**LAPORAN TUGAS BESAR**

**IF2211 – STRATEGI ALGORITMA**

ALGORITMA PENCARIAN STRING

KELOMPOK

Anggota Kelompok :

Krisna Fathurahman -13511006

Mahessa Ramadhana -13511077

Muhamad Andri Eka F -13511088

**PRODI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BAB I**

**DESKRIPSI MASALAH**

* 1. LATAR BELAKANG

Algoritma pencocokan *string* (*pattern*) yang mempunyai kinerja bagus adalah Knuth-Morris-Pratt (KMP) dan Algoritma Boyer-Moore. Banyaknya kumpulan teks yang tersedia di internet menyebabkan berkembangnya sistem analisis teks seperti mesin pencari (*search engine*), *chatbot*, peringkas teks, analisis sentimen, dll. Salah satu fungsi dasar yang digunakan dalam sistem analisis teks tersebut adalah pencocokan string.

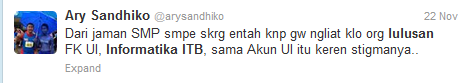
* 1. SPESIFIKASI TUGAS

Pada Tugas Besar III kali ini Anda diminta membuat aplikasi sederhana analisis sentimen berbasis kata kunci. Teks yang akan Anda proses adalah *tweets* berbahasa Indonesia di dalam jejaring sosial *Twitter*. Sebagai informasi, terdapat 30 juta akun Twitter dari Indonesia pada Juli 2012.

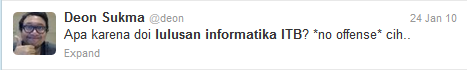
Pengguna aplikasi analisis sentimen sederhana ini akan memberikan 3 masukan yaitu: (1) *keyword* pencarian *tweet* yang akan dianalisis, misalnya “lulusan informatika itb”; (2) daftar *keyword* sentimen positif, misalnya “keren”, “hebat”, dll; (3) daftar *keyword* sentimen negatif, misalnya “sombong”, “aneh”, dll. Berdasarkan *keyword* pencarian *tweet*, aplikasi akan menggunakan *Twitter* *API* (*Application Programming Interface*) untuk mendapatkan kumpulan *tweets* sesuai *keyword* tersebut. Lalu, aplikasi menentukan sentimen dari setiap *tweet* dengan mengecek apakah *tweet* tersebut mengandung *keyword* positif. Jika tidak, cek apakah *tweet* mengandung *keyword* negatif. Jika tidak mengandung *keyword* positif dan negatif, *tweet* tersebut termasuk bersentimen netral.



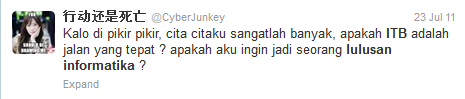
Gambar . Contoh tweets dengan pencarian “lulusan informatika itb”



Gambar . Contoh tweets dengan sentimen positif karena mengandung keyword “keren”



Gambar . Contoh tweets dengan sentimen negatif karena mengandung keyword “cih”



Gambar . Contoh tweets dengan sentimen netral

Pencocokan *string* yang anda buat adalah *exact matching*, jadi *tweet* yang diproses mengandung *string* yang tepat sama dengan *keyword* yang telah ditentukan oleh pengguna. Di sini Algoritma KMP dan Boyer-Moore dapat digunakan. Pencarian juga tidak bersifat *case sensitive*, jadi huruf besar dan huruf kecil dianggap sama (hal ini dapat dilakukan dengan mengganggap seluruh karakter di dalam *pattern* dan teks sebagai huruf kecil semua atau huruf kapital semua).

**Spesifikasi program :**

1. Aplikasi analisis sentimen yang anda buat merupakan aplikasi berbasis web yang menerima 3 masukan yaitu: (1) *keyword* pencarian *tweet* yang akan dianalisis, misalnya “lulusan informatika itb”; (2) daftar *keyword* sentimen positif, misalnya “keren”, “hebat”, dll; (3) daftar *keyword* sentimen negatif, misalnya “sombong”, “aneh”, dll. Tampilan antarmuka pengguna-komputer kira-kira seperti Gambar 5 di bawah ini:

|  |
| --- |
| **My Sentiment Analytics**  lulusan informatika itb  *Keyword* pencarian *tweets*:  Keren; hebat; jago  *Keyword* sentimen positif:  Sombong; aneh  *Keyword* sentimen negatif:  Analisis    Algoritma :   * Boyer-Moore * KMP   Perihal |

Gambar . Contoh antarmuka (laman web) aplikasi sederhana analisis sentimen

Perihal : link ke halaman tentang program dan pembuatnya

Analisis sentimen setiap *tweet* menggunakan hasil implementasi algoritma KMP dan Boyer-Moore dengan menggunakan Bahasa Java.

1. Luaran program (dengan mengklik tombol menu “Analisis”) adalah sebuah laman web baru yang berisi semua *tweet* yang sesuai dengan *keyword* yang diminta. *Tweets* tersebut dikelompokkan berdasarkan sentimen positif, sentimen negatif, dan sentimen netral. Pada setiap *tweet* yang dihasilkan, ada pranala ke laman *Twitter* yang berisi *tweet* tersebut.

Contoh:

Sebagai **lulusan** Teknik **Informatika** **ITB**, saya malu sistem registrasi **ITB** seperti ini. Apa bedanya dengan tahun 1990an?

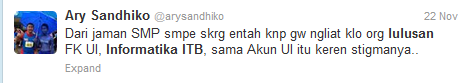
**Data Uji**

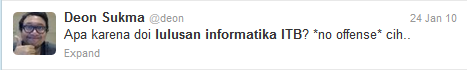
Data uji yang digunakan dapat anda tentukan sendiri, minimal terdapat 50 *tweets* yang dianalisis. *Tweet* dapat berbahasa Indonesia atau Inggris.

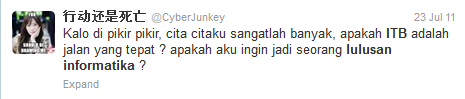
**Ihwal Pencocokan *String***

*String* berupa *keyword* yang dimasukkan oleh pengguna panjangnya dapat lebih dari satu kata (setiap kata dipisahkan oleh spasi). Salah satu trik pencarian untuk *pattern* berbentuk frase, pencocokan dilakukan kata per kata (ini memudahkan dalam dalam menghitung fungsi pinggiran pada KMP atau fungsi *last occurance* pada Boyer-Moore). Setelah kata pertama ditemukan, maka pencocokan dilakukan terhadap kata berikutnya. Jadi, meskipun di dalam *tweet* tidak terdapat frase yang tepat sama dengan *pattern*, namun hasil pencarian menampilkan semua *tweet* yang mengandung setiap kata di dalam frase. Anda bileh punya trik yang lain untuk menangani frase.

Misalnya jika *keyword* yang dimasukkan pengguna adalah “lulusan informatika ITB”, maka *tweets* yang berhasil diekstrak adalah seperti pada Gambar 1, 2, 3, dan 4 di atas (ditampilkan kembali di bawah ini):



****



**BAB II**

**DASAR TEORI**

* 1. JSP

Java Server Page adalah sebuaah metode untuk membuat halaman *web*  yang isinya secara generic dibuat yang mempunyai basis menggunakan HTML atau XML.JSP menggunakan bahasa pemrograman java sebagai syntax yang digunakan. Untuk menjalankan halaman web yang dibuat menggunakan JSP dibuthkan web server yang kompatibel seperti Apache Tomcat atau Jetty

Secara arsitektural, JSP bias dianggap sebagai abstraksi tingkat tinggi dari Java Servlet. JSP di terjemahkan menjadi servlet pada saat *runtime.*setiap servlet JSP akan di masukan ke dalama *cache* dan digunakan terusmenerus sampai data asli dari JSP berubah. Sehinga ketika halaman web pertama kali dibuka , akan diperlukan waktu yang lama untuk membukanya karena diperlukan waktu untuk memaskan data servlet ke cache, tetapai apabila di buka untuk kesekian kalinya, apabilan file jsp tidak beruabh , maka waktu yang dibutuhkan lebih sediki karena hanya perlu membaca dari cache.

* 1. PATTERN MATCHING
     1. Algoritma Knuth-Morris-Pratt

Pada algoritma *brute force* , setiap kali ditemukan ketidakcocokan *pattern* dengan teks, maka *pattern*  digeser satu karakter ke kanan, sedangkan pada algoritma KMP, dilakukan pemeliharaan informasi yang berguna untuk menghitung jumkah pergeseran yang dibutuhkan. Algoritma ini menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter seperti pada algoritma *brute force*

Algoritma KMP mengembangkan algoritma brute force menggunakan prefix dan suffix. Sebagai contoh, pada pattern P = 'abcabd', jika terjadi ketidakcocokan pada d (j = 5), kita cari suffix dari P[0..j-1] yang juga merupakan prefix dari P[0..j-1]. Dalam kasus ini, prefix 'ab' (P[0..1]) sama dengan suffix 'ab' (P[3..4]). Kita tahu bahwa prefix 'ab' sudah cocok, maka kita geser P sehinnga prefix 'ab' ada di posisi suffix 'ab' dengan menggeser P sejauh tiga karakter.

Contoh:

i 012345678

S: abcabcabd

P: abcabd

j 012345

Terjadi ketidakcocokan pd j = 5. Geser P sehingga prefix 'ab' ada di posisi suffix 'ab'

i 012345678

S: abcabcabd

P: abcabd

j 012345

Kita sudah tahu bahwa P[0..1] cocok dengan S[3..4], sehingga pencocokan dimulai dari j = 2. Pada akhirnya ditemukan bahwa P cocok dengan S di i = 3.

Untuk mempercepat pencocokan, dilakukan preprocessing pada pattern untuk membentuk tabel yang berisi berapa jauh pattern akan digeser jika terjadi ketidakcocokan pada j tertentu berdasarkan aturan prefix P[0..j-1] = suffix P[0..j-1]. Preprocessing ini dinamakan border function.

* + 1. Algoritma Boyer-Moore

Algoritma Boyer Moore mengembangkan brute force dengan cara mencari sejauh apa pattern bisa digeser jika terjadi ketidakcocokan berdasarkan posisi karakter pada pattern. Proses pencocokan string dengan pattern dilakukan dari ujung kanan pattern.

Contoh:

i 0123456789

S: abcceabcbd

P: abcbd

j 01234

Terjadi ketidakcocokan pada i = 3. Kita lihat bahwa S[3] = 'c'. Lalu kita cari di mana posisi 'c' terakhir di P, yang ada pada j = 2. Kita geser P sehingga 'c' terakhir di P ada pada posisi ketidakcocokan, yaitu dengan menggeser P sejauh satu karakter.

i 0123456789

S: abcceabcbd

P: abcbd

j 01234

Terjadi ketidakcocokan pada i = 5. Kita lihat bahwa S[5] = 'a', dan 'a' terkahir ada di j = 0. Kita geser P sejauh 4 karakter.

i 0123456789

S: abcceabcbd

P: abcbd

j 01234

Sehingga ditemukan kecocokan P dengan S.

Untuk mempercepat pencocokan, dilakukan preprocessing untuk mencari posisi terakhir tiap karakter yang ada pada pattern. Karakter yang tidak ada pada pattern bernilai -1. Preprocessing ini disebut last occurence function.

**BAB III**

**ANALISIS PEMECAHAN MASALAH**

* 1. **Algoritma Knuth-Morris-Pratt**

Dilakukan pattern matching berdasrkan sentiment yang berasal dari masukan user yaitu *pattern* yang akan dicari dan sentiment positif dan negative. Pattern matching terlebih dahulu dilakukan menggunakan twitter api untuk mendapatkan tweet yang berisi kata kunci, kemudiam dilakukana *pattern matching*  terhadap sentiment positif dan negative, untuk setiap sentiment positif yang ditemukan, tweet tersebut diberinilai +1 dan apabilan ditemukan kata bersentimen negative , maka nilai dari tweet itu akan dikurangi 1. Semua tweet pada aawal nya memiliki nilai 0 yang beraarti tweet ini memiliki sentiment netral.

Apabila setelah semua sentiment dilcari pada tweet, dihitung nilai akhir pada tweet tersebut, yaitu apabila nilai sentiment lebih besar dari 0 maka tweet dikategorikan sebagai tweet positif. Apabila nilai sentiment totoal pada tweet lebih kecil dari 0, maka tweet dikategorikan sebagi tweet dengan sentiment negative. Apabila tweet memiliiki nlai sentiment = 0 , maka twet dikategorikan sebagi tweet dengan sentiment negative

Contoh :

User memasukan kata kunci “”

**BAB IV**

**IMPLEMENTASI DAN PENGUJIAN**

* 1. PROGRAM

Source Code :

public class searchHandler **{**

public String nama**;**// nama dari orang yang menulis tweet

public String tweet**;** //isi dari tweet

public int jenistweet**=**0**;** //ini bisa dipake buat nandain dia masuk yang mana, kalo 0 = netral , 1= positif , 2 = negatif

public String URL**;** //ini nyimpen url dari tweet nya

**}**

public class Twitter\_testing **{**

// This is where you enter your Oauth info

static String OAuthConsumerKey **=** "SlzgGdUxCJNdDOtnNyFaCA"**;**

static String OAuthConsumerSecret **=** "8Gq1uQRL410JYs5BksiD1Q5RpfRo1HhYueOu6jo"**;**

// This is where you enter your Access Token info

static String AccessToken **=** "2217976202-RCZisGUldqZwwQ2wJacBSEks3nBl2vOmf19X3Hj"**;**

static String AccessTokenSecret **=** "M7w451umoEnYxdxNgE7zuiJyFivxjOq4Is669kSWk7NsP"**;**

// Just some random variables kicking around

String myTimeline**;**

java**.**util**.**List statuses **=** **null;**

User**[]** friends**;**

TwitterFactory twitterFactory**;**

Twitter twitter**;**

RequestToken requestToken**;**

String**[]** theSearchTweets **=** **new** String**[**11**];**

//variable buat nyimpen hasil search

public Vector**<**searchHandler**>** shholder **=** **new** Vector**<**searchHandler**>();**

public void setup**()** **{**

**}**

// Initial connection

private void connectTwitter**()** **{**

**}**

// Sending a tweet

public void sendTweet**(**String t**)** **{}**

// Loading up the access token

private static twitter4j**.**auth**.**AccessToken loadAccessToken**(){}**

// Get your tweets

public void getTimeline**()** **{}**

// Search for tweets

public void getSearchTweets**(**String regex**)** **{}**

**}**

public class TweetParser **{**

private int sentiment**;** //Overall sentimen

private List**<**String**>** keywords**;** //List keyword untuk pencarian tweet

private List**<**String**>** positiveWord**;** //List kata2 positif

private List**<**String**>** negativeWord**;** //List kata2 negatif

private List**<**List**<**Integer**>** **>**kmpTable1**;** //Tabel border function KMP untuk keywords

private List**<**List**<**Integer**>** **>**kmpTable2**;** //Tabel border function KMP untuk positiveWord

private List**<**List**<**Integer**>** **>**kmpTable3**;** //Tabel border function KMP untuk negativeWord

private List**<**Map**<**Character**,** Integer**>** **>**bmTable1**;** //Tabel border function KMP untuk keywords

private List**<**Map**<**Character**,** Integer**>** **>**bmTable2**;** //Tabel border function KMP untuk positiveWord

private List**<**Map**<**Character**,** Integer**>** **>**bmTable3**;** //Tabel border function KMP untuk negativeWord

boolean mode**;** //true = KMP, false = BM

/\*

Constructor.

\*/

public TweetParser**(){}**

/\*

Reset object. Fungsi digunakan untuk set keywords dan mode pencarian.

\*/

public void Reset**(**List**<**String**>** key**,** List**<**String**>** pos**,** List**<**String**>** neg**,** boolean mod**){}**

private void BuildKMPTables**(){}**

/\*Search keyword pencarian. Akan mengembalikan string dimana kata yang dicari akan dibungkus dengan HTML tag bold.

Jika tidak ada match, akan mengembalikan string dengan panjang = 0.

\*/

public String SearchKey**(**String tweet**){}**

/\*Pencarian keyword2 positif dan negatif.

Untuk tiap kata positif, akan menambah nilai sentimen dr tweet tsb.

Untuk tiap kata positif, akan mengurangi nilai sentimen dr tweet tsb.

Jika sentimen tweet > 0, overall sentimen bertambah 1.

Jika sentimen tweet < 0, overall sentimen berkurang 1.

\*/

public int SearchSentiment**(**String tweet**){}**

/\*

Pencocokan menggunakan KMP

\*/

private int MatchKMP**(**String s**,** String p**,** List**<**Integer**>** table**){}**

private int LastBM**(**char c**,** Map**<**Character**,** Integer**>** table**){}**

private int MatchBM**(**String s**,** String p**,** Map**<**Character**,** Integer**>** table**){}**

//Overall sentimen

public int Result**(){}**

//Menghapus seluruh isi object.

public void Clear**(){}**

* 1. HASIL PENGUJIAN

**BAB V**

**KESIMPULAN DAN SARAN**

DAFTAR PUSTAKA

* Cormen, Thomas, dkk. *Introduction to Algorithm Third Edition*. The MIT Press. 2009. Cambridge, London.