LAPORAN TUGAS BESAR III

IF2211 Strategi Algoritma

String Matching

­NAMA KELOMPOK

FOTO

Dipersiapkan oleh:

1. Fawwaz Muhammad / 13511XXX
2. Renusa Andra Prayogo / 13511063

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Sekolah Teknik Elektro dan Informatika ITB** | - | Halaman |
| *IF2211-TBIII* |  |
| *2 Desember 2013* |

Daftar Isi

[1 Bab I. Deskripsi Masalah 3](#_Toc373663433)

[2 Bab II. Dasar Teori 6](#_Toc373663434)

[2.1 Algoritma Knuth-Morris-Pratt (KMP) 6](#_Toc373663435)

[2.2 Algoritma Boyer Moore (BM) 6](#_Toc373663436)

[2.2.1 Looking-Glass Technique 6](#_Toc373663437)

[2.2.2 Character-jump technique 6](#_Toc373663438)

[2.2.3 Last Occurrence Function 8](#_Toc373663439)

[3 Bab III. Analisis Pemecahan Masalah 8](#_Toc373663440)

[4 Bab IV. Implementasi dan Pengujian 8](#_Toc373663441)

[4.1 Struktur Data Program 8](#_Toc373663442)

[4.1.1 Struktur Data Algoritma Pattern Matching 10](#_Toc373663443)

[4.2 Pengujian Program 12](#_Toc373663444)

[4.3 Analisis Pengujian 13](#_Toc373663445)

[5 Bab V. Kesimpulan dan Saran 13](#_Toc373663446)

[5.1 Kesimpulan 13](#_Toc373663447)

[5.2 Saran 13](#_Toc373663448)

[Daftar Referensi 13](#_Toc373663449)

# Bab I. Deskripsi Masalah

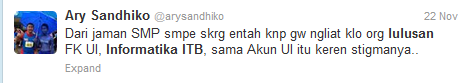
**Deskripsi tugas :**

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. 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.

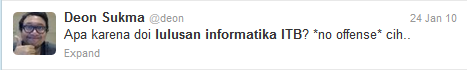
Pengguna aplikasi analisis sentimen sederhana ini akan memberikan 3 masukan yaitu: (1) *keyword* pencarian *tweet* yang akan dianalisis, misalnya “lulusan informatika itb”; (2) daftar *keyword* sentimen positif, misalnya “keren”, “hebat”, dll; (3) daftar *keyword* sentimen negatif, misalnya “sombong”, “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.



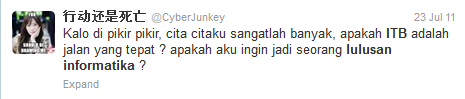
Gambar 1. Contoh tweets dengan pencarian “lulusan informatika itb”



Gambar 2. Contoh tweets dengan sentimen positif karena mengandung keyword “keren”



Gambar 3. Contoh tweets dengan sentimen negatif karena mengandung keyword “cih”



Gambar 4. 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:

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

Gambar 4. 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?

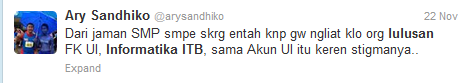
**Data Uji**

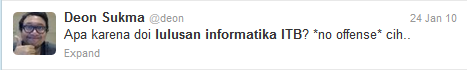
Data uji yang digunakan dapat anda tentukan sendiri, minimal terdapat 50 *tweets* yang dianalisis. *Tweet* dapat berbahasa Indonesia atau Inggris.

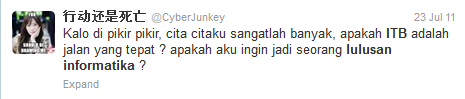
**Ihwal Pencocokan *String***

*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):



****



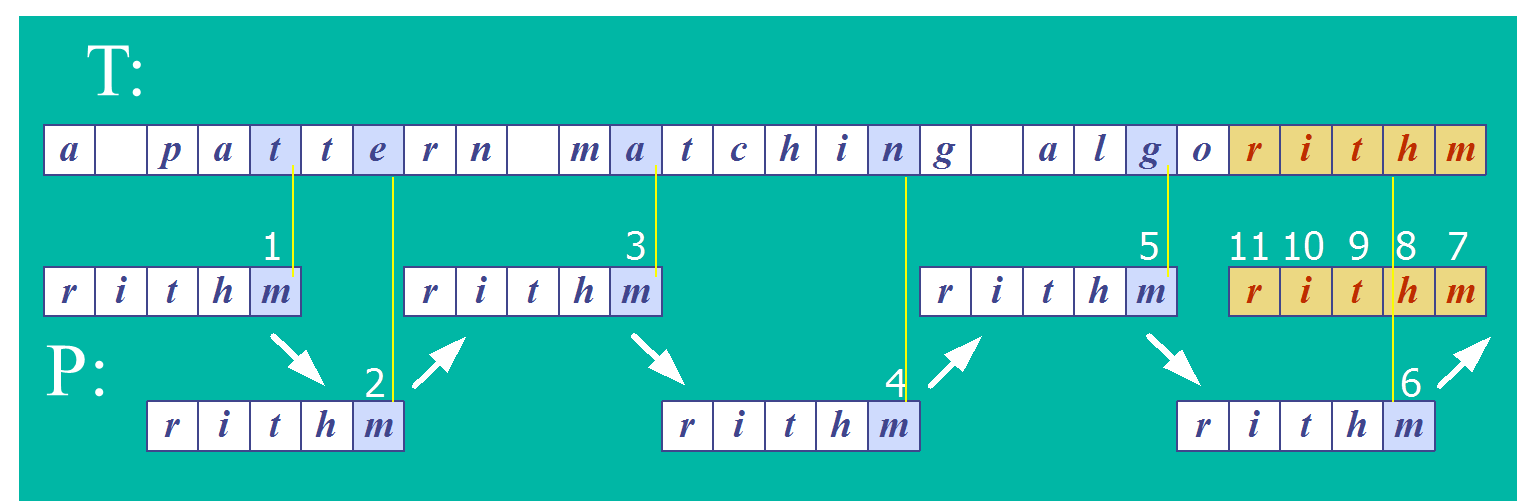
# Bab II. Dasar Teori

## Algoritma Knuth-Morris-Pratt (KMP)

## Algoritma Boyer Moore (BM)

Algoritma Boyer Moore adalah sebuah teknik *Pattern Matching* yang berdasarkan oleh dua teknik, yaitu *looking-glass technique* dan *character-jump technique*.

### Looking-Glass Technique

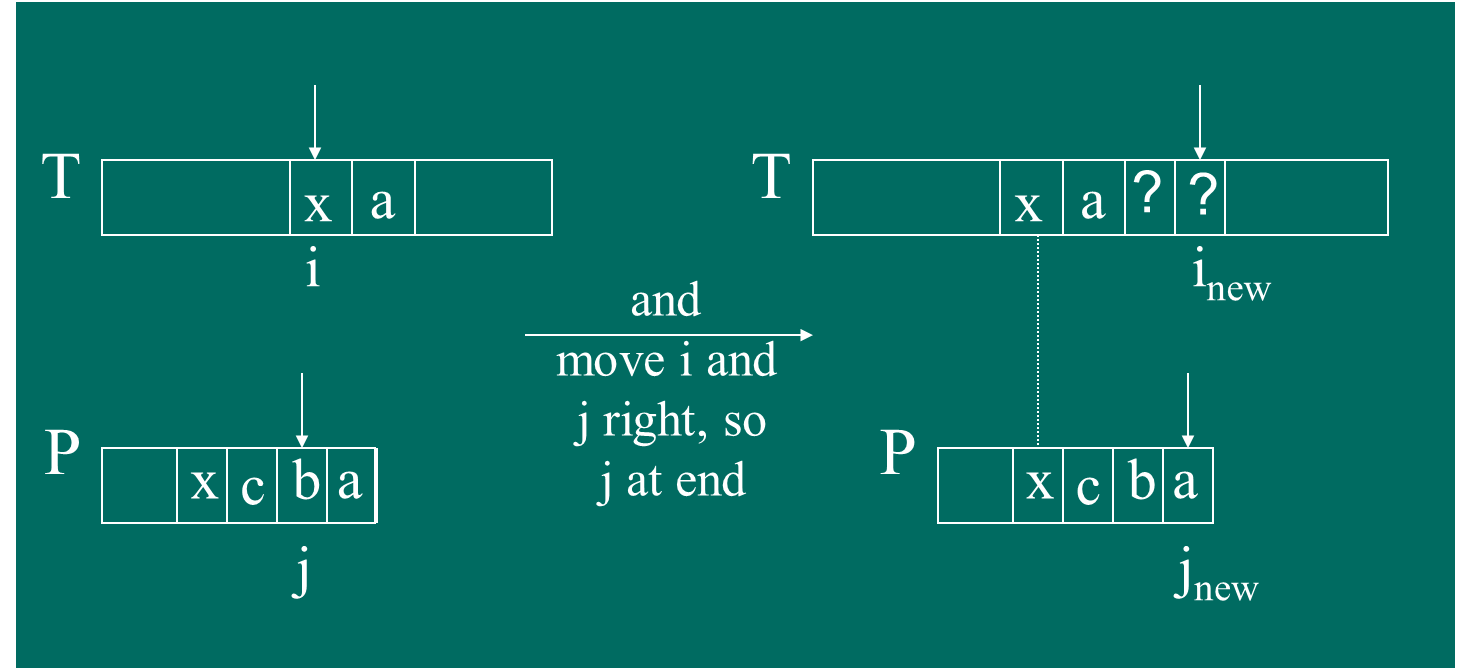


Pada teknik ini, pencarian string P pada string T bergerak secara mundur, dimulai dari ujung akhir dari string P.

### Character-jump technique

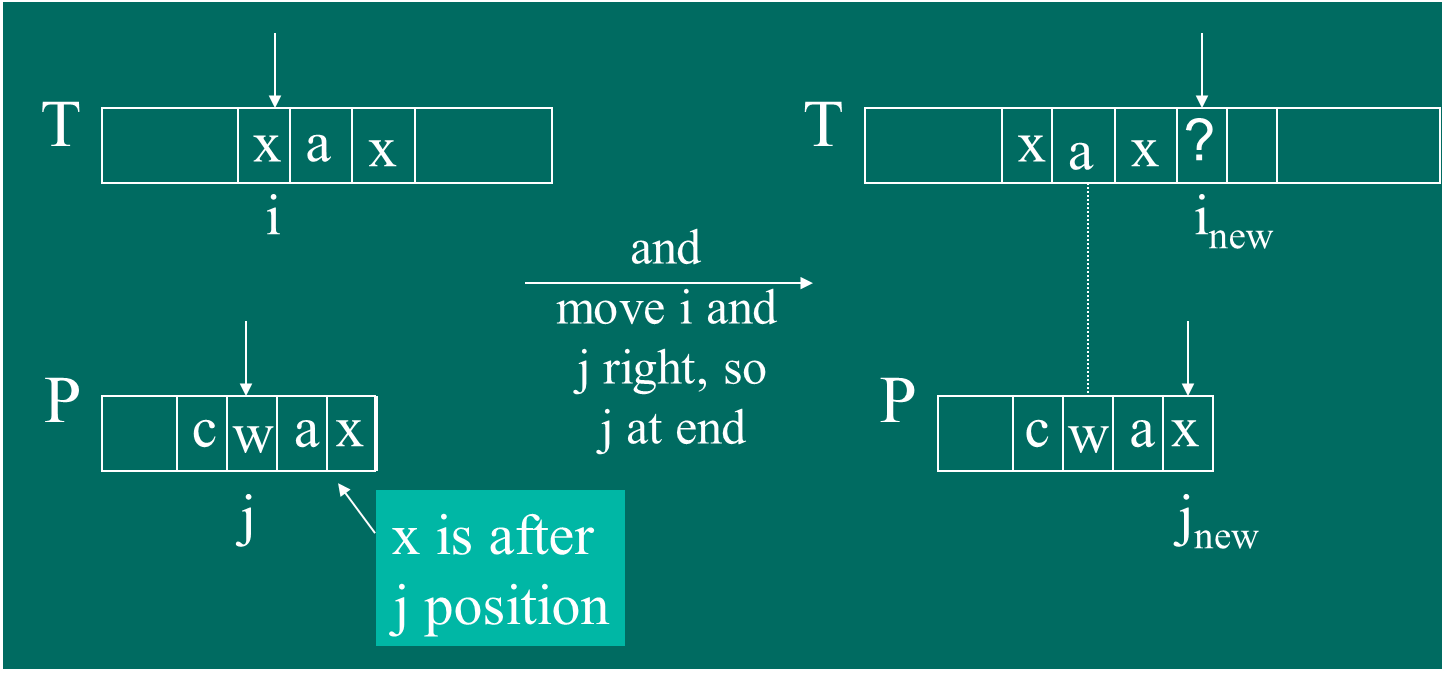
Pada teknik ini, apabila terjadi suatu kondisi dimana ditemukan T[i] = x dan T[i] ≠ P[i], akan diproses menjadi tiga kasus yang berbeda :

* Case 1



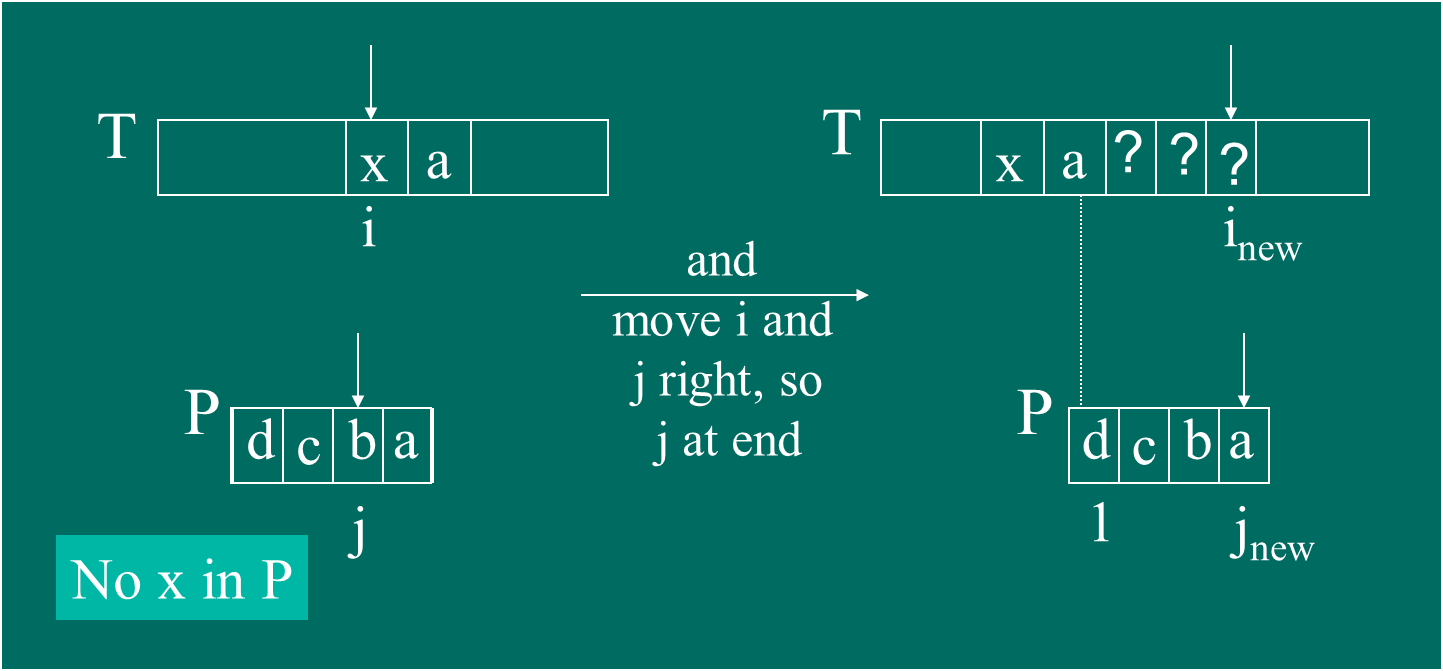
Case 1 terjadi apabila pada string P terdapat x pada suatu tempat P[j] dimana j < i. Saat kasus ini terjadi, geser string P hingga P[j] tersebut dipasangkan dengan T[i] = x.

* Case 2



Case 2 terjadi apabila string P memiliki x, namun letaknya di P[k] dengan k > i. Maka saat kasus ini terjadi, geser string P satu karakter ke kanan.

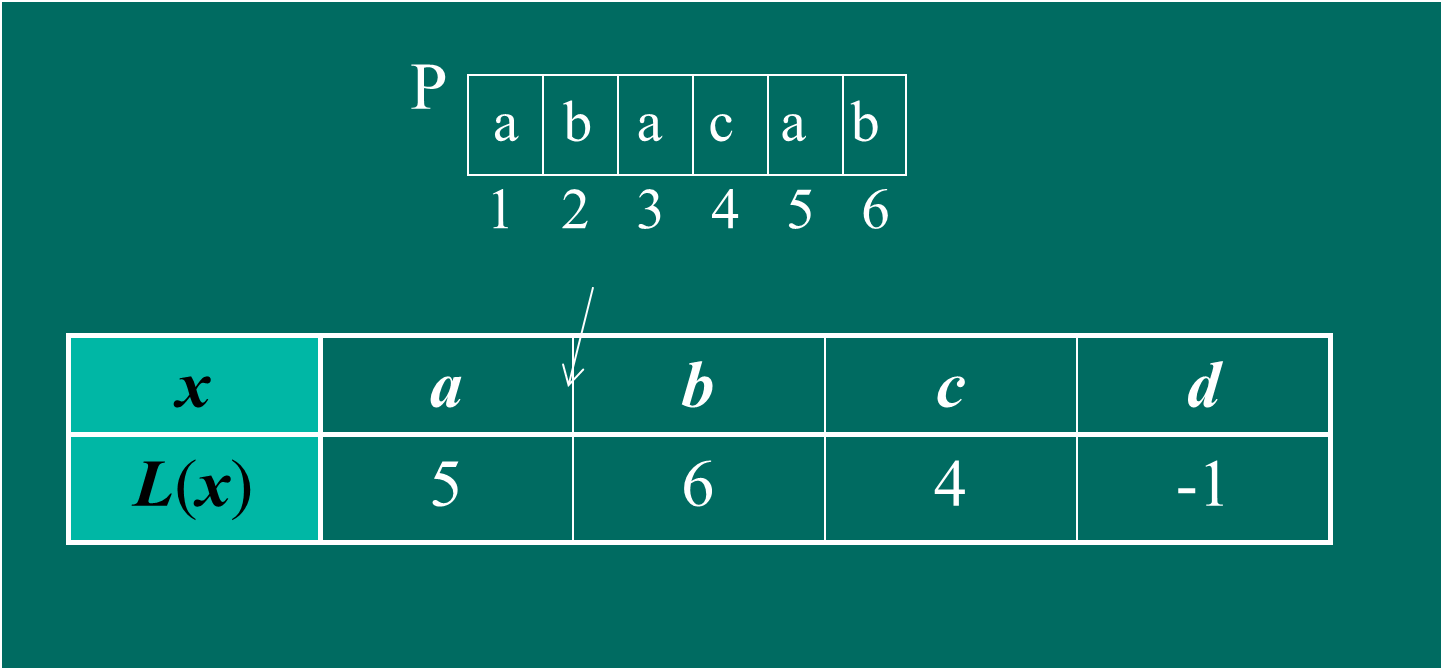
* Case 3



Jika Case 1 dan Case 2 tidak memenuhi kondisi tersebut, maka geser string P tersebut, hingga string P[1] dipasangkan dengan string T[i+1].

### Last Occurrence Function

Untuk dapat menggunakan algoritma Boyer Moore ini, program harus dapat mengetahui posisi dari setiap karakter unik yang ada pada string P. Oleh karena itu, sebelum membandingkan kedua string P dan T, program harus mencatat terlebih dahulu posisi-posisi string tersebut.



# Bab III. Analisis Pemecahan Masalah

Langkah-langkah pemecahan masalah ada di sini beserta contoh ilustrasi.

# Bab IV. Implementasi dan Pengujian

## Struktur Data Program

Pada implementasi dari program tugas besar ini, dibuat tiga struktur utama, yaitu struktur data untuk algoritma pattern matching (KMP dan Boyer Moore), struktur data yang mengolah data dari halaman web untuk ditampilkan kembali berupa tampilan tweet yang memiliki sentiment positif/negatif, dan yang terakhir adalah halaman web itu sendiri.

Index.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>  <title>Insert title here</title>  </head>  <body>  <h1>Welcome to Sentiment Tweet finder!</h1>  <form name=*"Input"* action=*"ProcessInput"* method=*"get"*>  Keyword : <input type=*"text"* name=*"keyword"*/><br>  Sentimen positif (pisahkan dengan spasi): <input type=*"text"* name=*"positif"*/><br>  Sentimen negatif (pisahkan dengan spasi): <input type=*"text"* name=*"negatif"*/><br>  Search By : <br>  <input type=*"radio"* name=*"metode"* value=*"KMP"*>KMP<br>  <input type=*"radio"* name=*"metode"* value=*"BM"*>Boyer Moore<br>  <input type=*"submit"* value=*"Search"*/>  </form>  </body>  </html> |

Result.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <%@page import=*"twitter4j.Status"*%>  <%@page import=*"java.util.ArrayList"*%>  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>  <title>Insert title here</title>  </head>  <body>  <h1>Result tweet :</h1>  <h2>With <% out.println(request.getAttribute("mode")); %> Algorithms</h2>  <%  ArrayList<Status> positive = (ArrayList<Status>)request.getAttribute("positiveResult");  ArrayList<String> positiveK = (ArrayList<String>)request.getAttribute("positiveKeyword");  out.println("<h2>Positive Sentiment Tweets : </h2>");  **int** i = 0;  **for** (Status tweet : positive) {  out.println("<p style=\"color:blue\"> @" + tweet.getUser().getScreenName() + "</p>" + tweet.getText() + " <br><p style=\"color:red\"> Keyword : "+ positiveK.get(i) + "</p><br>");  i++;  }    ArrayList<Status> negative = (ArrayList<Status>)request.getAttribute("negativeResult");  ArrayList<String> negativeK = (ArrayList<String>)request.getAttribute("negativeKeyword");  out.println("<br><h2>Negative Sentiment Tweets : </h2>");  **int** j = 0;  **for** (Status tweet : negative) {  out.println("<p style=\"color:blue\"> @" + tweet.getUser().getScreenName() + "</p>" + tweet.getText() + " <br><p style=\"color:red\"> Keyword : "+ negativeK.get(i) + "</p><br>");  }  %>  </body>  </html> |

### Struktur Data Algoritma Pattern Matching

Pada program ini, baik Boyer Moore dan KMP keduanya dijadikan sebuah kelas tersendiri dengan nama BoyerMoore.java dan KMP.java. Keduanya dimasukkan dalam package patternMatching.

**BoyerMoore.java**

Pada kelas ini, terdapat fungsi utama yaitu run(String pattern). Fungsi dari method ini adalah untuk menentukan apakah string pattern merupakan bagian dari string text. Apabila ya, maka method tersebut akan mengembalikan nilai true.

|  |
| --- |
| **package** patternMatching;  **public** **class** BoyerMoore {  **private** String string;  **public** BoyerMoore(){    }    **public** BoyerMoore(String string) {  **this**.string = string;  }  **public** **boolean** run(String pattern) {  **int** last[] = buildLast(pattern);  **int** n = string.length();  **int** m = pattern.length();  **int** i = m - 1;  **if** (i > n - 1)  **return** **false**; // no match if pattern is \_ // longer than text  **int** j = m - 1;  **do** {  **if** (pattern.charAt(j) == string.charAt(i)) {  **if** (j == 0) {  **return** **true**; // match  } **else** { // looking-glass technique\_  i--;  j--;  }  } **else** { // character jump technique\_  **int** lo = last[string.charAt(i)]; // last occ  i = i + m - Math.*min*(j, 1 + lo);  j = m - 1;  }  } **while** (i <= n - 1);  **return** **false**; // no match\_  }  **private** **int**[] buildLast(String pattern) {  **int** last[] = **new** **int**[256];  **for** (**int** i = 0; i < 128; i++)  last[i] = -1; // initialize array  **for** (**int** i = 0; i < pattern.length(); i++)  last[pattern.charAt(i)] = i;  **return** last;  }  **public** String getString() {  **return** string;  }  **public** **void** setString(String string) {  **this**.string = string;  }  } |

**KMP.java**

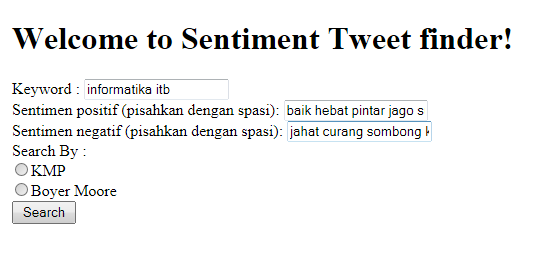
Pada kelas ini, method yang berfungsi sebagai pattern matching adalah KMPMatch(String text, String pattern). Apabila pattern tersebut ada pada text, maka method ini akan mengembalikan nilai berupa index lokasi dari pattern tersebut pada text. Jika tidak ditemukan, maka nilai kembaliannya adalah -1.

|  |
| --- |
| **package** patternMatching;  **public** **class** KMP {  **public** **void** KMP() {  }  **public** **static** **int** KMPMatch(String text, String pattern) {  **int** n = text.length();  **int** m = pattern.length();  **int** border[] = *ComputeBorder*(pattern);  **int** i = 0, j = 0;  **while** (i < n) {  **if** (pattern.charAt(j) == pattern.charAt(i)) {  **if** (j == m - 1)  **return** i - m + 1;  i++;  j++;  } **else** **if** (j > 0) {  j = border[j - 1];  } **else** {  i++;  }  }  **return** -1; // gak ketemu  }  **public** **static** **int**[] ComputeBorder(String pattern) {  **int** border[] = **new** **int**[pattern.length()];  border[0] = 0;  **int** m = pattern.length(), j = 0, i = 1;  **while** (i < m) {  **if** (pattern.charAt(j) == pattern.charAt(i)) {  border[i] = j + 1;  i++;  j++;  } **else** **if** (j > 0) {  j = border[j - 1];  } **else** {  border[i] = 0;  i++;  }  }  **return** border;  }  } |

## Pengujian Program

Pencarian tweet “informatika ITB” dengan sentiment :

* Positif : baik, hebat, pintar, jago, sukses, selamat, bagus
* Negatif : jahat, curang, sombong, kurang, lambat, pikun



Gambar 1 Tampilan awal program



Gambar 2 Hasil pencarian dengan algoritma BM

## Analisis Pengujian

# Bab V. Kesimpulan dan Saran

## Kesimpulan

## Saran

# Daftar Referensi

[1] Munir, Rinaldi. *Algoritma Pattern Matching*. Diakses 29 November 2013.

< <http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014/stmik13-14.htm> >