Penerapan Regex dan String Matching untuk Filter Chat pada E-Commerce

Thea Josephine Halim - 13522012
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522012@mahasiswa.itb.ac.id

***Abstrak*—Dalam era digital yang terus berkembang, e-commerce telah menjadi sektor penting yang mengalami pertumbuhan signifikan. Aplikasi belanja online memungkinkan pengguna untuk melakukan transaksi dengan mudah dan cepat. Namun, dengan meningkatnya penggunaan platform ini, tantangan seperti pencegahan transaksi di luar platform yang dapat merugikan penyedia layanan juga muncul. Untuk menjaga keamanan dan integritas transaksi, banyak platform e-commerce mengimplementasikan filter chat yang mendeteksi dan memblokir penyebaran informasi pribadi seperti nomor telepon, alamat, dan rincian rekening bank, hingga penggunaan kata-kata vulgar.**

Kata Kunci—Filter Chat, E-Commerce, Regex, Pencocokan String

Abstract*—*In the rapidly evolving digital era, e-commerce has become a crucial sector experiencing significant growth. Online shopping applications enable users to conduct transactions easily and quickly. However, with the increasing use of these platforms, challenges such as preventing off-platform transactions that can harm service providers have also emerged. To maintain transaction security and integrity, many e-commerce platforms implement chat filters that detect and block the sharing of private information such as phone numbers, addresses, and bank account details, as well as the use of vulgar words.

Keywords—Chat Filter, E-Commerce, Regex, String Matching

I. PENDAHULUAN

Di dunia maya, kita bisa bertemu banyak orang dengan berbagai latar belakang dan sifat, menjadikan salah satu daya tarik dari kehidupan online. Kita bisa berkomunikasi dengan orang lain tanpa batas dari balik layar. Komunikasi dengan orang asing secara anonim dalam chat (pesan teks) sudah menjadi biasa, dan orang-orang bebas menulis apa saja. Sayangnya, kebebasan ini bisa saja disalahgunakan.

Dalam era digital yang terus berkembang, e-commerce atau aplikasi bisnis online telah menjadi salah satu sektor yang mengalami pertumbuhan pesat karena memungkinkan pengguna untuk melakukan transaksi dengan mudah dan cepat tanpa harus repot-repot mengunjungi toko, dapat dilakukan kapan saja, dan di mana saja. Akan tetapi, seiring dengan meningkatnya penggunaan platform ini, timbul pula berbagai tantangan, salah satunya adalah upaya untuk mencegah transaksi di luar aplikasi yang dapat merugikan penyedia layanan maupun membahayakan konsumen. Untuk menjaga keamanan dan integritas transaksi, banyak platform e-commerce mengimplementasikan filter chat yang mendeteksi dan memblokir penyebaran informasi pribadi seperti nomor telepon, alamat, dan informasi rekening bank. Pentingnya chat filtering ini guna melindungi pengguna dari potensi penipuan dan memastikan semua transaksi tetap berada dalam sistem aplikasi yang mencegah kerugian pihak e-commerce sendiri. Selain itu, filter chat pada e-commerce juga memastikan agar chat antara penjual dan pembeli tetap sopan dengan filter chat yang dapat mendeteksi kata-kata vulgar ataupun mengandung pornografi.

Gambar 1.1 Ilustrasi Kasus Pelanggaran Chat
Sumber: Seller Shopee Edu

Salah satu primadona e-commerce akhir-akhir ini adalah Shopee, aplikasi jual beli online asal Singapura yang dimiliki perusahaan Sea Limited. Pada makalah ini akan dibahas mengenai penggunaan RegEx (Regular Expression) dan algoritma *pattern matching* pada simulasi *chat filtering* sederhana pada pesan teks antara penjual dan pembeli di e-commerce Shopee. Berdasarkan petunjuk kebijakan pelayanan pembeli pada laman pusat edukasi penjual Shopee, pesan yang dianggap melanggar meliputi:

1) Mengirim *chat* yang tidak pantas, kasar, atau mengandung unsur SARA (suku, agama, ras, dan antar golongan).
2) Mengirim *chat* pornografi atau pelecehan seksual.
3) Mengirim *chat* yang bersifat mengancam.
4) Meminta Pembeli untuk membatalkan pesanan.

5) Mengarahkan Pembeli untuk bertransaksi di luar aplikasi/situs Shopee.

6) Melakukan *spam chat*.

Bahan analisis pada makalah ini akan dibatasi pada implementasi *chat filtering* sederhana berupa deteksi kata-kata kasar ataupun mengandung pornografi, permintaan untuk membatalkan pesanan, dan niat mengarahkan pembeli pada transaksi di luar aplikasi dengan adanya *sharing* informasi pribadi (email, sosial media, nomor rekening, nomor telepon, dan website).

II. LANDASAN TEORI

A. *Regular Expression (Regex)*

Regular Expression atau biasa disebut dengan regex adalah semacam pola yang digunakan untuk mencari dan memanipulasi teks berdasarkan aturan tertentu. Regex memungkinkan identifikasi dan pencocokan pola teks yang kompleks dan beragam, seperti alamat email, nomor telepon, atau pola teks khusus lainnya. Regex terdiri dari karakter normal dan karakter khusus (meta-characters) yang bersama-sama membentuk pola pencarian. Regex mampu melakukan pencocokan pola teks dengan efisiensi yang tinggi untuk menentukan apakah suatu teks diterima dengan aturan yang ada.

Beberapa contoh pendefinisian pola regex adalah sebagai berikut:

1) Karakter

   Semua karakter, mulai dari angka 0-9, huruf a-z termasuk varian huruf besar dan huruf kecil, serta substring karakter lainnya.

2) Metakarakter dan Karakter Spesial

   Karakter khusus yang memiliki arti khusus di regex terdiri dari ., +, *, ?, ^, $, (,), [,], {, }, |, \.

   - Tanda titik "." : cocok dengan karakter apapun. Contoh regex .uku akan cocok dengan kuku, Duku, buku, dan tuku.
   - Tanda "|" : menyatakan OR
   - Tanda "^" : menyatakan negasi

3) Kelas karakter

Construct	Deskripsi
[abc]	a, b, atau c (simple class)
[^abc]	Semua karakter **selain** a,b,c (negasi)
[a-zA-Z]	a sampai z atau A sampai Z, inclusive (range)
[a-d[m-p]]	a sampai d atau m sampai p (gabungan)
[a-z&&[def]]	d, e atau f (irisan)
[a-z&&[^bc]]	a sampai z, kecuali b dan c (subtraksi)
[a-z&&[^m-p]]	a sampai z, dan bukan m sampai p (subtraksi)

Tabel 2.3.1 Konstruksi Regex Character Class
Sumber: Modul Praktikum Regex

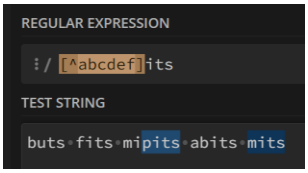
a) Simple class

Karakter dalam kurung siku, akan dicari yang cocok dengan substring salah satu karakter dari a,b,c,d,e,f dengan substring lanjutan "its".

Gambar 2.1.3.1 Contoh Simple Class
Sumber: Dokumentasi Penulis

b) Negasi

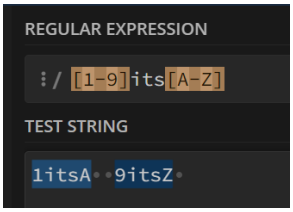
Karakter "^" menandakan negasi, substring yang diikuti "its" tidak boleh mengandung substring "a,b,c,d,e,f" pada bagian awalnya.



Gambar 2.1.3.2 Contoh Penggunaan Negasi
Sumber: Dokumentasi Penulis

c) Range

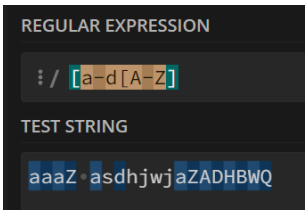
Karakter "-" menandakan range karakter pada suatu rentang. Substring "its" harus diawali dengan salah satu karakter angka dari rentang 1-9 dan diakhiri karakter A-Z.



Gambar 2.1.3.3 Contoh Penggunaan Range
Sumber: Dokumentasi Penulis

d) Union

Gabungan dari range, Gambar 2.3.4 menunjukkan ilustrasi union berupa gabungan dari huruf a-d dengan akhiran karakter A-Z



Gambar 2.1.3.4 Contoh Penggunaan Union
Sumber: Dokumentasi Penulis

4) Predefined Character Class

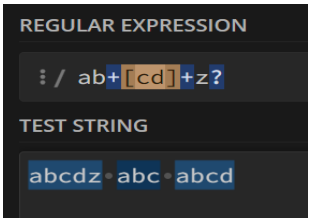
Construct	Deskripsi
.	Semua karakter
\d	Digit [0-9]
\D	Non digit [^0-9]
\s	Whitespace character [\t\n\x0B\f\r]
\S	Non whitespace character [^s]
\w	Word character [a-zA-Z_0-9]
\W	Non word character [^\w]

Tabel 2.1.4.1 Predefined Character Class
Sumber: Modul Praktikum Regex

5) Quantifier dan Repetition Operator

Construct	Deskripsi
X?	X muncul satu atau tidak sama sekali
X*	X muncul nol atau banyak
X+	X muncul satu atau banyak
x{n}	X muncul tepat n kali
x{n,}	X muncul setidaknya n kali
x{n,m}	X muncul antara n sampai m kali

Tabel 2.1.5.1 Regex Quantifiier dan Operator
Sumber: Modul Praktikum Regex



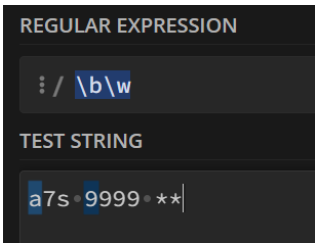
Gambar 2.1.5.1 Contoh Penggunaan Quantifier dan Operator
Sumber: Dokumentasi Penulis

6) Boundary Matchers

Pencocokan posisi seperti awal kalimat, awal kata, akhir kata.

Construct	Deskripsi
^	Awal baris
$	Akhir baris
\b	Batas kata
\B	Batas bukan kata
\G	Akhir match sebelumnya
\Z	Akhir dari input tapi untuk final terminator jika ada
\z	Akhir dari input

Tabel 2.1.6.1 Regex Boundary Matchers
Sumber: Modul Praktikum Regex



Gambar 2.1.6.1 Contoh Penggunaan Regex Positioning

Sumber: Dokumentasi Penulis

Dengan penerapan pola karakter regex dan operator, kita bisa membuat sebuah aturan yang akan menentukan apakah suatu string diterima atau tidak.

B. Konsep Singkat String

Sebelum membahas algoritma pengecekan string, perlu dipahami konsep dasar dari sebuah string. Pada string S dengan panjang n kita akan memiliki:

$$S = x_0\, x_1 \ldots x_{m-1}$$

Maka kita memiliki prefix (substring S[0…k]) dan suffix (substring S[k…m-1]), dengan k indeks di antara 0 dan m-1.

C. Pattern/String Matching

String matching atau *pattern matching*, adalah proses mencari kemunculan suatu pola (pattern) tertentu dalam sebuah teks (text). Tujuan utama dari string matching adalah untuk menemukan semua posisi dalam teks di mana pola muncul. String matching adalah fundamental dalam berbagai aplikasi komputasi seperti pencarian teks, analisis DNA, analisis citra sidik jari, pemrosesan bahasa alami, dan pencarian data di basis data. Proses ini bisa dilakukan menggunakan berbagai metode, baik yang sederhana seperti brute-force hingga yang lebih kompleks dan efisien seperti algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM).

Dalam string matching, akan terdefinisi sebuah teks (T) berupa string panjang yang akan menjadi lokasi pencarian kita dengan panjang n karakter, serta sebuah pattern (P) berupa string dengan panjang m karakter (m<<<n) dan akan dicari dalam teks (T) yang ada. Pada string matching kita ditugaskan

untuk menentukan apakah pattern ditemukan dalam teks, dan mencari lokasi di mana ditemukannya pertama kali.

Algoritma string matching dapat diklasifikasikan ke dalam dua kategori utama: algoritma eksak dan algoritma aproksimasi. Algoritma eksak mencari kecocokan sempurna antara pola dan teks (*exact match*), sedangkan algoritma aproksimasi mengizinkan beberapa perbedaan atau kesalahan antara dua string. Kesalahan ini bisa berupa penyisipan, penghapusan, atau substitusi karakter. Salah satu contoh dari algoritma pencocokan aproksimasi adalah Levenshtein Distance. Algoritma ini sangat berguna dalam aplikasi di mana kesalahan atau variasi dalam data sangat umum, seperti dalam pengenalan suara, analisis DNA, pemrosesan bahasa alami, dan pencarian teks yang tidak terstruktur. Dalam algoritma pencocokan aproksimasi akan digunakan dua metrik pengukur:

1) Jarak Edit: Jarak minimum antara dua string yang diukur berdasarkan operasi penyisipan, penghapusan, atau substitusi.

2) Persentase Kesalahan (Toleransi Kesalahan): Proporsi kesalahan yang diizinkan.

Untuk pencocokan *exact*, algoritma klasik yang sering digunakan adalah algoritma brute-force (kurang efektif), KMP, dan BM.

1) Brute Force (BF)

Brute Force adalah algoritma pencocokan string *exact* paling sederhana yang akan melakukan pengecekan pada setiap huruf pada teks dengan setiap huruf pada pattern. Sifatnya yang mengecek setiap posisi dalam teks menjadikannya memiliki kompleksitas waktu O(n*m), terjadi saat karakter pertama hingga dua dari terakhir sama terus-menerus dengan teks, sedangkan kasus terbaiknya O(n) terjadi ketika karakter pertama pattern selalu berbeda dengan teks. Cara kerjanya sebagai berikut:

Teks: NOBODY NOTICED HIM
Pattern: NOT

```
  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

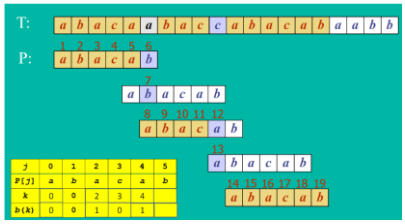
Gambar 2.3.1.1 Pencocokan Brute Force
Sumber: Website Homepage Rinaldi Munir

a) Mulai dari karakter pertama dalam teks.

b) Bandingkan karakter pertama pola dengan karakter teks saat ini. Jika karakter cocok, lanjutkan ke karakter berikutnya dari pola dan teks. Jika karakter tidak cocok, geser pola satu posisi ke kanan dalam teks dan ulangi proses pencocokan karakter pertama pola.

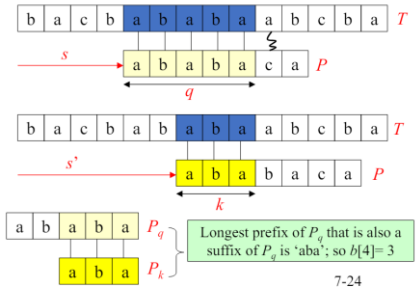
c) Jika seluruh pola berhasil dicocokkan dengan bagian dari teks, maka kecocokan ditemukan.

2) Knuth Morris Pratt (KMP)



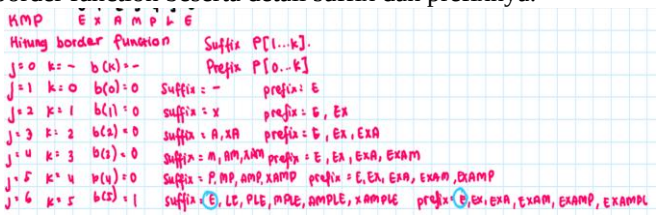
Gambar 2.3.2.1 Pencocokan Knuth Morris Pratt
Sumber: Website Homepage Rinaldi Munir

Knuth Morris Pratt adalah algoritma pencocokan string *exact* yang menggunakan tabel *longest prefix suffix* (LPS) atau *Border Function* untuk menentukan indeks posisi pengecekan pada pattern berikutnya serta menghindari pemeriksaan ulang pattern. Pengecekannya mirip dengan Brute Force, mengecek pattern dari kiri ke kanan, tetapi dengan batasan tertentu dengan pembuatan LPS. LPS adalah jumlah prefix dan suffix sama terpanjang yang bisa didapatkan pada *missmatch pattern* pada indeks tertentu. Ilustrasi proses pembentukan tabel LPS dengan i indeks *missmatch* pada teks dan j indeks *missmatch* pada pattern, serta k adalah j-1 adalah sebagai berikut:



Gambar 2.3.2.1 Cara Pembentukan Tabel LPS
Sumber: Website Homepage Rinaldi Munir

Berikut adalah salah satu contoh hasil penentuan border function beserta detail suffix dan prefixnya:



Gambar 2.3.2.2 Pembentukan Tabel LPS
Sumber: Website Homepage Rinaldi Munir

Setelah dibuat tabel LPS, pencocokan karakter akan dilakukan sebagai berikut:

a) Mulai dari karakter pertama dalam teks dan pola.

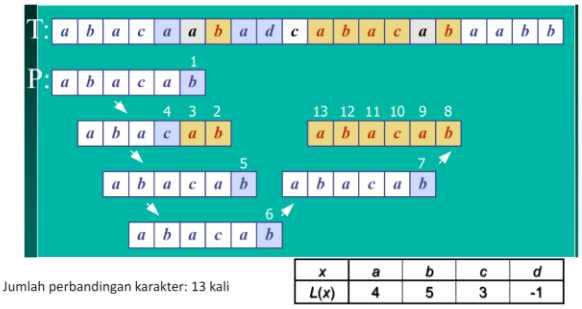
b) Bandingkan karakter dari pola dengan teks. Jika karakter cocok, lanjutkan ke karakter berikutnya. Jika karakter tidak cocok dan bukan karakter pertama dari pola, gunakan nilai LPS untuk menentukan posisi

pencocokan berikutnya.

c) Jika karakter cocok sampai akhir pola, maka kecocokan ditemukan.

Dalam menghitung tabel LPS dibutuhkan waktu O(m) dan proses pencarian string memakan waktu O(n), sehingga kompleksitas waktu KMP adalah O(m+n). Akan tetapi, KMP tidak terlalu bagus untuk kasus pencocokan string dengan variasi karakter pada teks yang banyak sebab memungkinkan banyak *missmatch* di awal dan membuatnya hanya bergeser 1 indeks dari teks. KMP lebih cepat jika *missmatch* terjadi di akhir pattern.

3) Boyer Moore (BM)



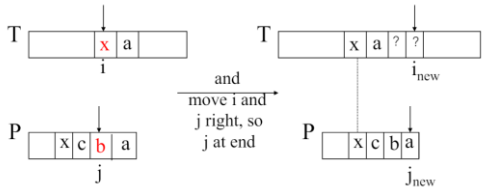
Gambar 2.3.3.1 Pencocokan Boyer Moore
Sumber: Website Homepage Rinaldi Munir

Boyer Moore adalah algoritma pencocokan string *exact* menggunakan tabel *last occurrence* (indeks terakhir munculnya karakter pada pattern) dengan 2 teknik:

a) Looking-glass technique
Pencocokan dilakukan dari akhir pattern hingga awal pattern.

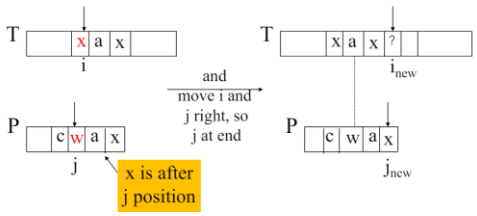
b) Character-jump technique
Terjadi *missmatch* pada teks indeks ke-i dan pattern karakter ke-j tidak sama dengan teks indeks ke-i. Dengan menetapkan karakter x sebagai karater pada teks tempat terjadi *missmatch*, pada teknik ini akan dibagi menjadi 3 kasus:

- Karakter x terakhir kali muncul pada pattern pada sisi kiri indeks j. Penghitungan indeks i untuk pengecekan selanjutnya dilakukan dengan rumus $i_{new} = i + m - (LO -1)$



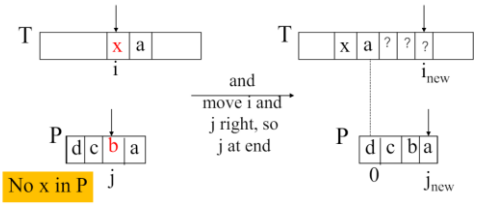
Gambar 2.3.3.2 Missmatch Case 1
Sumber: Website Homepage Rinaldi Munir

- Karakter x terakhir kali muncul pada pattern pada sisi kanan indeks j (j+1). Penghitungan indeks i untuk pengecekan selanjutnya dilakukan dengan rumus $i_{new} = i + m - j$



Gambar 2.3.3.3 Missmatch Case 2
Sumber: Website Homepage Rinaldi Munir

- Karakter x tidak pernah muncul pada pattern. Penghitungan indeks i untuk pengecekan selanjutnya dilakukan dengan rumus $i_{new} = i + m - (LO -1)$



Gambar 2.3.3.4 *Missmatch* Case 3
Sumber: Website Homepage Rinaldi Munir

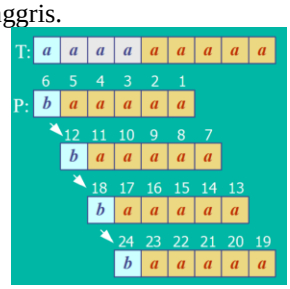
Berdasarkan peraturan 3 kasus *missmatch*, kita akan membuat tabel *last occurrence* semua huruf yang ada pada teks. Untuk huruf yang tidak pernah muncul pada pattern akan dituliskan dengan -1.

Cara pencocokan Boyer Moore dilakukan sebagai berikut:

a) Mulai dari akhir pola dan cocokkan dengan teks dari kanan ke kiri pattern. Jika karakter tidak cocok, tentukan jenis kasus *missmatch* dan tentukan indeks baru i_{new} dan lanjutkan pencarian dengan menggeser indeks terakhir pattern ke lokasi i_{new}, lalu ulangi langkah ini.

b) Jika seluruh pola cocok dengan bagian dari teks, maka kecocokan ditemukan.

Kompleksitas waktu terburuk Boyer Moore adalah O(mn+A), dengan A adalah jumlah alfabet berbeda yang ada. Kasus terburuk ini terjadi ketika *missmatch* selalu terjadi pada indeks pertama pattern, menyebabkan Boyer Moore harus melakukan pengecekan seluruh huruf yang ada pada pattern dan teks. Akan tetapi, algoritma Boyer Moore bagus untuk teks dengan variasi karakter yang banyak (A banyak) sehingga bagus untuk pencarian pada teks bahasa Inggris.



Gambar 2.3.3.5 Ilustrasi Kasus Terburuk BM
Sumber: Website Homepage Rinaldi Munir

III. PEMBAHASAN

A. Implementasi Kode

Kode-kode di bawah ini dapat diakses melalui link github berikut. Berikut adalah implementasi kode algoritma dasar Brute Force, KMP, dan BM yang menggunakan bahasa Java. Ketiga kelas akan melakukan pengecekan apakah pattern ditemukan pada teks, jika ya akan direturn indeks pertama kali ditemukannya, jika tidak akan mengembalikan angka -1.

Fungsi BFAlgo akan melakukan iterasi for loop i mulai dari indeks pertama teks hingga n-m, dengan pada tiap-tiap iterasinya dilakukan pengecekan pattern. Sebuah variable *counter counstsame* digunakan untuk menghitung jumlah kecocokan selama pengecekan patttern, apabila nilainya sudah sama dengan panjang pattern (m), maka pencarian akan dihentikan.

```java
public class BFAlgorithm {
    public static int BFAlgo(String text, String pattern){
        int n = text.length();
        int m = pattern.length();
        int idxfound = -1;
        for (int i = 0; i<=n-m; i++){
            int countsame = 0;
            for (int j = 0; j<m; j++){
                if (pattern.charAt(j)!=text.charAt(i+j)){
                    break;
                }
                countsame++;
            }
            if(countsame==m){
                System.out.println("Found it!");
                idxfound = i;
                break;
            }
        }
        return idxfound;
    }
}
```

Gambar 3.1.1 Kode Brute Force
Sumber: Dokumentasi Penulis

Metode KMPAlgo pertama-tama akan memanggil fungsi lpsValue untuk mendapatkan array of integer yang berisi border function. Selanjutnya akan dilakukan iterasi for loop i mulai dari indeks 0 hingga n dan melakukan pengecekan pattern setiap kali pergeseran indeks i, dengan setiap *start index* pada pattern ditentukan oleh tabel LPS. Apabila pada suatu saat ditemukan nilai indeks iterasi j sama dengan m-1, atau selesai mengecek semua karakter pada pattern, maka pattern sudah ditemukan dan return indeks i.

```java
public class KMPAlgorithm {
    // function to build LPS / border func table in KMP
    // will return an array of border func
    public static int[] lpsValue(String pattern) {
        System.out.println("Let's build the border function!\n");
        int[] borderfunc = new int[pattern.length()];
        int prefixbestlength = 0;
        int idx = 1; // the lps array index starts from 1 since borderfunc[0] is always 0
        borderfunc[0] = 0; // when j = 0 we always get border func 0

        while (idx < pattern.length()) {
            if (pattern.charAt(idx) == pattern.charAt(prefixbestlength)) {
                prefixbestlength++;
                borderfunc[idx] = prefixbestlength;
                idx++;
            } else {
                if (prefixbestlength > 0) { // based on matching prefix
                    prefixbestlength = borderfunc[prefixbestlength - 1];
                } else { // no match
                    borderfunc[idx] = 0;
                    idx++;
                }
            }
        }
        return borderfunc;
    }

    // main KMP algo that will return the idx found on text
    public static int KMPAlgo(String pattern, String text) {
        int n = text.length();
        int m = pattern.length();
        int[] borderfunc = lpsValue(pattern);

        // initialize starting mismatch index
        int idx_i = 0;
        int idx_j = 0;

        while (idx_i < n) {
            if (pattern.charAt(idx_j) == text.charAt(idx_i)) {
                if (idx_j == m - 1) {
                    System.out.println("Found it!");
                    return idx_i - m + 1;
                }
                idx_i++;
                idx_j++;
            } else {
                if (idx_j > 0) {
                    idx_j = borderfunc[idx_j - 1];
                } else {
                    idx_i++;
                }
            }
        }
        System.out.println("We can't find the pattern");
        return -1;
    }
}
```

Gambar 3.1.2 Kode Knuth Morris Pratt
Sumber: Dokumentasi Penulis

Metode BMAlgo pertama-tama akan memanggil fungsi lastoccur untuk mendapatkan array of integer yang berisi last occurrence untuk setiap karakter yang ada pada ASCII. Selanjutnya akan dilakukan iterasi for loop i mulai dari indeks 0 hingga n-1 dan melakukan pengecekan pattern, apabila terjadi *missmatch* maka akan digunakan nilai lastoccurence untuk menentukan indeks i pencarian selanjutnya. Apabila pada suatu saat ditemukan nilai indeks iterasi j aadalah 0, atau selesai mengecek semua karakter pada pattern, maka pattern sudah ditemukan dan return indeks i.

Gambar 3.1.3 Kode Boyer Moore
Sumber: Dokumentasi Penulis

B. *Filter Kata-Kata Kasar*

Dalam penanganan filter kata-kata kasar ini akan digunakan sebuah kamus yang memuat kata-kata kasar yang kerap digunakan masyarakat Indonesia. Kata-kata pada kamus tersebut akan menjadi pattern, dan kita akan melakukan pencarian dengan KMP dan BM pada teks. Algoritma yang terdefinisi ada pada parameter algo.

Gambar 3.2.1 Kode Filter Kata-Kata Tidak Senonoh
Sumber: Dokumentasi Penulis

C. *Filter Pengajuan Pembatalan*

Filter pengajuan pembatalan dilakukan dengan cara yang sama seperti filter kata-kata kasar. Pertama kali akan dilakukan pembacaan file kamus terdefinisi, kemudian akan dilakukan iterasi pada teks untuk mencari kemunculan pattern dari kamus berdasarkan algoritma yang terpilih. Algoritma yang digunakan akan terdefinisi pada parameter algo.

Gambar 3.1.3 Kode Filter Cancel
Sumber: Dokumentasi Penulis

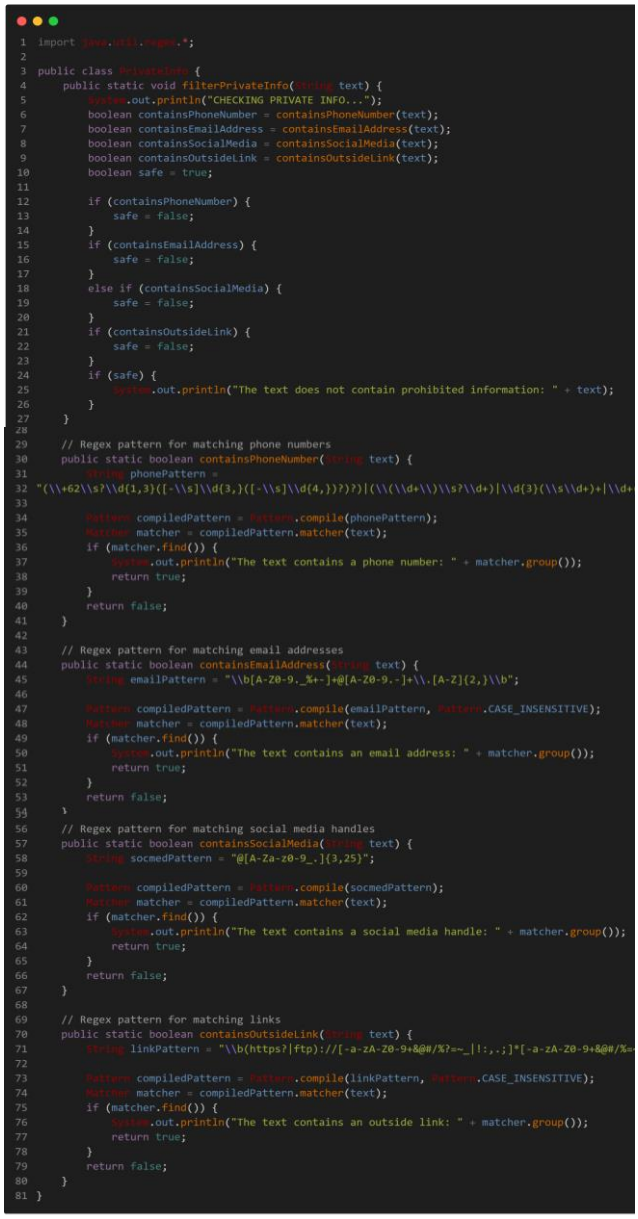
D. *Translate*

Sudah tidak asing adanya orang-orang yang menggunakan angka sebagai pengganti huruf, seperti huruf “e” yang digantikan oleh angka 3, huruf “s” digantikan oleh angka 5, dan seterusnya. Maka dari itu, sebelum dilakukan pemrosesan *pattern match*, akan dilakukan terjemahan ke bentuk aslinya.

Gambar 3.4.1 Kode Translate
Sumber: Dokumentasi Penulis

E. Filter Sharing Informasi Pribadi

Dalam filtering informasi pribadi, akan dilakukan empat kali pengecekan terpisah untuk nomor telepon, alamat email, username sosial media, dan tautan. Setiap pengecekan akan dipisah supaya bisa diketahui penyebab ditolaknya suatu teks. Fungsi filterPrivateInfo akan menyatukan seluruh fungsi pengecekan dan mengeluarkan output yang sesuai dengan hasil pengecekan. Pada pengecekan email dan username sosial media ditambahkan pada satu blok *if conditional* yang sama karena email pasti mengandung karakter "@". Berikut adalah detail peraturan regex yang digunakan:



```
import java.util.regex.*;

public class PrivateInfo {
    public static void filterPrivateInfo(String text) {
        System.out.println("CHECKING PRIVATE INFO...");
        boolean containsPhoneNumber = containsPhoneNumber(text);
        boolean containsEmailAddress = containsEmailAddress(text);
        boolean containsSocialMedia = containsSocialMedia(text);
        boolean containsOutsideLink = containsOutsideLink(text);
        boolean safe = true;

        if (containsPhoneNumber) {
            safe = false;
        }
        if (containsEmailAddress) {
            safe = false;
        }
        else if (containsSocialMedia) {
            safe = false;
        }
        if (containsOutsideLink) {
            safe = false;
        }
        if (safe) {
            System.out.println("The text does not contain prohibited information: " + text);
        }
    }

    // Regex pattern for matching phone numbers
    public static boolean containsPhoneNumber(String text) {
        String phonePattern =
"(\\+62\\s?\\d{1,3}([-\\s]\\d{3,}([-\\s]\\d{4,})?)?)|(\\(\\d+\\)\\s?\\d+)|\\d{3}(\\s\\d+)+|\\d+

        Pattern compiledPattern = Pattern.compile(phonePattern);
        Matcher matcher = compiledPattern.matcher(text);
        if (matcher.find()) {
            System.out.println("The text contains a phone number: " + matcher.group());
            return true;
        }
        return false;
    }

    // Regex pattern for matching email addresses
    public static boolean containsEmailAddress(String text) {
        String emailPattern = "\\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,}\\b";

        Pattern compiledPattern = Pattern.compile(emailPattern, Pattern.CASE_INSENSITIVE);
        Matcher matcher = compiledPattern.matcher(text);
        if (matcher.find()) {
            System.out.println("The text contains an email address: " + matcher.group());
            return true;
        }
        return false;
    }

    // Regex pattern for matching social media handles
    public static boolean containsSocialMedia(String text) {
        String socmedPattern = "@[A-Za-z0-9_.]{3,25}";

        Pattern compiledPattern = Pattern.compile(socmedPattern);
        Matcher matcher = compiledPattern.matcher(text);
        if (matcher.find()) {
            System.out.println("The text contains a social media handle: " + matcher.group());
            return true;
        }
        return false;
    }

    // Regex pattern for matching links
    public static boolean containsOutsideLink(String text) {
        String linkPattern = "\\b(https?|ftp)://[-a-zA-Z0-9+&@#/%?=~_|!:,.;]*[-a-zA-Z0-9+&@#/%=

        Pattern compiledPattern = Pattern.compile(linkPattern, Pattern.CASE_INSENSITIVE);
        Matcher matcher = compiledPattern.matcher(text);
        if (matcher.find()) {
            System.out.println("The text contains an outside link: " + matcher.group());
            return true;
        }
        return false;
    }
}
```

Gambar 3.5.1 Kode Filter Informasi Pribadi
Sumber: Dokumentasi Penulis

1) Untuk pengecekan pola nomor telepon ada pada deklarasi variable phonePattern: *(\\+62\\s?\\d{1,3}([-\\s]\\d{3,}([\\s]\\d{4,})?)?)|(\\(\\d+\\)\\s?\\d+)|\\d{3}(\\s\\d+)+|\\d+([-\\s]\\d+)+|\\d{8,12}.* Pola ini juga akan menangani pemisahan dengan spasi dan "-" untuk mencegah usaha *bypassing.* Regex ini akan dibatasi untuk nomor telepon Indonesia saja.

Potongan Regex	Deskripsi
\\+62\\s?\\d{1,3}	Menangkap nomor telepon yang dimulai dengan kode negara Indonesia (+62), diikuti oleh 1 hingga 3 digit. \\s? berarti ada atau tidak ada spasi.
([-\\s]\\d{3,}([-\\s]\\d{4,})?)?	Mengizinkan format dengan tanda hubung atau spasi, misalnya -3456 atau -3456-7890.
(\\(\\d+\\)\\s?\\d+)	Menangkap nomor telepon dengan format kode area dalam kurung, misalnya (021) 123456.
\\d{3}(\\s\\d+)+	Menangkap format nomor telepon dengan grup tiga digit diikuti oleh spasi dan digit lainnya, misalnya 123 4567 8901.
\\d+([-\\s]\\d+)+	Menangkap nomor telepon dengan tanda hubung atau spasi, misalnya 123-4567-8901.
\\d{8,12}	Menangkap nomor telepon dengan panjang minimal 8 digit karena nomor telepon Indonesia dimulai dari 8 digit dan maksimal 12 digit, misalnya 12345678 dan 467321875

Tabel 3.5.1 Penjelasan Regex Nomor Telepon

2) Untuk pengecekan pola email ada pada deklarasi variabel emailPattern: \\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,}\\b

Potongan Regex	Deskripsi
[A-Z0-9._%+-]+	Menangkap karakter yang valid dalam bagian username email (huruf, angka, titik, underscore, persen, plus, dan tanda minus).
@[A-Z0-9.-]+	Menangkap karakter yang valid setelah simbol @ (huruf, angka, titik, dan tanda minus).
\\.[A-Z]{2,}	Menangkap domain level atas yang dimulai dengan titik diikuti oleh minimal dua huruf, misalnya .com dan .id

Tabel 3.5.2 Penjelasan Regex Alamat Email

3) Untuk pengecekan pola username sosial media ada pada deklarasi variabel socmedPattern: @[A-Za-z0-9_.]{3,25}

Potongan Regex	Deskripsi
@	Menangkap simbol @ yang umumnya digunakan dalam username media sosial, misalnya @regexinstaofficial
[A-Za-z0-9_.]{3,25}	Menangkap karakter yang valid dalam handle media sosial (huruf, angka, titik, dan underscore) dengan panjang antara 3 hingga 25 karakter.

Tabel 3.5.3 Penjelasan Regex Username Sosial Media

4) Untuk pengecekan pola link ada pada deklarasi

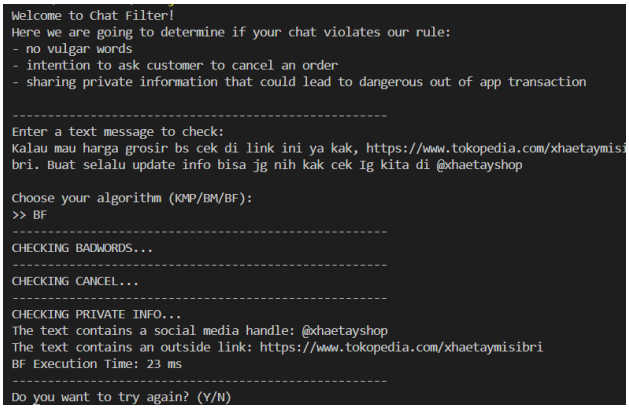
variabel linkPattern: \\b(https?|ftp)://[-a-zA-Z0-9+&@#/%?=~_|!:,.;]*[-a-zA-Z0-9+&@#/%=~_|]\\b
Pola ini juga akan menangani pemisahan dengan spasi dan karakter lain untuk mencegah usaha *bypassing*.

Potongan Regex	Deskripsi
(https?|ftp)://	Menangkap protokol yang umum digunakan (http, https, ftp).
[-a-zA-Z0-9+&@#/%?=~_|!:,.;]*	Menangkap karakter yang valid dalam URL (huruf, angka, dan berbagai simbol yang umum digunakan dalam URL).
[-a-zA-Z0-9+&@#/%=~_|]	Menangkap karakter yang valid di akhir URL, misalnya path/to/resources

Tabel 3.5.4 Penjelasan Regex Tautan

F. *Perbandingan dan Analisis Algoritma*

Gambar 3.6.1 Contoh Hasil Kompilasi Program
Sumber: Dokumentasi Penulis



Gambar 3.6.2 Contoh Hasil Kompilasi Program 2
Sumber: Dokumentasi Penulis

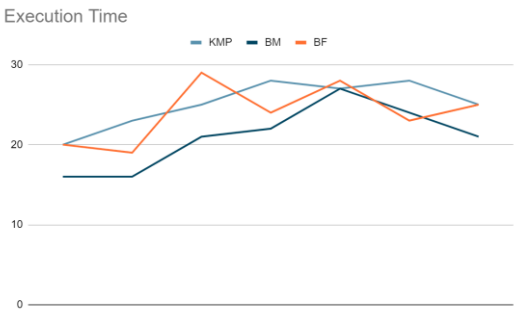
Pada program filter chat ini akan dilakukan pengujian pada algoritma KMP dan BM. Pengujian akan dilakukan dengan inputan yang sama.

Test Case	Teks Input
1.	Hi welcome to my humble shop! For far more cheap prices you can contact me thru WAs 084 3984 3287 or at email@mail.com thank you!
2.	Kalau emang gasuka yaudah batalin aja ngapa, sulit banget dih t0l0l.
3.	Kalau mau harga grosir bs cek di link ini ya kak, di https://www.tokopedia.com/xhaetaymisibri. Buat selalu update info bisa jg nih kak cek Ig kita di @xhaetayshop. Di sana banyak diskon juga kok buat pembelian per pcs jadi ajak teman kakak ya! Ajak 1 teman bisa dapat promo buy 2 get 1 di pembelian berikutnya dengan menyertakan foto bukti struk pembelian. Yang pesanan ini boleh dibatalin dulu aja gapapa kok.
4.	Maaf Kak ini kok barangnya sepertinya ada yang rusak ya. Waktu ak buka kok ada robek di bagian lengan bajunya, nih ak ada video unboxing buat buktinya. Aku ajukan claim ya kak? Ongkir yang tanggung saya saja tidak apa -- Heh cok jangan ngarang ya, barang kita ini udh dicek sblm ngirim!
5.	Pagi kak, Oya kak maaf sekali boleh tolong cancel dl orderannya tidak ya? Stok kami lagi habis, boleh nnt chat aja ke (+62) 857 999 000 biar dikasih harga spesial buat kakak, pesan lewat situ saja
6.	Tulisan di deskripsi tuh dibaca! Batalin nih pesanan! Sudah jelas-jelas ditulis kalau minimal beli itu 10 pcs kok malah Cuma beli 1 biji. Ya kita rugi dong! Rada T0l0l emang ini
7.	Kakak boleh nih cek toko kita yang lain lagi di https://www.tokopedia.com/getmorebeauty kami baru launching toko di sana tuh jadi banyak promonya!

Tabel 3.6.1 Test Case dan Input

Lama Eksekusi (ms)				
Test Case	**Panjang Kata**	**KMP**	**BM**	**BF**
1.	129	23	16	19
2.	69	20	16	20
3.	410	25	21	25
4.	287	28	24	23
5.	196	27	27	28
6.	177	28	22	25
7.	146	25	21	29
Rata-Rata	202	25.14	21	24.14

Tabel 3.6.2 Waktu Eksekusi



Gambar 3.6.3 Grafik Hasil Kompilasi Program
Sumber: Dokumentasi Penulis

Berdasarkan hasil pengujian waktu eksekusi algoritma, didapatkan bahwa algoritma BM menjadi yang paling optimal di antara ketiganya, diikuti dengan algoritma BM di posisi kedua dan KMP di posisi ketiga. Sifat algoritma BM yang unggul dalam pencocokan string dengan banyak jenis karakter membuatnya mudah dalam bergeser indeks, berbeda dengan

algoritma KMP yang melakukan pengecekan mirip BF, tetapi dengan bantuan tabel LPS. Banyaknya macam karakter pada string akan membuat KMP lebih sering *missmatch* di indeks awal pattern, padahal KMP akan jauh lebih efisien apabila indeks *missmatch* ada pada bagian akhir pattern. *Missmatch* yang banyak terjadi di awal menyebabkan KMP bekerja seperti BF dengan bergeser indeks teks satu per satu dan mengecek pattern mulai dari indeks ke-0. Ditambah dengan perlunya *preprocessing* membangun tabel LPS membuat *overhead* pada algoritma KMP sehingga menjadikannya lebih lambat daripada BF, terlebih pada panjang string yang tidak panjang. Pada algoritma BM sebenarnya juga terdapat *overhead* karena harus *preprocessing* tabel *last occurrence*, tetapi karena keuntungannya dalam melakukan loncatan jauh membuatnya mengungguli algoritma BF pada kebanyakan kasus. Maka dari itu, BF mungkin jadi lebih tepat untuk kasus pencarian string pendek karena tidak memerlukan *preprocessing*.

IV. KESIMPULAN

Dalam pengecekan pola informasi pribadi seperti email, nomor telepon, links, dan username sosial media dibutuhkan Regex karena sifatnya yang fleksibel dan proses deteksinya tergantung dengan aturan. Berbeda dengan pengecekan kata-kata kasar dan pengajuan pengembalian, kita gunakan algoritma KMP dan BM karena sifatnya yang mencocokkan secara eksak dengan kata-kata yang sudah terdefinisi dalam sebuah kamus. Pada pencocokan string, algoritma BM jauh lebih efektif daripada KMP karena sifat teks yang memiliki banyak macam huruf. *Overhead* pada pembangunan tabel LPS pada KMP dan *last occurrence* pada BM dapat menyebabkan kinerja yang lebih lambat daripada BF. Maka dari itu, terdapat beberapa kasus di mana BF menjadi lebih cepat daripada KMP dan BM, terutama ketika panjang kata yang pendek.

UCAPAN TERIMA KASIH

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa atas penyertaan dan berkat yang telah diberikan selama ini sehingga makalah yang berjudul "Penerapan Regex dan String Matching untuk Filtering Informasi Sensitif dalam Pesan Chat" ini dapat diselesaikan dengan baik. Saya juga ingin mengucapkan terima kasih pada pihak-pihak yang telah membantu dalam penyusunan makalah ini:

1. Ibu Dr. Nur Ulfa Maulidevi, selaku dosen pengajar mata kuliah IF2211 Strategi Algoritma K02, atas bimbingan dan pengajaran yang sangat membantu saya dalam memahami materi,
2. Orang tua saya yang selalu mendukung dan membantu saya selama ini,
3. Teman-teman dan sahabat-sahabat yang memberikan saya semangat,

Penulis-penulis jurnal, artikel, dan laman website yang tidak bisa saya sebutkan satu per satu atas sumber informasi yang sangat bermanfaat sebagai bahan referensi makalah ini.

TAUTAN KODE

https://github.com/pandaandsushi/ChatFilter

REFERENSI

[1] Delta. (2022, Nov 26). @username Regular Expression for social media with JavaScript. https://stackoverflow.com/questions/74579624/username-regular-expression-for-social-media-with-javascript (diakses pada 12 Juni 2024)

[2] Drizki. (2023, Juli 10). Indonesian Badwords. https://github.com/drizki/indonesian-badwords/blob/main/src/dict.json (diakses pada 12 Juni 2024)

[3] Makragic, A. (2017, Juni 21). Regex for Indonesian Number. https://stackoverflow.com/questions/44670612/regex-for-indonesian-phone-number (diakses pada 12 Juni 2024)

[4] Moore, A. (2013, Sep 29). Regular Expression to Match URLs in Java. https://stackoverflow.com/questions/163360/regular-expression-to-match-urls-in-java (diakses pada 12 Juni 2024)

[5] Munir, R. (2024) Homepage Rinaldi Munir. Sekolah Teknik Elektro dan Informatika (STEI) ITB. *String Matching dengan Regex.* Retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf (diakses pada 11 Juni 2024)

[6] Munir, R. (2023) Homepage Rinaldi Munir. Sekolah Teknik Elektro dan Informatika (STEI) ITB. *Pencocokan String.* Retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (diakses pada 11 Juni 2024)

[7] NTU. (2018, November). *Regular Expressions (Regex).* Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html (diakses pada 11 Juni 2024)

[8] Rock. D. (2014, Feb 4). RegEx for Validationg Emails. https://stackoverflow.com/questions/21608294/regex-for-validating-emails (diakses pada 12 Juni 2024)

[9] Seller Shopee Center (2024, April 30). Pelanggaran Chat. Retrieved from https://seller.shopee.co.id/edu/article/11462/Pelanggaran-Chat (diakses pada 10 Juni 2024)

[10] Wibisono, Y. et all. (2020, April 11). "Modul Praktikum: Pengantar Regular Expression". Retrieved from https://docs.google.com/document/d/1ls6h1A6m-Zhzw6e5eriwMNUAG0D1iwL-eVmVMS2XQoc/edit (diakses pada 10 Juni 2024)

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, 12 Juni 2024

Thea Josephine Halim
(13522012)