# Pemodelan *Chatbot* menggunakan Algoritma *Knuth-Morris-Pratt*, *Boyer-Moore*, dan *Regular Expression*

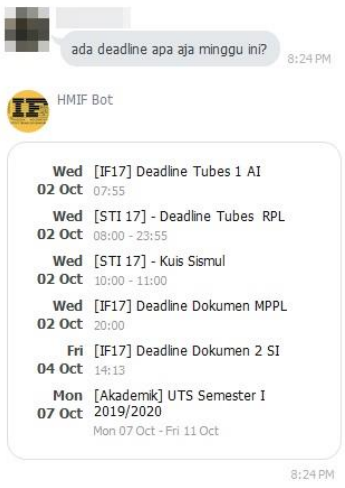
Muhammad Firas | 13518117
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518117@std.stei.itb.ac.id

***Abstract*—Kemajuan teknologi memberi banyak dampak positif kepada berbagai macam industri dalam berbagai aspek. Pada dasarnya, dengan kemajuan teknologi yang ada, maka kinerja bisnis pada beberapa perusahaan bisa dilakukan dengan lebih mudah dan hanya perlu mengandalkan teknologi. Salah satu dari kemajuan teknologi yang memberi dampak positif adalah *chatbot. Chatbot* telah digunakan pada berbagai aplikasi sosial media. *Chatbot* ini dapat diimplementasikan dengan berbagai macam cara. Oleh karena itu, penulis menggunakan algoritma *Knuth-Morris-Pratt, Boyer-Moore,* dan *regular expression* dalam penerapan *chatbot* agar dapat menerima respon dari lawan bicaranya dan memberi pesan balasan.**

***Kata kunci—chatbot; Algoritma Knuth-Morris-Pratt; Algoritma Boyer-Moore; Regular Expression.***

## I. PENDAHULUAN

*Chatbot* merupakan program komputer atau terkadang kecerdasan buatan yang berupa suatu layanan yang secara fundamental mensimulasikan percakapan manusia. Ini memungkinkan suatu bentuk interaksi antara manusia dan mesin yang terjadi melalui pesan atau perintah suara. Beberapa *chatbot* diprogram untuk bekerja secara independen dari operator manusia, seperti *chatbot* yang menggunakan *Natural Language Processing* (NLP). Akan tetapi, ada juga *chatbot* yang hanya memproses pesan dari lawan bicaranya dengan mencocokkan *string* suatu pesan agar *chatbot* dapat merespon lawan bicaranya.




Gambar 1.1. *Contoh sebuah Chatbot*

*Chatbot* yang akan penulis bahas adalah jenis *chatbot* yang lebih simpel dan tidak menggunakan kecerdasan buatan (AI), yaitu dengan menggunakan algoritma *Knuth-Morris-Pratt* (KMP), *Boyer-Moore* (BM), dan *regular expression*. Ketiga algoritma ini mencocokkan *string* yang sesuai dengan pola yang dicari oleh bot. Pada dasarnya algoritma ini bekerja dengan cara melihat kata kunci dalam data yang masuk dan membalasnya dengan kata kunci yang paling sesuai dengan pola kata-kata yang dikirim oleh lawan bicara *chatbot*. Artinya, jika lawan bicara mengirim suatu pesan maka *chatbot* akan membalasnya dengan respon yang spesifik sesuai dengan kata kunci yang dikirim. Sebagai contoh, misalnya lawan bicara mengirimkan suatu pesan "Ada deadline apa saja minggu ini?" maka dengan informasi yang sudah tersedia, *chatbot* akan segera merespon sesuai dengan pesan yang dikirim lawan bicara tersebut "(Mon, 4 May), [IF 18] Deadline Makalah STIMA"

Ide penulisan makalah ini bermula dari rasa penasaran penulis tentang pengimplementasian *chatbot* HMIF di sosial media LINE. *Chatbot* HMIF ini dibuat oleh mahasiswa senior Teknik Informatika yang berguna sebagai pengingat suatu tugas dan kegiatan lainnya. Untuk melakukan penelitian ini, penulis menggunakan algoritma *Knuth-Morris-Pratt* (KMP), *Boyer-Moore* (BM), dan *regular expression* untuk menganalisis bagaimana *chatbot* ini bekerja.

## II. LANDASAN TEORI

### A. Algoritma Brute-Force

Algoritma *Brute Force* membandingkan pola dengan teks, satu karakter pada satu waktu, hingga karakter yang tidak cocok ditemukan. Misal terdapat teks dengan panjang karakter m dan *pattern* dengan panjang karakter n, maka algoritmanya dalam *pseudocode* adalah seperti berikut

```
Algorithm BruteForceStringMatch(T[0...n-1], P[0...m-1])
        for i ← 0 to n-m do
                j ← 0
                while j < m and P[j] = T[i+j] do
                        j++
                if j = m then return i
        return -1
```

Pada kasus rata-rata algoritma ini memiliki kompleksitas waktu O(m+n).

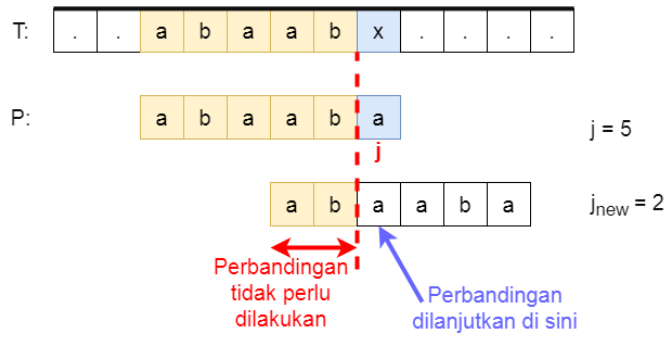
*pattern :* the
teks : tetththehe

```
   0 1 2 3 4 5 6 7 8 9
   t e t t h t h e h e
1. t h e
2.   t h e
3.     t h e
4.       t h e
5.         t h e
6.           t h e
```

Pada contoh di atas, beberapa perbandingan dinilai tidak efisien. Misalkan pada iterasi ke-4, diketahui bahwa T[4] = 'h' sehingga pada iterasi ke-5 tidak perlu membandingkan P[0] = 't' dengan T[4] = 'h'. Karena algoritma ini akan tetap memeriksa satu per satu, pengecekan dengan algoritma *Brute-Force* dianggap kurang cepat.

### B. *Algoritma Knuth-Morris-Pratt*

Untuk pencarian *string* menggunakan algoritma *brute force*, setiap kali ditemukan ketidakcocokkan *pattern* dengan teks, maka *pattern* akan digeser satu karakter ke kanan. Sedangkan algoritma *Knuth-Morris-Pratt* (KMP) melakukan hal yang serupa tetapi pergeseran yang dilakukan algoritma KMP lebih pintar dari *brute force*. Pada algoritma KMP, jika ketidakcocokkan antara teks T dan *pattern* P terjadi pada P[j], yaitu T[i] ≠ P[j] dan misal s adalah panjang prefiks terbesar dari P[0..j-1] yang juga merupakan sufiks dari P[1..j-1] maka pergeseran dilakukan sebesar j – s dan perbandingan pada *pattern* P dilanjutkan pada indeks ke-s.




Gambar 2.1. *Visualisasi algoritma Knuth-Morris-Pratt*
Sumber : *Pencocokan String (2018) Strategi Algoritma Rinaldi Munir*

Pada contoh di atas, ketidakcocokkan terjadi pada j = 5, dan kita mengetahui bahwa prefiks terpanjang pada "abaab" yang juga merupakan sufiks dari "baab" adalah "ab", s = 2 sehingga pergeseran dilakukan sebesar j – s = 3 karakter ke kanan dan perbandingan dilanjutkan pada P[s] = P[2].

Untuk mengetahui prefiks terbesar dari P[0..j-1] yang merupakan sufiks dari P[1..j-1] pada program, kita harus menghitung fungsi pinggiran KMP (KMP *border function*). Fungsi pinggiran ini juga biasa disebut *failure function*. Misalkan j = posisi ketidakcocokkan pada P[] dan k = posisi sebelum terjadinya ketidakcocokkan pada P[] (k = j-1) maka fungsi pinggiran b(k) adalah panjang dari prefiks terbesar dari P[0..k] yang merupakan sufiks dari P[1..k].

P : abaaba
j : 012345

| | k = j-1 | | | | | |
|---|---|---|---|---|---|---|
| j | 0 | 1 | 2 | 3 | 4 | 5 |
| P[j] | a | b | a | a | b | a |
| k | - | 0 | 1 | 2 | 3 | 4 |
| b(k) | - | 0 | 0 | 1 | 1 | 2 |

b(k) adalah panjang dari prefiks terbesar dari P[0..k] yang juga merupakan sufiks dari P[1..k]

Gambar 2.2. *Contoh fungsi pinggiran KMP*
Sumber : *Pencocokan String (2018) Strategi Algoritma Rinaldi Munir*

Untuk merepresentasikan algoritma ini, maka dibuat *pseudocode* seperti berikut.

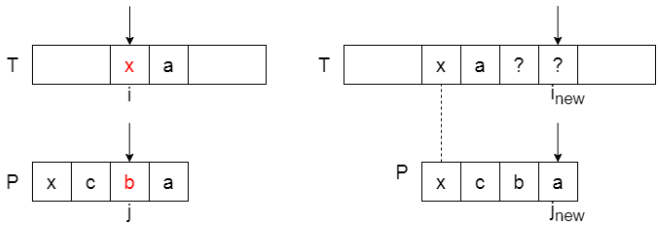
```
Algorithm KMPSearch(T[0..n-1], P[0..m-1])
bord[] ← borderfunction(P)
i ← 0
j ← 0
while (i < n) do
    if (P[j] = T[i]) then
        if (j = m - 1)
            return i-m+1
        endif
        i ← i + 1
        j ← j + 1
    else if(j > 0) then
        j ← bord[j-1]
    else
        i ← i + 1
    endif
endwhile
return -1
```

Kelebihan dari algoritma KMP ini adalah tidak perlu bergerak mundur dalam input teks T, sehingga membuat algoritma ini baik untuk memproses file yang sangat besar yang dibaca dari perangkat eksternal.

### C. *Algoritma Boyer-Moore*

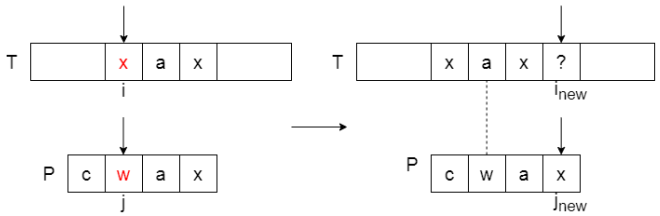
Algoritma *Boyer-Moore* (BM) adalah salah satu algoritma untuk mencari suatu *string* di dalam teks. Ide di balik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat. Algoritma *Boyer-Moore* bekerja dengan menggunakan dua teknik. Pertama, teknik *looking-glass*, yaitu membandingkan teks dan *pattern* dengan cara mundur dimulai dari indeks paling kanan dari P sampai indeks paling kiri. Kedua, teknik *character-jump*, yaitu ketika T[i] = x dan ketidakcocokkan terjadi pada T[i] ≠ P[j], maka terdapat tiga kasus yang mungkin.

Kasus pertama, jika P mengandung x di dalamnya, maka pergeseran dilakukan dengan menyamakan posisi x di P dan x di T, lalu perbandingan dilakukan kembali dengan teknik *looking-glass*.




Gambar 2.3. *Visualisasi kasus pertama teknik character-jump*
Sumber : *Pencocokan String (2018) Strategi Algoritma Rinaldi Munir*

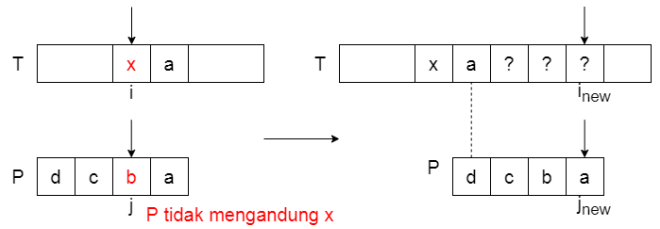
Kasus kedua, jika P mengandung x di dalamnya, namun pergeseran ke kanan sampai indeks terakhir terdapatnya x tidak mungkin dilakukan, maka pergeseran P dilakukan sebanyak satu karakter ke kanan ke T[i+1].




Gambar 2.4. *Visualisasi kasus kedua teknik character-jump*
Sumber : *Pencocokan String (2018) Strategi Algoritma Rinaldi Munir*

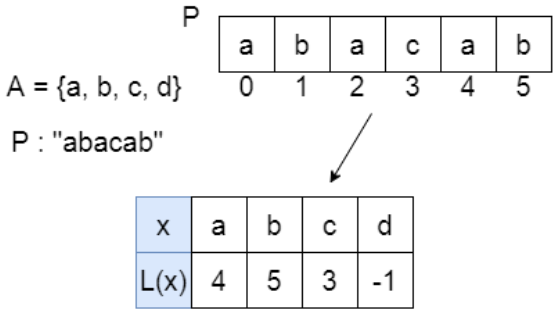
Pada contoh gambar di atas, ketidakcocokkan terjadi pada j = 2, *pattern* P mengandung karakter x, akan tetapi karakter x berada di kanan j, sehingga tidak mungkin melakukan pergeseran mundur, maka pergeseran dilakukan ke kanan sebanyak 1 karakter.

Kasus ketiga, jika kasus pertama dan kasus kedua tidak memungkinkan, maka pergeseran P dilakukan dengan menyamakan posisi P[0] dengan T[i+1].




Gambar 2.5. *Visualisasi kasus ketiga teknik character-jump*
Sumber : *Pencocokan String (2018) Strategi Algoritma Rinaldi Munir*

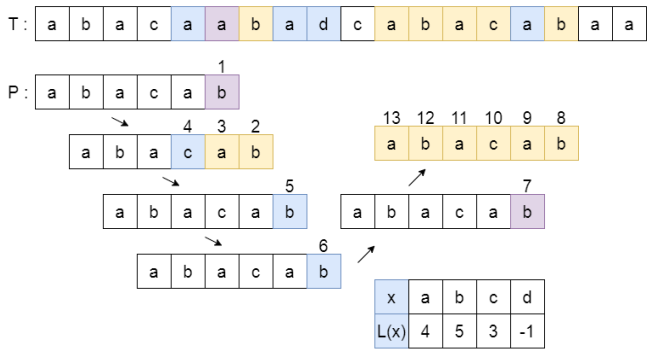
Sebelum memproses perbandingan karakter, algoritma *Boyer-Moore* memiliki fungsi *last occurence* L(), fungsi L ini memetakan seluruh karakter yang ada di teks T menjadi *integer*. L(x) dengan x adalah karakter di teks T didefinisikan sebagai indeks terbesar i di mana P[i] = x atau -1 jika indeks tidak ditemukan. Fungsi L ini digunakan untuk melakukan teknik *character-jump* pada kasus pertama.




Gambar 2.6. *Contoh fungsi last occurence L(x)*
Sumber : *Pencocokan String (2018) Strategi Algoritma Rinaldi Munir*

Melalui ide di atas, kita dapat membentuk *pseudocode* untuk algoritma *Boyer-Moore* ini.

```
Algorithm BMMatch(T, P):
Input: String T (text with n characters and P (pattern)
with m characters
Output: Starting index offirst substring of T matching p,
or an indication that P is not a substring of T

i <-- m - 1
j <-- m - 1
repeat
   if P[j] = T[i] then
        if j = 0 then
             return i {a match!}
        else {check next character}
             i <-- i - 1
             j <-- j - 1
   else { P[j] <> T[i] move the pattern}
        i <-- i + m - j - 1 {reset i to where it began in
most-recent test}
        i <-- i + max(j - last(T[i]), match(j)) {shift P
relative to T}
        j <-- m-1
until i > n - 1
return "There is no substring of T matching P."
```




Gambar 2.7. *Visualisasi algoritma Boyer-Moore*
Sumber : Pencocokan String (2018) Strategi Algoritma, Rinaldi Munir

Pencarian dengan algoritma *Boyer-Moore* ini lebih cepat dari algoritma *brute force* dan *Knuth-Morris-Pratt* untuk teks berbahasa inggris, karena algoritma *Boyer-Moore* cepat ketika

karakter pada teksnya beragam dan lambat ketika hanya memiliki karakter yang unik, misalnya *binary*.

### *D. Regular Expression*

*Regular Expression* (Regex) adalah sebuah notasi yang dapat digunakan untuk mendeskripsikan pola dari kata yang ingin dicari. Sintaks dari Regex berupa kumpulan karakter khusus yang digunakan untuk menentukan pola pencarian pada teks. Sintaks Regex memiliki beberapa aturan seperti berikut.

i. Character Classes

Aturan ini membentuk character class dalam konstruksi regex.

| 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 (substraksi) |
| `[a-z&&[^m-p]]` | a sampai z, dan bukan m sampai p (substraksi) |

Tabel 2.1. *Regex Character Classes*

ii. Predefined Character Class

Predefined class membuat Regex lebih mudah dibaca dan mengurangi kesalahan.

| 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 2.2 *Regex Predefined Character Class*

iii. Quantifier

Quantifier digunakan untuk mendefinisikan jumlah perulangan pola.

| 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 2.3 *Regex Quantifier*

iv. Boundary Matchers

Boundary matcher digunakan untuk mencari pola yang muncul di posisi tertentu. Misalnya di awal atau di akhir.

| 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 |

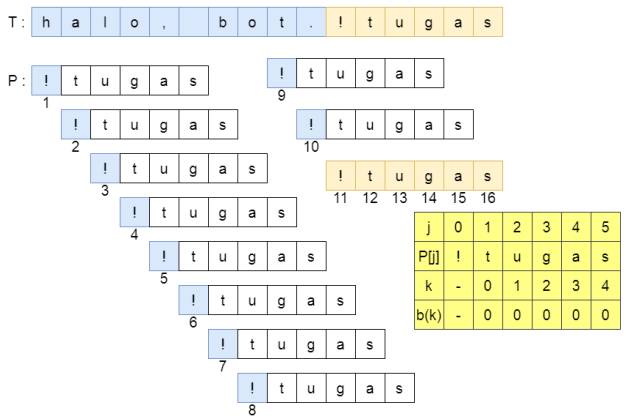
Tabel 2.4. *Regex Boundary Matchers*

## III. PEMODELAN CHATBOT

Seperti yang sudah dijelaskan sebelumnya, *chatbot* yang akan penulis teliti bukanlah *chatbot* yang menggunakan kecerdasan buatan (AI). Pengembangan *chatbot* ini tidak terlalu rumit karena *chatbot* selalu mengeluarkan pesan yang sama yang telah ditentukan sebelumnya. *Chatbot* yang penulis coba untuk dikembangkan hanyalah sebuah prototipe sederhana dan tidak menggunakan basis data untuk menyimpan jenis-jenis pesan balasan. *Chatbot* dibuat dengan mencari *pattern* "!tugas" pada pesan, lalu akan membalas dengan menampilkan pesan sesuai dengan masukan pengguna. Program ditulis dengan menggunakan bahasa Python.

### A. Menggunakan algoritma *Knuth-Morris-Pratt*

Perhatikan analisis perbandingan *pattern* P dan teks T pada gambar berikut.



Gambar 3.1. *Proses perbandingan dengan algoritma Knuth-Morris-Pratt*

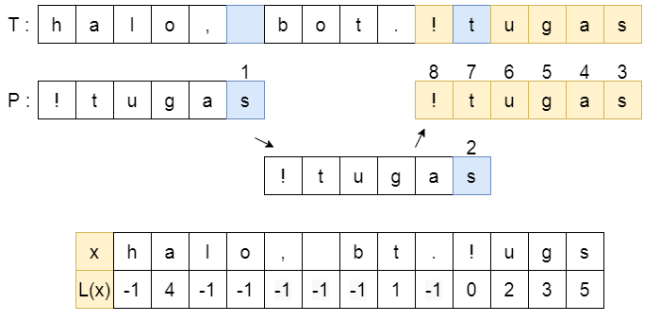
```
User : halo, bot. !tugas

Bot : (Mon, 4 May), [IF 18] Deadline Makalah STIMA

Waktu eksekusi program: 23.521 mikrosekon
```

Gambar 3.2. *hasil Screenshot program dengan algoritma Knuth-Morris-Pratt*

Pada contoh di atas, algoritma KMP melakukan pencarian seperti algoritma *brute force* dikarenakan ketidakcocokkan terjadi di awal *pattern*, sehingga ini memakan waktu lebih banyak untuk jenis pesan yang memiliki banyak karakter unik.

B. Menggunakan algoritma *Boyer-Moore*

Perhatikan analisis perbandingan *pattern* P dan teks T pada gambar berikut.




Gambar 3.3 *Proses perbandingan dengan algoritma Boyer-Moore*

```
User : halo, bot. !tugas

Bot : (Mon, 4 May), [IF 18] Deadline Makalah STIMA

Waktu eksekusi program: 12.829 mikrosekon
```

Gambar 3.4. *hasil Screenshot program dengan algoritma Boyer-Moorer*

Pada perbandingan dengan algoritma *Boyer-Moore* di atas, jumlah perbandingan lebih sedikit dan waktu eksekusi program juga lebih cepat. Ini dikarenakan banyaknya karakter unik pada teks dan *pattern* dan ketidakcocokkan terjadi di awal perbandingan. Sehingga, teknik *character-jump* yang dilakukan menggeser posisi P[0] ke posisi T[i+1] dan jumlah perbandingan yang dilakukan menjadi lebih sedikit dibandingkan dengan algoritma *Knuth-Morris-Pratt*.

Namun algoritma *Boyer-Moore* tidak selalu lebih cepat dari algoritma KMP. Jika lawan bicara *chatbot* mengirimkan pesan yang sedikit memiliki karakter unik, contohnya "ada tugas? tugas apa? !tugas", maka pada kasus ini algoritma KMP memiliki waktu yang lebih cepat daripada algoritma *Boyer-Moore*. Perhatikan contohnya pada kasus berikut.

```
ada tugas? tugas apa? !tugas

Bot : (Mon, 4 May), [IF 18] Deadline Makalah STIMA

Waktu eksekusi program: 11.119 mikrosekon
```

Gambar 3.4. *hasil Screenshot program dengan algoritma KMP untuk sedikit karakter teks yang unik*

```
ada tugas? tugas apa? !tugas

Bot : (Mon, 4 May), [IF 18] Deadline Makalah STIMA

Waktu eksekusi program: 25.66 mikrosekon
```

Gambar 3.5 *hasil Screenshot program dengan algoritma Boyer-Moore untuk sedikit karakter teks yang unik*

C. Menggunakan Regular Expression

Pada Python, digunakan import re untuk menggunakan Regex .Karena *string* yang dicari hanya berupa satu kata yaitu "!tugas", maka pada Python hanya perlu menuliskan sintaks berikut.

```
re.findall(r'!tugas', text)
```

```
ada tugas? tugas apa? !tugas

Bot : (Mon, 4 May), [IF 18] Deadline Makalah STIMA

Waktu eksekusi program: 154.811 mikrosekon
```

Gambar 3.6 *hasil Screenshot program dengan regex*

Berdasarkan hasil program, Regex memiliki waktu eksekusi yang lebih lama dibandingkan algoritma KMP dan *Boyer-Moore*. Akan tetapi, berbeda dengan algoritma KMP dan *Boyer-Moore* yang pencocokkannya harus tepat, *chatbot* dengan Regex dapat dikembangkan agar dapat merespon berbagai jenis pesan berbeda tetapi polanya sama. Misalkan *chatbot* akan merespon untuk pesan yang memiliki pola *string* "ada tugas apa minggu ini". Meskipun pesan yang dikirim berbeda-beda, dengan menggunakan Regex dapat mencari kesamaan pola pada pesan tersebut. Maka dibuat sintaks Regex seperti berikut.

```
re.findall(r'
ada\s+(.+\s+)*tugas\s+(.+\s+)*apa\s+(.+\s+)*minggu\s+(.+\
s+)*ini', text)
```

```
halo bot, ada tugas ya? emang apa saja minggu ini deadlinenya?

Bot : (Mon, 4 May), [IF 18] Deadline Makalah STIMA

Waktu eksekusi program: 11633.492 mikrosekon
```

Gambar 3.7 *hasil Screenshot program dengan Regex dengan cara mendeteksi pola teks*

## IV. KESIMPULAN

Pada algoritma KMP dan *Boyer-Moore* jenis *chatbot* yang dapat diimplementasikan adalah *chatbot* yang menerima *exact matching* dan tidak dapat mendeteksi *string* yang memiliki isi pesan berbeda namun memiliki pola sama, berbeda halnya dengan Regular Expression. Kasus pada umumnya, pesan teks yang dikirimkan oleh pengguna biasanya banyak memiliki karakter unik, sehingga algoritma *Boyer-Moore* dianggap lebih efisien untuk pembuatan *chatbot*. Namun, jika *chatbot* ingin dibuat dengan tidak memiliki kata kunci spesifik, melainkan pola suatu kalimat, maka Regular Expression merupakan pilihan yang tepat.

## VIDEO LINK AT YOUTUBE

https://youtu.be/rx1W3qWZaCc

## PENUTUP

Pertama-tama penulis ingin mengungkapkan rasa syukur kepada Tuhan Yang Maha Esa karena penulis dapat menyelesaikan makalah Pemodelan *Chatbot* menggunakan Algoritma *Knuth-Morris-Pratt*, *Boyer-Moore*, dan *Regular Expression* ini. Penulis juga ingin mengucapkan rasa terima kasih yang sebesar-besarnya kepada bapak Dr. Ir. Rinaldi Munir, MT. selaku dosen pengajar mata kuliah Strategi Algoritma yang telah membimbing penulis dalam membuat makalah ini.

## REFERENSI

[1] Munir, Rinaldi, *Pattern Matching Strategi Algoritma* Bandung : Informatika, 2018
http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

[2] Wibisono, Yudi, *Modul Praktikum Kuliah Pengantar Regular Expression*, Bandung, Informatika, 2020
https://docs.google.com/document/d/1ls6h1A6m-Zhzw6e5eriwMNUAG0D1iwL-eVmVMS2XQoc/edit

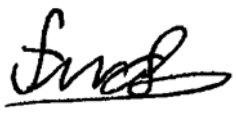
[3] https://searchcustomerexperience.techtarget.com/definition/chatbot, diakses pada 2 Mei 2020.

[4] https://www.wartaekonomi.co.id/read219026/apa-itu-chatbot, diakses pada 2 Mei 2020.

## 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, 3 Mei 2020



Muhammad Firas/13518117