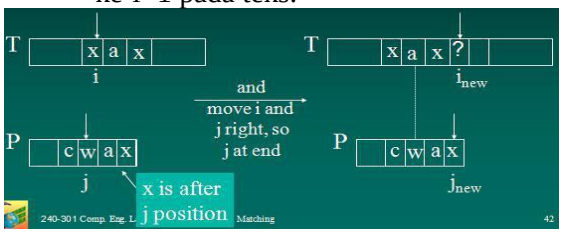
Algoritma Boyer-Moore memiliki 3 kasus yang dapat terjadi di dalamnya, yaitu :

1. Jika *pattern* P memiliki karakter x di dalamnya, lalu akan dicoba untuk menggeser P ke kanan hingga sejajar dengan kemunculan terakhir dari x di dalam P dengan indeks ke-i pada teks.



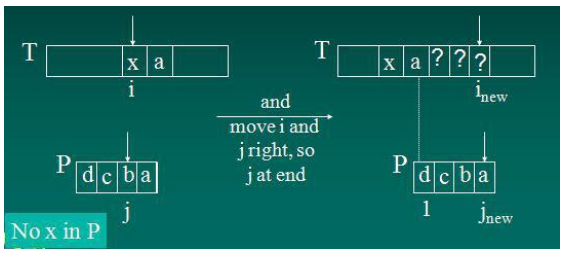
Gambar 2 : Kasus 1 pada Algoritma BM

2. Jika *pattern* P memiliki karakter x di dalamnya, namun apabila penggeseran ke kanan hingga kemunculan terakhir dari x itu tidak memungkinkan, maka geser P ke kanan sebanyak 1 karakter hingga indeks ke-i+1 pada teks.



Gambar 3 : Kasus 2 pada Algoritma BM

3. Apabila kasus 1 dan 2 tidak dapat diterapkan, lalu geser *pattern* P hingga indeks pertama pada *pattern* sejajar dengan indeks ke-i+1 pada teks.



Gambar 4 : Kasus 3 pada Algoritma BM

Algoritma Boyer-Moore memiliki praproses terhadap *pattern* P dan alfabet A untuk membuat fungsi L() / *last occurrence function*. L() memetakan semua huruf pada A menjadi integer.

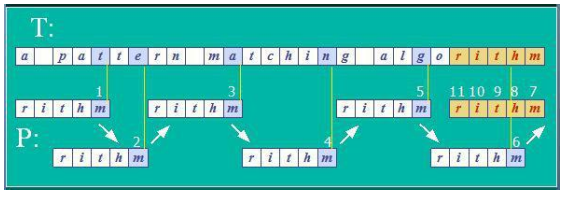
L(x) didefinisikan sebagai indeks i yang paling besar sehingga P[i] == x atau memiliki nilai -1 apabila tidak ada indeks yang didapatkan dengan x adalah sebuah huruf pada A.

Secara sistematis, berikut merupakan 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 dalam 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.

Kompleksitas untuk algoritma Boyer-Moore dapat dinyatakan dalam O(n + x) dengan x adalah besar ruang alfabet. Algoritma ini akan cepat apabila alfabet A memiliki ukuran yang besar, dan lambat apabila ukuran alfabet nya kecil (contoh : cepat untuk pencarian pada teks berbahasa Inggris, lambar untuk pencarian pada teks biner).

Ilustrasi cara kerja algoritma Boyer-Moore digambarkan pada gambar di bawah :



Gambar 5 : Ilustrasi cara kerja algoritma BM

III. VALIDASI PESAN DENGAN MENGGUNAKAN *STRING MATCHING* PADA SMS *BLOCKER*

Melalui telepon seluler, setiap orang bisa mengirim pesan (SMS) kepada pengguna telepon seluler lain. Interaksi dengan SMS ini sangat membantu komunikasi antar manusia. Namun, tidak ada yang bisa menjamin setiap pesan yang dikirim melalui SMS tersebut tidak negatif ataupun unsur SARA dan sebagainya.

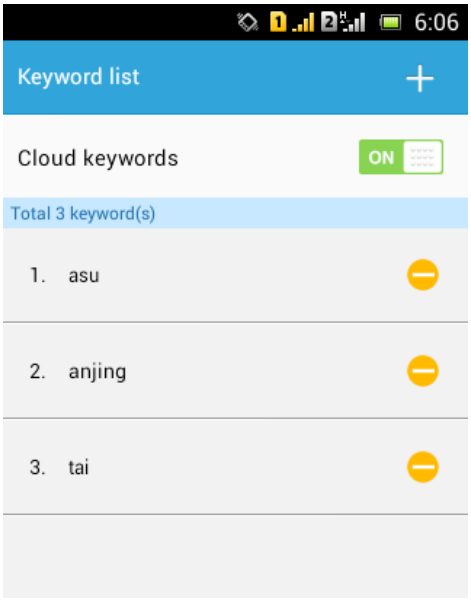
Operator telepon seluler sendiri tidak bisa begitu saja mem-*block* setiap pesan yang dikirim oleh pengirim SMS karena banyak hal. Pertama, karena setiap pesan itu privasi masing-masing orang yang harus disampaikan dari pengirim SMS kepada penerima SMS. Jika operator membatasi maka ini sama saja dengan melanggar privasi orang tersebut. Kedua, definisi unsur-unsur/kata-kata negatif itu sendiri yang relatif. Indonesia itu luas, setiap daerah mempunyai daerah masing-masing dan di satu daerah suatu kata bisa memiliki makna yang berbeda di daerah lain. Misalnya, di Padang kata "galak" itu berarti tertawa sedangkan di Jawa khususnya "galak" itu berarti sifat orang yang mudah marah. Definisi negatif pun tidak bisa disama-ratakan untuk seluruh Indonesia. Karena hal itulah SMS tidak bisa di*filter* langsung oleh operator seluler.

Untuk mengatasi hal tersebut muncullah sebuah solusi untuk membuat aplikasi/*plugin* pada SMS yang bisa mem*filter* pesan dengan kata-kata tidak baik/negatif yang kita definisikan sendiri. *Plugin*/aplikasi ini dikenal dengan nama SMS *blocker*.

SMS *blocker* ini bisa mem-*block* pesan yang masuk dengan kata-kata negatif/tidak baik yang kita definisikan sendiri. Meskipun namanya SMS *blocker* namun ia tidak langsung membuang SMS tersebut, namun diletakkan di tempat khusus SMS yang telah di*block* yang kemudian bisa diputuskan apakah pesan tersebut tetap ingin dibaca ataupun akan dihapus.

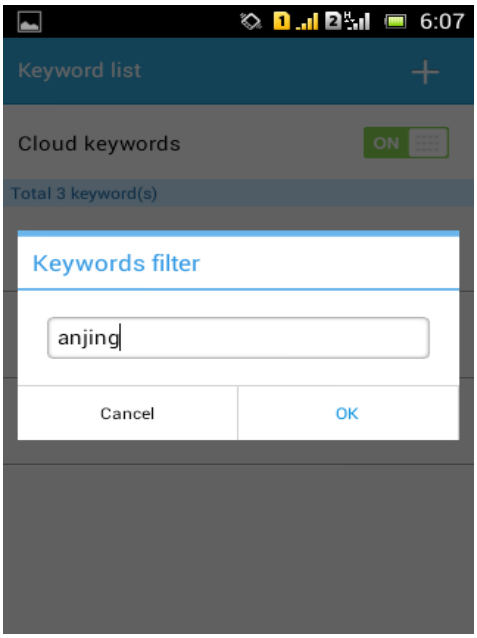
SMS *blocker* ini memiliki keyword list yang akan dicocokkan dengan isi pesan. Jika isin pesan mengandung kata-kata buruk yang kita definisikan pada keyword list maka pesan akan ditangani secara berbeda oleh aplikasinya.

Berikut merupakan tampilan keyword list dari SMS *blocker*:



Gambar 6 : *Keyword list* SMS *blocker*

Keyword inilah nantinya yang akan menjadi *pattern* untuk dilakukan pencocokan pada isi pesan. *Keyword list* ini bisa kita tambahkan definisi kata-kata yang akan diblokir.



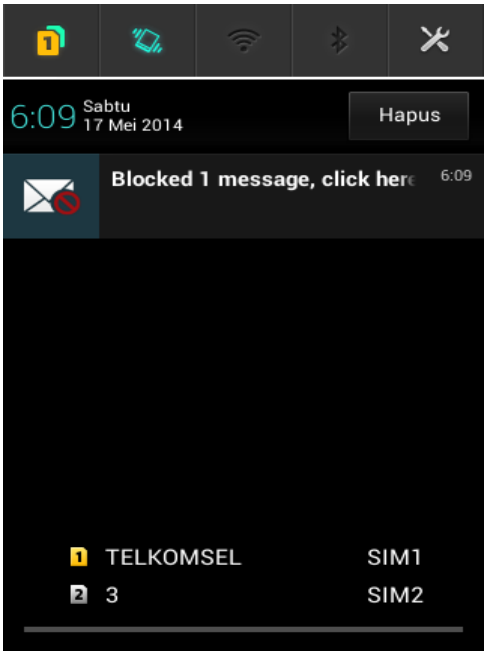
Gambar 7 : Penambahan *keyword*

Dengan bisa ditambahkannya *keyword* pada SMS *blocker* ini sangat membantu kita untuk mem*filter* SMS yang masuk ke telepon seluler yang kita miliki.

SMS *blocker* ini akan aktif bersamaan dengan aplikasi SMS bawaan pada telepon seluler yang kita miliki. Ketika ada pesan masuk maka SMS *blocker*

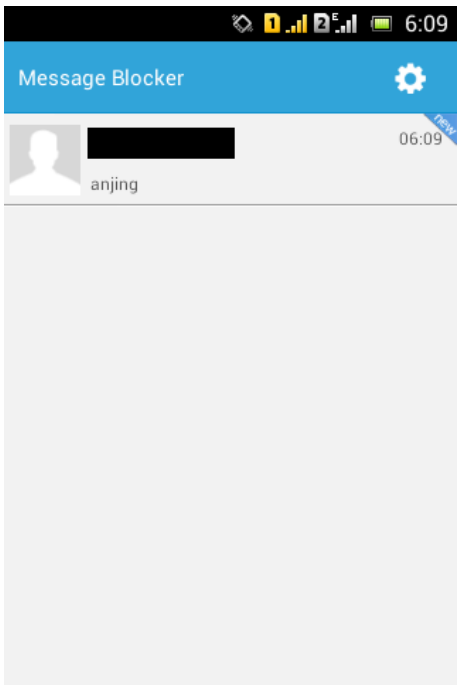
akan langsung bekerja dengan mengecek *pattern* pada *keyword list* yang kemudian dilakukan pencocokan *string* terhadap isi pesan. Algoritma yang diterapkan pada *pattern matching* isi SMS ini adalah algoritma Boyer-Moore. Dengan alfabet yang banyak, maka algoritma Boyer-Moore akan lebih baik kinerjanya.

Ketika melakukan *string matching* terhadap isi pesan, jika ditemukan kata-kata yang didefinisikan pada *keyword list* maka SMS *blocker* akan mengeluarkan notifikasi di layar telepon seluler. Notifikasinya seperti gambar dibawah :



Gambar 8 : notifikasi SMS *blocker*

Selanjutnya pesan yang ter*filter* akan diletakkan di folder terpisah dengan pesan masuk. Dipisahkannya pesan yang mengandung unsur negatif ini untuk memudahkan kita memilah pesan yang kita terima. Jika pesan tidak mengandung unsur negatif maka akan masuk ke kotak masuk, jika tidak maka akan masuk ke folder SMS *blocker*. Berikut tampilannya :



Gambar 9 : Folder khusus pesan yang diblokir

Pesan yang sudah dipisahkan ini bisa dilihat pada folder SMS *blocker*. Kita bisa memilih apa ingin melihat isi pesan tersebut atau kita ingin langsung menghapusnya.

Lalu kenapa pesan yang mengandung unsur negatif tidak langsung dihapus saja? Karena bisa jadi pengguna masih membutuhkan informasi yang terdapat dalam pesan tersebut. Lalu kemudian bisa menghapusnya sesudah itu.

IV. KESIMPULAN

String matching banyak kita temukan pengaplikasiannya dalam kehidupan sehari-hari. Algoritma *string matching* berguna untuk mencari *string* atau *pattern* tertentu dalam sebuah teks yang lebih panjang. Algoritma *brute force*, Knuth-Morris-Pratt (KMP), dan algoritma Boyer-Moore merupakan beberapa contoh dari algoritma *string matching* ini.

String matching dapat digunakan untuk memvalidasi pesan (SMS) pada telepon seluler. Pesan yang diterima kemudian dilakukan *string matching* berdasarkan *keyword list* yang telah didefinisikan. SMS yang mengandung *keyword list* yang telah didefinisikan sendiri oleh pengguna kemudian akan diblokir dan dipindahkan ke folder blokir. Kemudian di folder blokir tersebut kita bisa memilih apakah ingin melihat isi pesan tersebut ataupun langsung menghapus pesan tersebut.

DAFTAR REFERENSI

[1] Slide Presentasi Pencocokan String, Bahan kuliah IF2211 Strategi Algoritma, Teknik Informatika ITB

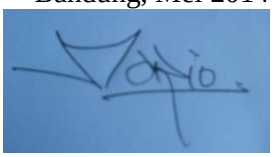
[2] Munir, Rinaldi, 2009, *Diktat Kuliah IF2211 Strategi Algoritma*, Bandung, Informatika Bandung.

[3] Lecroq, Thierry Charras, Christian. 2001. Handbook of Exact String Matching Algorithm.

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, Mei 2014



Mario Tressa Juzar - 13512016