Pencocokan String pada Aplikasi Chatbot Sederhana

Choirunnisa Fatima 13512084
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512084@std.stei.itb.ac.id

***Abstrak*—Chatbot adalah salah satu produk Intelejensia Buatan yang melakukan percakapan dengan manusia. Dengan kemampuan yang dia punya, dia dapat menjawab dan menanggapi percakapan yang kita lakukan dengannya. Semakin cerdas chatbot, maka semakin baik respon yang dia berikan kepada manusia. Kunci dari program chatbot ini adalah pencocokan string. Program menyimpan data-data dan pengetahuannya tentang percakapan, lalu mencari jawaban atas pertanyaan yang diajukan oleh pengguna dengan melakukan pencocokan string pada basis data tersebut. Dalam makalah ini, dituliskan metode untuk meningkatkan kecerdasan chatbot dengan memanfaatkan algoritma pencocokan string.**

***Istilah Indeks*—chatbot, NLP, pattern matching, pencocokan string.**

I. PENDAHULUAN

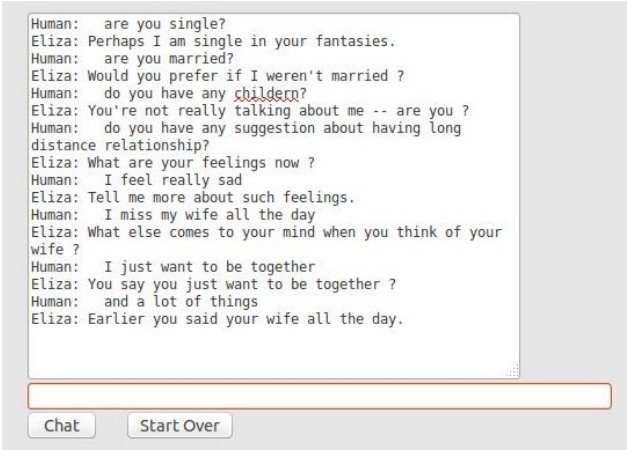
NLP adalah kepanjangan dari *Natural Language Processing*, sebuah sub bidang dari Intelejensia Buatan (*Artificial Intelligence*). NLP mempelajari dan menganalisis bahasa manusia, yang nantinya akan digunakan sebagai antarmuka komputer dengan manusia. Antarmuka tersebut dapat dalam bentuk tertulis atau lisan bukan dalam bahasa komputer, melainkan dalam bahasa manusia.

Tantangan dalam sub bidang NLP ini adalah mengajari komputer untuk mengerti cara manusia menggunakan bahasa. Misalkan sebuah potongan kalimat "Tukang gigi palsu". Potongan kalimat tersebut mempunyai lebih dari satu arti, tergantung pada cara pandang. Manakah yang palsu? Tukang giginya yang palsu, ataukah giginya yang palsu? Kalimat seperti ini biasa disebut dengan ambigu. Kalimat ini akan menjadi masalah bagi perangkat lunak, kecuali perangkat lunak itu dapat memahami konteks dan struktur bahasa yang manusia gunakan.

Salah satu aplikasi yang merupakan pengembangan dari *Natural Language Processing* adalah *chatbot*. Chatbot adalah manusia buatan, binatang buatan atau makhluk buatan lainnya yang melakukan percakapan dengan manusia. Chatbot dapat berupa percakapan tertulis atau lisan. Chatbot dapat kita persepsikan sebagai perangkat lunak yang dapat kita ajak bicara. Menarik ketika kita dapat berbicara pada komputer, lalu komputer itu menanggapi percakapan kita.

Chatbot yang dikenal sebagai chatbot intelejensia buatan pertama adalah chatbot Eliza. Eliza dikembangkan oleh profesor MIT bernama Joseph Weizenbaum pada tahun 1966.

Berikut merupakan salah satu percakapan antara seseorang dengan Eliza, chatbot yang berkarakter seperti psikolog, dia hanya membalikkan hal-hal yang kita ucapkan sebelumnya.



Gambar 1: Percakapan dengan Eliza

Boleh dibilang chatbot adalah sebuah program simulasi percakapan manusia. Ketika pengguna mengatakan A, program akan mengatakan B. Ketika pengguna menanyakan tentang X, maka program akan menjawab tentang X juga. Bisa jadi program menyimpan semua bentuk percakapan yang mungkin dibuat manusia, lalu mencari kalimat yang sesuai untuk menanggapi percakapan pengguna. Tetapi jika benar begitu jalannya, tentu program ini akan berukuran sangat besar, mengingat betapa banyakanya kalimat yang mungkin dikatakan oleh pengguna.

Terdapat banyak metode yang dapat digunakan untuk membangkitkan respon program terhadap pengguna. Salah satu metode yang paling mudah adalah menggunakan pencocokan string (*pattern matching*). Dengan pencocokan string, program akan mencari dan mencocokkan masukan pengguna dengan *pattern-pattern* yang telah disimpan di dalam program. Kemudian program akan memberikan respon sesuai dengan *pattern* yang cocok.

II. TEORI

Algoritma pencocokan string atau *pattern matching*

adalah algoritma yang melakukan pencocokan dua buah string, yaitu string pendek yang disebut *pattern* dan string yang lebih panjang yang disebut teks. Algoritma ini melakukan pencarian semua kemunculan *pattern* pada teks.

Algoritma pencocokan string dapat diklasifikasikan menjadi dua kategori, yaitu *Exact Pattern Matching* dan *Approximate String Matching.* Kedua kategori ini berbeda dalam hal pencocokannya. Algoritma-alagoritma pencocokan string seperti Brute Force, Boyer-Moore, dan Knuth-Morris-Pratt termasuk ke dalam *Exact Pattern Matching.* Algoritma *Exact Pattern Matching* melakukan pencocokan tepat sesuai dengan *pattern.* Sedangkan *Approximate String Matching* atau *Fuzzy String Search* melakukan pencocokan sesuai perkiraan. Biasanya dalam mesin pencari Google, algoritma ini digunakan untuk membenarkan kata yang salah tulis, lalu memunculkan "*Did you mean ..*" atau "Apakah yang kamu maksud adalah …?".

Salah satu algoritma *Approximate String Matching* bergantung pada pemrograman dinamis. Pemrograman ini menggunakan formulasi alternatif pada persoalan yang ditinjau : untuk setiap j posisi dalam teks T dan setiap posisi i dalam *pattern* P, hitung jarak minimum antara i karakter pertama dari *pattern* ,P_i , dan setiap substring $T_{j'j}$ dari T yang berakhir di posisi j .

Untuk setiap j posisi pada teks T, dan setiap posisi i dalam *pattern* P , melalui semua substring dari T berakhir di posisi j , dan menentukan salah satu dari keduanya memiliki jarak minimal terhadap i karakter pertama dari *pattern* P. Kita tuliskan jarak minimal ini sebagai *E*(*i, j*). Setelah menghitung *E*(*i, j*). untuk semua i dan j , kita dapat dengan mudah menemukan solusi untuk masalah awal : yaitu substring dengan *E*(*i, j*) minimal (*m* menjadi panjang *pattern P*.)

Penghitungan E(m, j) sangat mirip dengan penghitungan jarak antara dua string . Kita bahkan bisa menggunakan algoritma komputasi jarak Levenshtein untuk E(i, j), Satu-satunya perbedaan adalah bahwa kita harus menginisialisasi baris pertama dengan nol, dan menyimpan jalan perhitungan , yaitu, apakah kita menggunakan E (i - 1 , j) , E (i , j - 1) atau E (i - 1 , j - 1) dalam menghitung E (i , j).

Dalam baris yang berisi nilai *E(x , y)* , kita kemudian memilih nilai minimal di baris terakhir , misalkan *E(x_2 , y_2)* , dan mengikuti jalan perhitungan mundur , kembali ke baris 0 . Jika kita sampai pada bidang *E (x_1 , 0)* , maka *T [x_1 + 1] ... T [y_2]* adalah substring dari T dengan jarak minimal dengan *pattern* P.

Waktu yang diperlukan dalam penghitungan baris dengan algoritma pemrograman dinamis adalah O(mn) sedangkan dengan algoritma runut balik adalah O(n+m).

A. Fuzzy String Search dan Levenshtein Distance

Persoalan *Fuzzy String Search* dapat dituliskan sebagai berikut. "Carilah di dalam teks atau kamus berukuran **n** semua kata yang cocok dengan kata yang diberikan (atau dimulai dengan kata tersebut), dengan banyak **k** perbedaan atau kesalahan yang mungkin."

Untuk mengukur seberapa cocok antara dua string, dibutuhkan sebuah metrik atau fungsi. Salah satu metric itu adalah *Levehstein distance*. Pengukuran dilakukan berdasarkan banyaknya operasi primitif untuk mengubah string menjadi pencocokan yang tepat. Operasi-operasi primitive tersebut antara lain:

- Penyisipan: bisa → biasa
- Penghapusan: biasa → bisa
- Substitusi: biasa → biara

Secara matematis, *Levenshtein distance* antara dua string a, bdapat ditulis:

$$lev_{a,b}(|a|,|b|) = \begin{cases} max\,(|a|,|b|) \\ min \begin{cases} lev_{a,b}(|a|-1,|b|)+1 \\ lev_{a,b}(|a|,|b|-1)+1 \\ lev_{a,b}(|a|-1,|b|-1)+1_{(a_{|a|} \neq b_{|b|})} \end{cases} \end{cases}$$

Sebagai contoh, *Levenshtein distance* antara "kitten" dan "sitting" adalah 3, karena paling sedikit 3 operasi dilakukan untuk mengubah kedua kata tersebut.

1. kitten → sitten (substitusi "s" pada "k")
2. sitten → sittin (substitusi "i" pada "e")
3. sittin → sitting (penyisipan "g" di akhir)

Metrik Levenshtein ini mempunyai batas minimal dan maksimal yang sederhana, termasuk:

1. Perbedaaan ukuran dua buah string.
2. Panjang terjauhnya adalah panjang pada string yang lebih panjang.
3. Nilainya nol jika dan hanya jika keduanya sama panjangnya.
4. Jika keduanya memiliki ukuran yang sama, maka jaraknya dikenal sebagai jarak Hamming.
5. Jarak Levenshtein antara dua buah string tidak lebih besar daripada jumlah jarak Levenshtein dengan string ketiga. (Pertidaksamaan segitiga).

B. Algoritma Boyer-Moore

Algoritma ini melakukan pencocokan string dengan bergerak dari kanan ke kiri. Algoritma Boyer-Moore melakukan pencocokan string berdasarkan pada dua teknik, yaitu teknik *looking-glass* dan teknik *character jump*.

Teknik *looking-glass* melakukan pencarian pattern pada teks dengan bergerak mundur, dimuali dari bagian akhor teks. Pada teknik *character-jump*, ketika ketidakcocokan terjadi pada T[i] == x, maka karakter P[j] pada pattern tidak akan sama dengan T[i].

Pada algoritma ini, ada 3 kasus yang mungkin terjadi.

Kasus 1: Jika pattern berisi "X" di posisi tertentu, maka geser pattern ke kanan sesuai dengan kemunculan "X" di pattern, yaitu teks[i].

Kasus 2 : Jika pattern berisi "X" di posisi tertentu, tapi menggeser pattern ke kanan tidak mungkin dilakukan, maka geser P ke kanan sebesar 1 karakter, yaitu teks[i+1].

Kasus 3 : Jika kasus 1 dan 2 tidak mungkin terjadi, maka geser pattern ke pattern[1] dengan T[i+1].

Algoritima KMP menggunakan fungsi prefix untuk menyelesaikan permasalahan, algoritma Boyer-Moore menggunakan fungsi *Last Occurrence*.

Untuk kasus terburuk, kompleksitas waktu dari algoritma Boyer-Moore adalah O(mn+A). Boyer-Moore cenderung cepat ketika alfabetnya berukuran besar, lambat ketika alphabet berukuran kecil.

C.Algoritma Knuth-Morris-Pratt

Dalam melakukan pencarian, algoritma ini bergerak dari kiri ke kanan. Kompleksitas waktu untuk menghitung fungsi prefix adalah O(m). Kompleksitas waktu untuk pencarian string adalah O(n). Oleh karena itu, kompleksitas waktu algoritma KMP adalah O(m+n).

Keuntungan menggunakan algoritma KMP dalam pencocokan string adalah algoritma ini tidak pernah bergerak mundur pada teks input. Sedangkan kekurangannya adalah algoritma ini tidak bekerja cukup baik ketika ukuran alphabet bertambah besar.

III. ANALISIS MASALAH

A. Katakunci sebagai Pattern

Katakunci dapat berupa kalimat, potongan kalimat atau bahkan sebuah kata yang digunakan program untuk mengenali masukan pengguna. Dengan katakunci itu, program dapat memberi respon yang sesuai dengan masukan pengguna.

Misalkan terdapat sebuah kata kunci “Siapa namamu”. Katakunci ini menentukan respon yang sesuai, yang sebaiknya diberikan oleh program kepada pengguna. Respon-respon yang sesuai dengan katakunci tersebut antara lain: “Namaku Sherbot. Keahlianku mengetahui hal yang orang tidak tau.”; “Kurasa kamu pun tahu siapa aku.”; “Hanya orang cupu yang tidak kenal aku.”; “Beh, masa kamu tidak tau siapa aku sih.”.

Nantinya akan terdapat banyak katakunci yang berbeda. Program akan mencari katakunci yang sesuai untuk setiap masukan pengguna, lalu memberi respon yang sesuai dengan katakunci tersebut.

B. Pencocokan String yang Lebih Fleksibel

Untuk melakukan pencocokan string yang lebih fleksibel, maka akan dilakukan perampingan pattern untuk pencocokan string. Kita tidak akan membuat pattern untuk setiap masukan pengguna, melainkan kita hanya menggunakan pattern yang lebih umum. Misalkan untuk kalimat-kalimat “Aku ingin tidur”, “Aku ingin makan bakso”, “Aku ingin jalan-jalan” dan seterusnya, kita hanya membuat sebuah pattern, yaitu “Aku ingin X” (X adalah peubah).

Program juga akan menggunakan teknik yang sama untuk membangkitkan respon program. Misal untuk kalimat-kalimat “Mengapa kamu ingin tidur?”, “Mengapa kamu ingin makan bakso?”, dan “Mengapa kamu ingin jalan-jalan?”, program hanya memberikan respon “Mengapa kamu ingin X?” dengan X diambil dari masukan pengguna atau perkataan pengguna yang sebelumnya.

Cara lain untuk melakukan pencocokan string yang lebih fleksibel adalah dengan menggunakan konsep *Fuzzy String Search*. Sebelum menggunakan konsep tersebut, masukan pengguna da katakunci kita bagi menjadi dua potongan kalimat yang terpisah. Kedua belah tersebut ditampung dalam vektor yang berbeda. Vektor pertama untuk masukan pengguna dan vektor lainnya untuk menampung kata kunci. Lalu dengan menggunakan *Levenshtein distance* kita dapat menghitung jarak/perbedaan antara dua vektor tersebut.

C. Pengurutan Katakunci

Pengururtan katakunci adalah satu cara untuk program agar dapat memilih katakunci yang paling sesuai ketika terdapat lebih dari satu katakunci yang cocok dengan masukan pengguna.

Misalkan terdapat katakunci-katakunci berikut:

“Aku merasa tidak enak badan” → “Kamu sebaiknya pergi ke dokter.”

“Aku merasa X” → “Mengapa kamu merasa X?”

“Aku X” → “Apakah selalu harus tentangmu?”

Berdasarkan hal tersebut, tentu kita ingin program memilih “Kamu sebaiknya pergi ke dokter.” Sebagai respon yang paling sesuai jika masukan pengguna adalah “Aku merasa tidak enak badan.”, bukan respon-respon yang lain. Oleh karena itu, kata kunci “Aku merasa tidak enak badan” harus memiliki prioritas yang lebih tinggi dibandingkan “Aku merasa X”. Boleh jadi kita memberikan prioritas yang sangat rendah kepada katakunci “Aku X”, karena kita hanya menggunakannya ketika tidak ada katakunci lain yang cocok.

Dengan prioritas ini program dapat menentukan respon yang pas untuk masukan-masukan pengguna yang lain.

D. Transposisi Katakunci

Teknik inilah yang akan membuat program seakan-akan memahami perkataan pengguna dan menanggapinya dengan cara membalikkan perkataan tersebut. Misalkan pengguna berkata, “Aku sedih.”, Program akan menjawab “Mengapa kamu sedih?”.

Bagaimana caranya agar program dapat member respon demikian? Ada dua langkah yang harus dilakukan, yaitu:

– Pengembang harus membuat rangka respon untuk membuat berbagai respon baru. Biasanya menggunakan “*” (*wildcards*). Pada contoh sebelumnya, saya membuat respon berbentuk “Mengapa *?”, yang nantinya “*” akan disubstitusikan dengan masukan pengguna, yaitu “Aku sedih.”. Tapi jika hanya itu yang dilakukan, maka respon program adalah “Mengapa aku sedih?”. Tentu respon tersebut tidak sesuai. Oleh karena itu, terdapat langkah kedua untuk menyempurnakan respon tersebut.
– Transposisi harus dilakukan untuk mengganti kata orang ganti pertama menjadi kata orang ganti kedua,

atau sebaliknya. Sebagai contoh: kamu → aku; aku → kamu. Pada contoh sebelumnya dengan mengganti kata “Aku” menjadi “kamu”, maka respon aka menjadi “Mengapa kamu sedih?”. Inilah respon yang sesuai dengan perkataan pengguna.

E. Lokasi Katakunci

Pada bagian ini, algoritma Boyer-Moore dan KMP cukup berguna untuk melakukan pencocokan string. Mungkin saja program akan mengalami kesulitan ketika bertemu dengan masukan pengguna yang tidak terdapata dalam basisdata katakuncinya. Seperti ketika pengguna member masukan “Coba dong ceritakan siapa kamu.”. Masukan pengguna tersebut belum tentu terdapat dalam basisdata katakunci milik program. Tetapi kita dapat melihat bahwa Masukan pengguna tersebut mengandung katakunci yang sudah disimpan, yaitu “Siapa kamu?”. Seharusnya program dapat member respon yang sesuai dengan katakunci “Siapa kamu?”.

Oleh karena itu, algoritma pencocokan string Boyer-Moore dan KMP dapat digunakan untuk menyelesaikan masalah ini. Algoritma tersebut akan mencari katakunci yang merupakan bagian dari kalimat masukan pengguna.

Untuk mempermudah program mencari katakunci yang sesuai, pengembang dapat menerapkan hal-hal di bawah ini.

1. Katakunci yang hanya ditemukan di awal kalimat atau tengah kalimat dapat direpresentasikan menjadi _KATAKUNCI.
2. Katakunci yang hanya ditemukan di akhir atau tengah kalimat direpresentasikan dalam bentuk KATAKUNCI_.
3. Katakunci yang hanya ditemukan di tengah kalimat direpresentasikan dalam bentuk _KATAKUNCI_.
4. Katakunci yang bisa ditemukan di mana saja dalam kalimat, dapat direpresentasikan dalam KATAKUNCI.

IV. IMPLEMENTASI

A. Katakunci sebagai Pattern

Berikut merupakan sebagian katakunci dan respon yang saya gunakan.

Katakunci	Respon
"Siapa namamu"	“Namaku Sherbot. Keahlianku mengetahui hal yang orang tidak tau.” “Kurasa kamu pun tahu siapa aku.” “Hanya orang cupu yang tidak kenal aku.” “Beh, masa kamu tidak tau siapa aku sih.”
"Hai" "Halo"	"Halo juga" "Apa kabar"
"Aku mau"	"Kenapa kamu mau itu?" "Kamu beneran mau?" "Apalagi yang kamu mau?"
"Aku tidak suka"	"Kenapa kamu tidak suka?" "Sama" "Pasti ada alasan kenapa kamu tidak suka"
"Aku pikir"	"Jadi kamu berasumsi nih" "Jangan main asumsi dong" "Yakin?" "Bisa jadi" "Aku juga berpikir begitu"

Tabel 1 Katakunci dan respon

Katakunci-katakunci inilah yang digunakan sebagai pattern untuk dicocokkan pada masukan pengguna dengan menggunakan algoritma pencocokan string.

B. Pencocokan String yang Lebih Fleksibel

Berikut adalah pseudocode dari algoritma *Fuzzy String Search* yang digunakan.

```
Deklarasi
getLevenshteinDistance(string1, string2): fungsi untuk menghitung Levenshtein distance antara string1 dan string2.
knowledgebase[][][]: larik 3 dimensi untuk menyimpan katakunci dan respon.
distance[]: larik untuk menyimpan jarak string katakunci dengan input, pada indeks yang bersesuaian.
Min(): Mengeluarkan index elemen minimum
input: string masukan pengguna.

Algoritma
for setiap katakunci di knowledgebase do
    distance[] ←
       getLevenshteinDistance(katakunci, input)
idx ← min(distance[])
return katakunci di knowledgebase berdasarkan idx
```

C. Pengurutan Katakunci

Teknik ini diimplementasikan dengan cara memilih kata kunci yang paling panjang, jika ditemukan lebih dari satu kata kunci yang cocok.

Berikut pseudocodenya.

```
Deklarasi
findMatch(): mengeluarkan katakunci yang cocok berdasarkan algoritma pencocokan string.
matched[]: larik untuk menampung katakunci-katakunci yang cocok dengan input
input: string masukan pengguna

Algoritma
while katakunci belum habis do
      matched[] ← findMatch(input)
cari katakunci yang paling panjang di matched[]
return katakunci tersebut
```

D. Transposisi Katakunci

Berikut merupakan tabel yang berisi kata-kata yang akan dilakukan transposisi padanya.

aku	kamu
ku	mu
ayah	bapak
dia	beliau

Table 2 Transposisi kata

E. Lokasi Katakunci

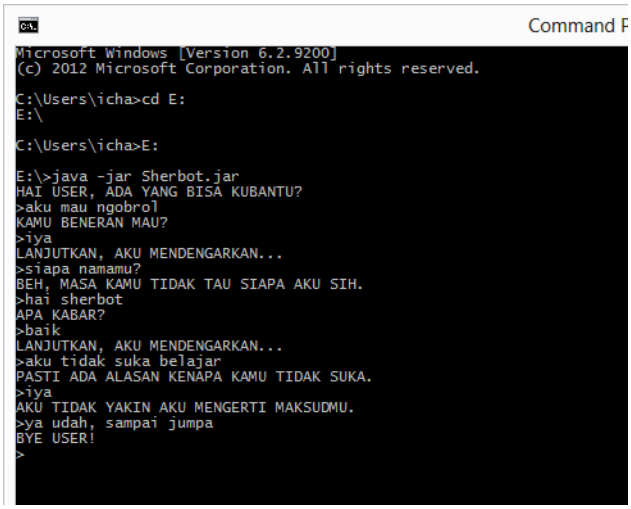
Berikut merupakan pseudocode algoritma Boyer-Moore dan KMP yang digunakan.

```
Boyer-Moore
Deklarasi
P: katakunci
T: teks masukan pengguna
last(): fungsi untuk menghitung indeks
kumunculan terakhir
Algoritma
Hitung fungsi last()
i ← m-1
j ← m-1
repeat
     if P[j] = T[i] then
         if j=0 then
             return i
         else
             i ← i-1
             j ← j-1
     else
         i ← i+m-Min(j, 1+last[T[i]])
         j ← m-1
until i>n-1
return -1
```

```
Knuth-Morris-Pratt
Deklarasi
P: katakunci
T: teks masukan pengguna
n: panjang T
m: panjang P
Algoritma
q ← 0
i ← 0
while (i < n) do
     if (P[q+1] = T[i+1]) then
        q ← q+1
        i ← i+1
        if (q = m) then
           return i-q
           geser P ke kanan
     else
         if (q=0) then i ← i+1
         else geser P ke kanan
```

F. Hasil Pengujian

Berikut adalah hasil tangkapan layar dari pengujian yang saya lakukan.



V. KESIMPULAN

Masih terdapat ketidaksesuaian pada program yang saya buat. Hal itu disebabkan karena kurangnya data percakapan yang saya simpan dalam program. Saran untuk pengembang, lebih banyak melakukan survey untuk mendapatkan percakapan-percakapan yang menarik. Semakin banyak data percakapan yang disimpan, akan semakin cerdas chatbot yang dibuat. Satu lagi saran untuk pengembang, tambahkan algoritma agar chatbot dapat belajar dari percakapan yang telah ia lakukan. Lebih baik simpan log percakapan yang dibuat, agar ketika chatbot tidak mengerti suatu percakapan, ia dapat melihat log percakapan yang bisa dijadikan refernsi untuk member respon.

VI. TERIMA KASIH

Penulis mengucapkan syukur dan terima kasih kepada Tuhan Yang Maha Esa atas selesainya tugas makalah ini. Penulis juga mengucapkan terima kasih kepada bapak Rinaldi Munir dan ibu Masayu selaku pengajar telah membagikan ilmu-ilmu yang beliau kuasai dengan ikhlas pada penulis dan memberikan kesempatan bagi penulis untuk menuangkan pikiran pada makalah ini. Tidak lupa, ucapan terima kasih juga diberikan kepada suami, orang tua, keluarga, dan teman-teman Informatika atas dukungan dan semangat yang diberikan selama menjalani hari-hari kuliah. Akhir kata, penulis berharap makalah ini dapat bermanfaat bagi pembaca.

REFERENSI

[1] Bruce Wilcox, "Beyond Façade: Pattern Matching for Natural Language Applications," Telltale Games, Feb 2011.
[2] http://www.codeproject.com/Articles/36106/Chatbot-Tutorial 09:00 AM 18/5/2014.
[3] http://www.chatbots.org/chatbot/ 09:00 AM 18/5/2014.
[4] http://ntz-develop.blogspot.com/2011/03/fuzzy-string-search.html 06:00 PM 18/5/2014.
[5] http://www.cs.uku.fi/~kilpelai/BSA05/lectures/slides03.pdf 10:00 AM 18/5/2014
[6] http://cs.indstate.edu/~kmandumula/presentation.pdf 09:30 AM 18/5/2014

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, 19 Mei 2014

Choirunnisa Fatima 13512084