

Laporan Tugas Besar III

IF2211 Strategi Algoritma

Aplikasi String Matching untuk Disposisi Tweets ke Dinas-Dinas dan Instansi di Bawah Pemerintah Kota Bandung



Micky Yudi Utama 13514011

Atika Azzahra Akbar 13514077

Ade Yusuf Rahardian 13514079

Program Studi Teknik Informatika
Institut Teknologi Bandung
Semester Genap Tahun Ajaran 2015/2016

Daftar Isi

Bab I Deskripsi Tugas	3
Bab II Dasar Teori	6
A. Algoritma Knuth-Morris-Pratt	6
B. Algoritma Boyer-Moore	8
Bab III Analisis Pemecahan Masalah.....	11
A. Spesifikasi Program	11
B. Langkah-langkah Pemecahan Masalah	11
Bab IV Implementasi dan Pengujian.....	12
A. Struktur Data.....	12
B. Fungsi dan Prosedur	12
C. Eksperimen / Pengujian.....	13
D. Analisis Hasil Pengujian	16
Bab V Kesimpulan dan Saran	17
A. Kesimpulan.....	17
B. Saran.....	17
Daftar Pustaka	18

Bab I

Deskripsi Tugas

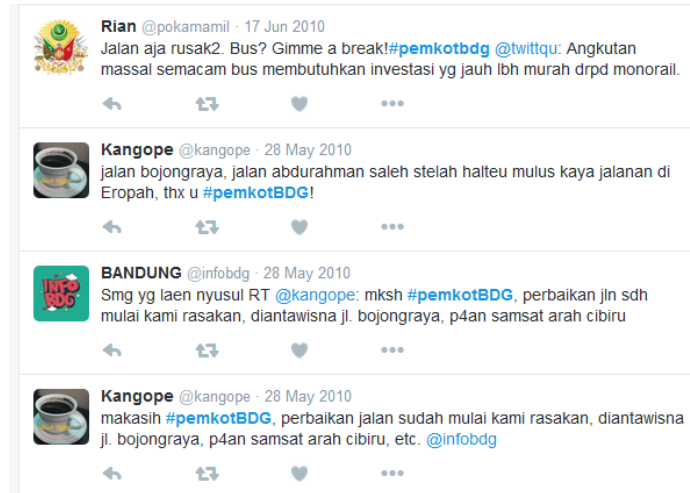
Pada Tugas Besar III kali ini Kami diminta membuat aplikasi sederhana analisis *tweet* berbasis kata kunci. *Tweet* yang berhasil diunduh dari *Twitter* dianalisis untuk selanjutnya didisposisikan ke dinas atau instansi yang terkait. Salah satu fungsi dasar yang digunakan dalam sistem analisis teks tersebut adalah pencocokan string. Algoritma pencocokan *string (pattern)* yang digunakan adalah *Knuth-Morris-Pratt (KMP)*, dan *Boyer-Moore (BM)*.

Pengguna aplikasi analisis *tweet* sederhana ini akan memberikan dua jenis masukan yaitu: (1) *keyword* pencarian *tweet* yang berasal dari tagar (*hashtag*) atau *mention* #pemkotbdg, @pemkotbdg, #pemkotbandung, @ridwan kamil, @infobdg, dan yang sejenis dengan itu, dan (2) *keyword*. Tabel berikut menunjukkan contoh masukan dari pengguna.

Keyword Pencarian Tweet	Keyword	Output ke Dinas/Lembaga
#pemkotbdg @pemkotbdg @ridwan kamil @infobdg	Pipa bocor; air ga ngalir; pdam mati; ga ngocor ...	PDAM Bandung
	PJU; Penerangan mati; lampu padam; terang; jalan gelap; ...	Dinas Bina Marga
	sampah; bau busuk; ...	Dinas kebersihan



Gambar 1. Contoh tweets @pemkotbdg



Gambar 2. Contoh tweets #pemkotbdg

Berdasarkan *keyword* pencarian *tweet* yang dimasukkan pengguna, aplikasi akan menggunakan *Twitter API* untuk mendapatkan minimal 100 *tweets* terbaru (*recent tweets*). Lalu, aplikasi menentukan kategori dinas/instansi dari setiap *tweet* dengan mengecek apakah *tweet* tersebut mengandung *keyword* yang diberikan untuk kategori dinas/lembaga tersebut.

- Jika *tweet* mengandung lebih dari satu *keyword* dari kategori dinas yang berbeda, *tweet* tersebut masuk dalam kategori dengan *keyword* yang muncul lebih dulu. Sebagai contoh, jika terdapat *keyword* “sampah” (kategori “dinas kebersihan”) dan “taman kotor” (kategori “dinas pertamanan”), *tweet* masuk kategori dinas kebersihan jika “sampah” muncul lebih dulu daripada kata “taman”.
- Jika tidak mengandung *keyword* apapun, *tweet* tersebut masuk dalam kategori dinas “unknown”.

Pencocokan *string* yang dibuat 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. Untuk *keyword* berupa frase, maka perlu dilakukan penanganan secara khusus untuk menemukan *tweet* yang mengandung kata-kata di dalam frasa. Pencarian juga tidak bersifat *case sensitive*, jadi huruf besar dan huruf kecil dianggap sama (hal ini dapat dilakukan dengan menggampang seluruh karakter di dalam *pattern* dan teks sebagai huruf kecil semua atau huruf kapital semua).

Jika *tweet* mengandung nama tempat (dimulai dengan kata depan “di”), maka sebagai **bonus**, Kami dapat menampilkan lokasi dari setiap kategori dengan *GoogleMaps API*.

Tweet	Lokasi Hasil Ekstraksi
#pemkotbdg 09.16 : lampu di jln tamansari jika malam padam	tamansari
@infobdg: 08.23 : Macet (lagi) di jln rancaekek - cicalengka maju dikit2 ga terlalu pic.twitter.com/gyK7PaCTVM	rancaekek - cicalengka

@Latifaa Aulia:	
#suaraBDG via @dionmudjenan : Hatihati di jembatan perbatasan komp GBI td ada yg kena begal pelaku mabok bawa golok nyandra yg bawa motor	jembatan perbatasan komp GBI
@ridwan kamil : di terminal leuwipanjang banyak anak jalanan yg ngemis2 sementara ibunya diem di pos polisi sambil bbman X_X #Suarabdg @dinsos bdg	terminal leuwipanjang

Bab II

Dasar Teori

A. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencarian string, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977.

Perhitungan penggeseran pada algoritma ini adalah sebagai berikut, bila terjadi ketidakcocokkan pada saat pattern sejajar dengan **teks[i..i+n-1]**, kita bisa menganggap ketidakcocokan pertama terjadi di antara **teks[i+j]** dan **pattern[j]**, dengan $0 < j < n$. Berarti, **teks[i..i+j-1] = pattern[0..j-1]** dan **a=teks[i+j]** tidak sama dengan **b=pattern[j]**. Ketika kita menggeser, sangat beralasan bila ada sebuah awalan **v** dari pattern akan sama dengan sebagian akhiran **u** dari sebagian teks. Sehingga kita bisa menggeser pattern agar awalan **v** tersebut sejajar dengan akhiran dari **u**.

Dengan kata lain, pencocokkan string akan berjalan secara efisien bila kita mempunyai tabel yang menentukan berapa panjang kita seharusnya menggeser seandainya terdeteksi ketidakcocokkan di karakter ke-j dari pattern. Tabel itu harus memuat **next[j]** yang merupakan posisi karakter **pattern[j]** setelah digeser, sehingga kita bisa menggeser pattern sebesar **j-next[j]** relatif terhadap teks.

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

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:
 - a. Karakter di pattern dan di teks yang dibandingkan tidak cocok (*mismatch*).
 - b. 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.

Berikut adalah pseudocode algoritma KMP pada fase pra-pencarian:

```
procedure preKMP (  
    input P : array[0..n-1] of char,  
    input n : integer,  
    input/output kmpNext : array[0..n] of integer  
)
```

```

Deklarasi:
i,j: integer

Algoritma
  i := 0;
  j := kmpNext[0] := -1;
  while (i < n) {
    while (j > -1 and not(P[i] = P[j]))
      j := kmpNext[j];
    i:= i+1;
    j:= j+1;
    if (P[i] = P[j])
      kmpNext[i] := kmpNext[j];
    else
      kmpNext[i] := j;
    endif
  endwhile

```

Dan berikut adalah pseudocode algoritma KMP pada fase pencarian:

```

procedure KMPSearch(
  input m, n : integer,
  input P : array[0..n-1] of char,
  input T : array[0..m-1] of char,
  output ketemu : array[0..m-1] of boolean
)

Deklarasi:
i, j,next: integer
kmpNext : array[0..n] of integer

Algoritma:
preKMP(n, P, kmpNext)
i:=0
while (i<= m-n) do
  j:=0
  while (j < n and T[i+j] = P[j]) do
    j:=j+1
  endwhile

```

```

    if(j >= n) then
        ketemu[i]:=true;
    endif
    next:= j - kmpNext[j]
    i:= i+next
endwhile

```

Algoritma ini menemukan semua kemunculan dari pattern dengan panjang n di dalam teks dengan panjang m dengan kompleksitas waktu $O(m+n)$. Algoritma ini hanya membutuhkan $O(n)$ ruang dari memory internal jika teks dibaca dari file eksternal. Semua besaran O tersebut tidak tergantung pada besarnya ruang alpabet.

B. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah salah satu algoritma pencarian string, dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada tahun 1977.

Algoritma ini dianggap sebagai algoritma yang paling efisien pada aplikasi umum. Algoritma Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat.

Misalnya ada sebuah usaha pencocokan yang terjadi pada **teks[i..i+n-1]**, dan anggap ketidakcocokan pertama terjadi di antara **teks[i+j]** dan **pattern[j]**, dengan $0 < j < n$. Berarti, **teks[i+j+1..i+n-1] = pattern[j+1..n-1]** dan **a=teks[i+j]** tidak sama dengan **b=pattern[j]**. Penggeseran yang mungkin adalah:

Penggeseran bad-character yang terdiri dari menyejajarkan **teks[i+j]** dengan kemunculan paling kanan karakter tersebut di pattern. Bila karakter tersebut tidak ada di pattern, maka pattern akan disejajarkan dengan **teks[i+n+1]**.

Secara sistematis, 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 teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
 - a. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
 - b. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.

3. Algoritma kemudian menggeser pattern nilai penggeseran bad-character, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Berikut adalah pseudocode algoritma Boyer-Moore pada fase pra-pencarian:

```
procedure preBmBc(  
    input P : array[0..n-1] of char,  
    input n : integer,  
    input/output bmBc : array[0..n-1] of integer  
)  
Deklarasi:  
    i: integer  
  
Algoritma:  
    for (i := 0 to ASIZE-1)  
        bmBc[i] := m;  
    endfor  
    for (i := 0 to m - 2)  
        bmBc[P[i]] := m - i - 1;  
    endfor
```

Dan berikut adalah pseudocode algoritma Boyer-Moore pada fase pencarian:

```
procedure BoyerMooreSearch(  
    input m, n : integer,  
    input P : array[0..n-1] of char,  
    input T : array[0..m-1] of char,  
    output ketemu : array[0..m-1] of boolean  
)  
  
Deklarasi:  
i, j, shift, bmBcShift: integer  
BmBc : array[0..255] of interger  
  
Algoritma:  
preBmBc(n, P, BmBc)  
i:=0  
while (i<= m-n) do  
    j:=n-1  
    while (j >=0 n and T[i+j] = P[j]) do
```

```

        j:=j-1
    endwhile
    if(j < 0) then
        ketemu[i]:=true;
    endif
    bmBcShift:= BmBc[charToInt(T[i+j])]-n+j+1
    shift:= bmBcShift
    i:= i+shift

```

Tabel untuk penggeseran bad-character dapat dihitung dengan kompleksitas waktu dan ruang sebesar $O(n + \sigma)$ dengan σ adalah besar ruang alfabet. Sedangkan pada fase pencarian, algoritma ini membutuhkan waktu sebesar $O(mn)$, pada kasus terburuk, algoritma ini akan melakukan $3n$ pencocokkan karakter, namun pada performa terbaiknya algoritma ini hanya akan melakukan $O(m/n)$ pencocokkan.

Bab III

Analisis Pemecahan Masalah

A. Spesifikasi Program

1. Aplikasi berbasis web analisis *tweets* yang anda buat menerima 2 masukan yaitu: (1) *keyword* pencarian *tweet* yang akan dianalisis berupa *hashtag* atau *mention*; (2) daftar *keyword*.
2. Luaran program adalah sebuah laman web baru yang berisi hasil analisis berupa daftar *tweets* untuk setiap dinas terkait, minimal seperti yang ditunjukkan oleh gambar berikut. Pada setiap *tweet* yang dihasilkan, ada pranala ke laman *Twitter* yang berisi *tweet* tersebut.

B. Langkah-langkah Pemecahan Masalah

Pada tugas besar 3 ini, kami menggunakan API TweetSharp untuk memperoleh 100 tweet terbaru berdasarkan kata kunci yang diminta oleh pengguna. Kata kunci yang diminta bisa lebih dari 1. Contoh: @PemkotBandung;#PemkotBdg.

Selanjutnya, dilakukan iterasi pada setiap tweet untuk mengetahui hubungan tweet tersebut dengan dinas tertentu. Pada setiap iterasi dilakukan pencocokan string antara teks pada tweet dengan kata kunci pada dinas tertentu. Dinas yang memiliki hasil pencocokan string dengan indeks terkecil merupakan dinas yang memiliki hubungan dengan tweet tersebut. Jika setelah iterasi selesai dilakukan, tidak terdapat dinas dengan kata kunci yang cocok dengan teks pada tweet, maka tweet tersebut masuk dalam kategori dinas lainnya. Algoritma pencocokan string yang dapat digunakan adalah algoritma Knutt-Morris-Pratt dan algoritma Boyer-Moore.

Selain itu pada setiap tweet juga dilakukan pencocokan string dengan kata “ di “ untuk mengetahui apakah terdapat lokasi yang dicantumkan pada tweet tersebut. Jika terdapat lokasi yang dicantumkan, maka dengan menggunakan API Google Map Jmelosegui dicantumkan pula lokasi tersebut pada map google.

Bab IV

Implementasi dan Pengujian

A. Struktur Data

- a. `List<int>` digunakan untuk membangun tabel berupa list yang dipakai dalam pencocokkan string dengan algoritma KMP.
- b. `Dictionary<int,int>` digunakan untuk mencatat posisi indeks terakhir untuk setiap kemunculan karakter.
- c. `IEnumerable<TwitterStatus>` digunakan untuk menyimpan seratus *tweets* terakhir.
- d. `List<TwitterStatus>` digunakan untuk menyimpan status-status *tweet* yang bersesuaian dengan kategori yang diinginkan.
- e. `List<String>` digunakan untuk menyimpan lokasi dari setiap *tweet*.

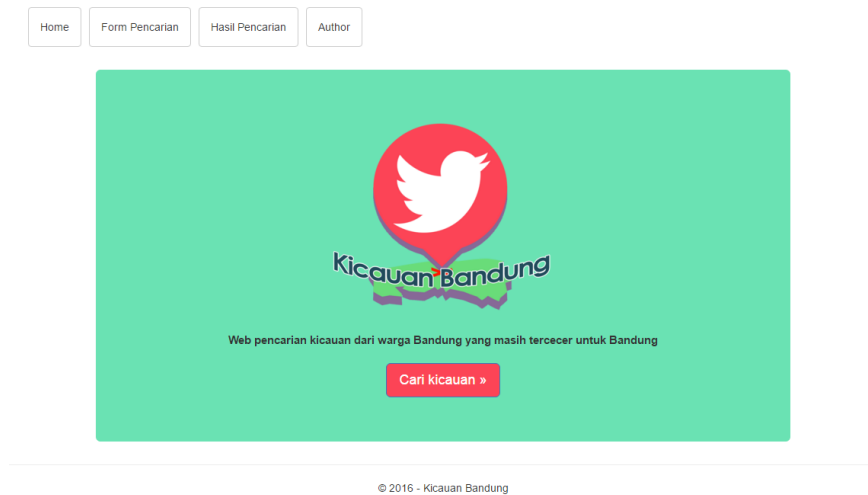
B. Fungsi dan Prosedur

Semua fungsi dan prosedur berikut ada pada file `HomeController.cs` :

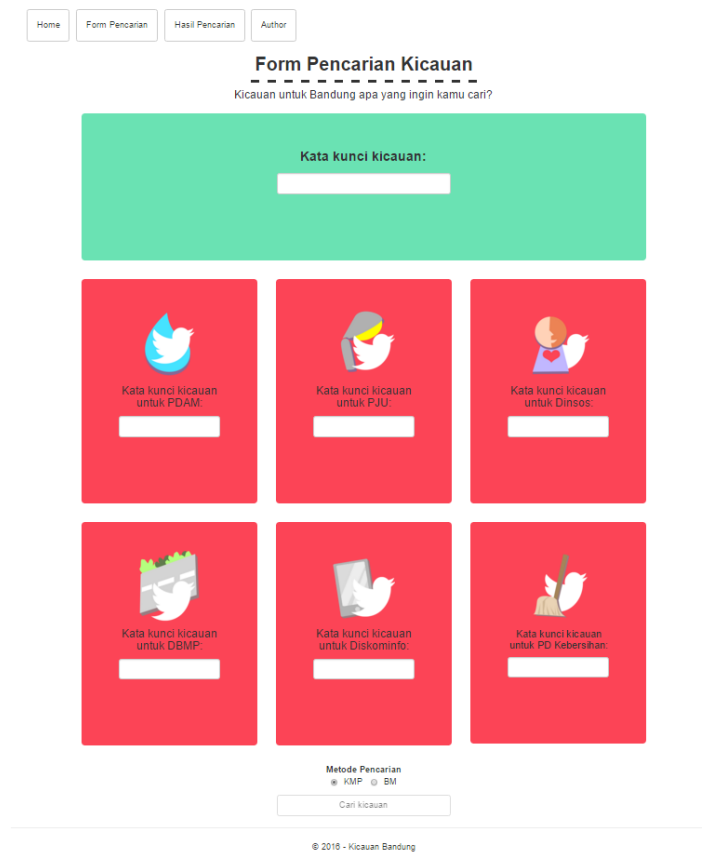
1. `buildTable` → untuk membangun tabel yang akan digunakan untuk pencarian dengan algoritma KMP.
2. `KMP` → untuk mencari posisi indeks *keyword* pada teks dengan menggunakan algoritma KMP.
3. `compute_last` → untuk mencatat posisi indeks terakhir untuk setiap kemunculan karakter.
4. `BM` → untuk mencari posisi indeks *keyword* pada teks dengan menggunakan algoritma BM.
5. `getLocation` → untuk mendapatkan lokasi pada *tweet*.

C. Eksperimen / Pengujian

1. Program dijalankan



2. Tekan tombol “Cari kicauan >>”




3. Masukkan kata kunci tweet dan kata kunci pada setiap dinas, serta pilihlah algoritma pencocokan string yang diinginkan

[Home](#) [Form Pencarian](#) [Hasil Pencarian](#) [Author](#)


Form Pencarian Kicauan

Kicauan untuk Bandung apa yang ingin kamu cari?


Kata kunci kicauan:




Kata kunci kicauan untuk PDAM:




Kata kunci kicauan untuk PJU:




Kata kunci kicauan untuk Dinsos:



Kata kunci kicauan untuk DBMP:



Kata kunci kicauan untuk Diskominfo:




Kata kunci kicauan untuk PD Kebersihan:

Metode Pencarian
☐ KMP ☒ BM

© 2016 - Kicauan Bandung

4. Keluaran program



Kicauan untuk Diskominfo


Kata kunci untuk kicauan: informasi
Jumlah: 0

Kicauan untuk PD Kebersihan

Kata kunci untuk kicauan: sampah; bersih
Jumlah: 3



vikingsukajadi - Bandung
RT @pdkebersihanbdg: Selamat utk Kafilah MTQ @PemkotBandung. Smg juaranya m'inspirasi warga utk ++ sholeh yg dibuktikn dgn mjaga kBERSIHn h...
23/4/2016 18:48:37




PD KEBERSIHAN - Jl. Surapati No. 126
Selamat utk Kafilah MTQ @PemkotBandung. Smg juaranya m'inspirasi warga utk ++ sholeh yg dibuktikn dgn mjaga kBERSIHn <https://t.co/PexHE76U9x>
23/4/2016 18:22:55




#GerakanPungutSampah - Indonesia
RT @kel_skwarna: giat bebersih hari ini ... @Keo_Sukajadi @PemumBdg @PemkotBandung @BDGcleanaction @GPSbdg @DiskominfoBdg <https://t.co/9xknJ...>
23/4/2016 10:30:34

Kicauan untuk dinas lainnya

Jumlah: 82



fanda yp -
Twitter saya di hack omg tidaakkk. dibajak jugaa di itb #TestingTubes #abaikan #inidebugtugas #pemkotbdg
24/4/2016 11:12:53



Laporan Tugas Besar 3
IF2211 Strategi Algoritma

15

5. Author



Author



Micky Yudi Utama
13514011



Ade Yusuf Rahardian
13514079



Atika Azzahra Akbar
13514077

© 2016 - Kicauan Bandung

D. Analisis Hasil Pengujian

Pada pengujian di atas, didapatkan 1 tweet berhubungan dengan PDAM, 3 tweet berhubungan dengan PJU, tidak ada tweet yang berhubungan dengan Dinsos, 11 tweet berhubungan dengan DBMP, tidak ada tweet berhubungan dengan Diskominfo, 3 tweet berhubungan dengan PD Kebersihan, dan 82 tweet yang berhubungan dengan dinas lainnya. Dari hasil di atas, didapatkan dinas dengan respons masyarakat terbesar merupakan DBMP. Selain itu, dapat dilihat bahwa tweet yang berhubungan dengan dinas lainnya cukup banyak. Hal ini dikarenakan di kota Bandung tidak hanya terdapat 6 dinas melainkan sekitar 20 dinas.

Bab V

Kesimpulan dan Saran

A. Kesimpulan

Algoritma Knuth-Morris-Prath dan/atau Boyer-Moore adalah algoritma yang tepat untuk menyelesaikan masalah pencocokan string. Kedua algoritma ini lebih cepat dibandingkan dengan Brute Force. Algoritma KMP tidak pernah bergerak mundur pada teks input, hal ini menyebabkan algoritma ini bagus untuk memproses file yang sangat besar dan dibaca dari eksternal. Namun algoritma KMP tidak bekerja cukup baik dengan meningkatnya variasi alfabet. Hal ini karena semakin tingginya kemungkinan *mismatch* dan terjadinya *mismatch* cenderung terjadi pada tiap awal pattern. Namun KMP bekerja cepat jika mismatch terjadi di akhir *pattern*. Sebaliknya dengan algoritma Boyer-Moore, pada Boyer-Moore bekerja dengan cepat jika variasi alfabetnya banyak dan bekerja lambat jika variasi alfabetnya sedikit.

B. Saran

Secara umum, apabila kita ingin melakukan pencocokan string pada text biasa, sebaiknya menggunakan algoritma Boyer-Moore daripada KMP. Algoritma KMP lebih baik untuk pencarian pada string seperti DNA (string dengan variasi alfabet sedikit).

Daftar Pustaka

Munir, Rinaldi. 2014. Slide Kuliah IF2211 Strategi Algoritma : Pencocokan String (2015). Bandung : Institut Teknologi Bandung.

<https://mcansozeri.wordpress.com/2013/07/02/blog-posthow-to-get-an-users-timeline-using-twitter-api-v1-1-with-tweetsharp-updated/> diakses pada tanggal 13 April 2016

<http://www-igm.univ-mlv.fr/~lecroq/string/node8.html> diakses 22 April 2016 pukul 10.00.

<http://www-igm.univ-mlv.fr/~lecroq/string/node14.html> diakses 22 April 2016 pukul 10.00.

<http://www.jmelosegui.com/map/Services/Geocoding> diakses pada tanggal 23 April 2016