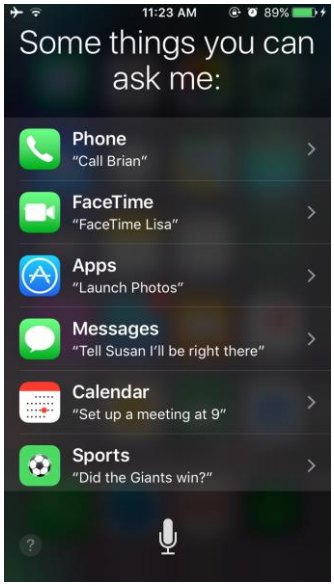
# Penerapan Algoritma *String Matching* dalam *Intelligent Personal Assistant Siri*

Adya Naufal Fikri - 13515130
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13515130@std.stei.itb.ac.id*

***Abstract*—Dewasa ini, teknologi semakin memudahkan kita dalam melakukan segala sesuatu. Salah satunya adalah dengan adanya *intelligent personal assistant* seperti Siri milik Apple. Dengan adanya Siri kita dapat mencari informasi, menjalankan suatu aplikasi, membuat suatu *reminder*, dan masih banyak yang lainnya dan semua itu cukup dilakukan dengan suara tanpa perlu mengetik sedikitpun. Dalam implementasinya Siri menggunakan beberapa algoritma, termasuk salah satunya adalah *string matching*. *String matching* digunakan untuk mencocokan command mana yang dimaksud oleh pengguna, berdasarkan teks yang didapat dari menerjemahkan suara pengguna ke dalam teks.**

***Keywords— Siri; String Matching; Knuth-Morris-Path; Boyer-Moore; Intelligent Personal Assistant;***

## I. PENDAHULUAN




Gambar 1. Beberapa *command* pada Siri.
Sumber : Dokumentasi pribadi

Perkembangan zaman sekarang ini sedang pesat-pesatnya. Teknologi robotika dan kecerdasan buatan saat ini sedang merajalela. Menjadi suatu topik yang paling menarik untuk diteliti dan dikembangkan, dan penerapannya mendapat sambutan hangat dari semua orang. Salah satunya adalah asisten pribadi cerdas (Intelligent Personal Assistant) seperti Siri milik Apple, Google Assistant milik Google, dan Cortana milik Windows. Program tersebut memanfaatkan kecerdasan buatan yang mengimplementasikan machine learning untuk membuat sebuah personal assistant yang cerdas, yang bisa mengerti permintaan dari pengguna, yang tidak menggunakan bahasa program, hanya menggunakan bahasa sehari-hari saja. Tentu hal ini tidak terlepas juga dengan andil dari algortima-algortima yang sederhana namun menarik, seperti string matching. Penggunaan string matching ini sangat *powerful*, bisa digunakan untuk melakukan pencarian di web, misalnya pada google, dan juga dalam Siri string matching digunakan untuk mencari command apa yang dimaksud oleh pengguna, sehingga Siri bisa menjalankan perintah pengguna itu secara presisi.

## II. DASAR TEORI

*String matching* atau pencocokan string merupakan salah satu metode pencarian yang paling populer. Metode ini menggunakan dua hal sebagai parameter, pertama adalah teks yang menjadi tempat pencarian, dan yang kedua adalah pola yang dicari pada teks. *String matching* melakukan pencarian pola pada teks, dan mengembalikan *true* atau suatu bilangan integer tertentu jika pola ditemukan dalam teks. Dalam keberjalanannya terdapat beberapa algoritma yang dapat digunakan untuk mengimplementasikan metode *string matching* ini. Berikut akan dibahas dua algoritma yang populer digunakan untuk mengimplementasikan string matching.

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

Algoritma *knuth-morris-pratt* atau yang akrab disebut dengan algoritma KMP merupakan algoritma *string matching* yang melakukan pengecekan dari kiri ke kanan, sama seperti brute force. Tetapi algoritma KMP melakukan penggeseran pengecekan string yang jauh lebih efisien dibandingkan dengan brute force. Algoritma KMP tidak melakukan pengecekan dengan menggeser posisi secara satu per satu, tetapi menggunakan prefix terbesar dari pola yang juga merupakan suffix dari pola tersebut sebagai acuan untuk mengetahui jarak paling jauh untuk menggeser pengecekan pola.

Pada dasarnya algoritma KMP berusaha untuk membuang pengecekan yang ‘percuma’ dan bisa dipastikan salah. Sehingga pengecekan seperti itu tidak perlu dilakukan dan dilewati saja sehingga proses pencocokan string dapat berjalan lebih cepat. Berikut merupakan ilustrasi dari penerapan algoritma KMP.

Tahap 1 :

Teks

| A (posisi) | B | X | A | B | X | A |
|---|---|---|---|---|---|---|

Pola

| A (posisi) | B | X | A | B | A |
|---|---|---|---|---|---|

CEK : Apakah A & A sama? → Sama, lanjut posisi selanjutnya.

Tahap 2 :

Teks

| A | B | X | A | B | X (posisi) | A |
|---|---|---|---|---|---|---|

Pola

| A | B | X | A | B | A (posisi) |
|---|---|---|---|---|---|

CEK : Apakah X & A sama? → Berbeda, maka lakukan penggeseran berdasarkan algoritma KMP. Yaitu panjang pola yang sudah match - panjang prefix Pola[0..panjang pola match] yang juga suffix dari Pola[1..panjang pola match]. Dalam kasus ini :

Panjang pola match "ABXAB" = 5

Panjang prefix "ABXAB" yang juga suffix "BXAB" = "AB" = 2

Maka penggeseran dilakukan sebanyak = 5 - 2 = 3

Tahap 3 :

Teks

| ..(skip) | A | B | X (posisi) | A | B | A |
|---|---|---|---|---|---|---|

Pola

| A | B | X (posisi) | A | B | A |
|---|---|---|---|---|---|

CEK : Apakah X & X sama? → Sama, lanjut ke posisi selanjutnya. Karena sampai karakter terakhir pada pola cocok dengan teks di atas, maka akan mengembalikan *true* sebagai tanda bahwa polanya terdapat pada teks.

Proses penggeseran pada algoritma KMP ini disebut dengan fungsi pinggiran. Dimana fungsi pinggiran ini mengimplementasikan perhitungan pergeseran yang harus dilakukan pola seperti yang dilakukan pada tahap 2. Inilah yang membuat algoritma KMP menjadi algoritma pencocokan string yang cepat dan efisien dikarenakan proses pergeseran yang efektif dan tidak pernah melakukan backtrack ketika pencarian. Tetapi meskipun begitu algoritma KMP juga memiliki beberapa kekurangan. Diantaranya adalah ketika variasi alfabet pada teks sangat banyak dan memungkinkan terjadinya banyak mismatch atau ketidaksesuaian pada pengecekan, terutama ketika mismatch terjadi pada bagian awal pola yang menyebabkan penggunaan fungsi pinggiran menjadi tidak terlalu memberikan pengaruh besar pada peningkatan performansi.

Kompleksitas waktu dari algoritma KMP dapat dilihat sebagai berikut:

Penggunaan Fungsi Pinggiran : O(m);

Pencarian String : O(n);

Kompleksitas waktu algoritma KMP = O(m+n)

### *B. Algoritma Boyer-Moore*

Algoritma boyer-moore dapat dikatakan sebagai algoritma yang memiliki pendekatan yang berbeda dengan dua algoritma sebelumnya. Algoritma boyer-moore melakukan pencarian secara mundur (dari kanan ke kiri) yang disebut dengan looking glass technique. Selain itu dalam rangka membuang proses pengecekan yang tidak diperlukan, algoritma boyer-moore menggunakan character jump technique. Teknik ini digunakan berdasarkan kemunculan karakter teks yang salah ketika dilakukan pengecekan dengan salah satu huruf pada pola untuk kemudian menyesuaikan posisi pola agar sesuai dengan karakter tersebut.

Pada dasarnya algoritma boyer-moore ini merupakan kebalikan dari algoritma brute force yang dilakukan beberapa perbaikan dalam proses pergerseran pola. Dengan adanya penggunaan dua teknik khusus pada boyer-moore membuat algoritma ini cukup efisien dan populer digunakan terutama dalam proses pencocokan string pada website. Berikut merupakan ilustrasi dari pencocokan string dengan algoritma boyer-moore.

Tahap 1 :

Teks

| A | B | A | A | B | X | B (posisi) |
|---|---|---|---|---|---|---|

Pola

| ? | ? | ? | A | X | B (posisi) |
|---|---|---|---|---|---|

CEK : Apakah A & A sama? → Sama, lanjut posisi mundur selanjutnya. Disini terlihat pengecekan akan salah pada B & A.

Tahap 2 :

Teks

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | A | A | B (posisi) | X | B |

Pola

| | | | | | |
|---|---|---|---|---|---|
| ? | ? | ? | A (posisi) | X | B |

CEK : Apakah B & A sama? → Berbeda, maka selanjutnya terdapat 3 kemungkinan untuk tahap selanjutnya.

Tahap 3 (Kasus 1) :

Teks

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | A | A | B (posisi) | X | B |

Pola

| | | | | | |
|---|---|---|---|---|---|
| A | B | C | A (posisi) | X | B |

KASUS 1 : Pola memiliki karakter ‘B’ di sebelah kiri posisi pengecekan saat ini, maka kemudian akan dilakukan teknik character-jump untuk menyamakan ‘B’ pada pola dengan ‘B’ pada teks. Kemudian ulangi pengecekan dari belakang kembali.

Teks

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | X | B | ? | ? (posisi) | ? |

Pola

| | | | | | |
|---|---|---|---|---|---|
| A | B | C | A | X | B (posisi) |

Tahap 3 (Kasus 2) :

Teks

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | A | A | B (posisi) | X | B |

Pola

| | | | | | |
|---|---|---|---|---|---|
| A | X | C | A (posisi) | X | B |

KASUS 2 : Pola memiliki karakter ‘B’ di sebelah kanan posisi pengecekan saat ini dan tidak ada ‘B’ di sebelak kiri posisi pengecekan, maka kemudian karena tidak memungkinkan teknik melakukan character jump dengan melakukan pergeseran ke kanan, maka pola digeser ke kanan sejauh 1 huruf saja. Kemudian ulangi pengecekan dari belakang kembali.

Teks

| | | | | | | |
|---|---|---|---|---|---|---|
| A | A | B | X | B | ? (posisi) | ? |

Pola

| | | | | | |
|---|---|---|---|---|---|
| A | X | C | A | X | B (posisi) |

Tahap 3 (Kasus 3) :

Teks

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | A | A | B (posisi) | X | A |

Pola

| | | | | | |
|---|---|---|---|---|---|
| A | X | C | A (posisi) | X | A |

KASUS 3 : Pola tidak memiliki karakter ‘B’ baik di sebelah kiri maupun kanan dari posisi pengecekan saat ini, maka kemudian karena tidak memungkinkan teknik melakukan character jump, maka pola digeser ke kanan hingga posisi pola paling kiri. Kemudian ulangi pengecekan dari belakang kembali.

Teks

| | | | | | | |
|---|---|---|---|---|---|---|
| A | ? | ? | ? | ? | ? (posisi) | ? |

Pola

| | | | | | |
|---|---|---|---|---|---|
| A | X | C | A | X | A (posisi) |

Seperti yang telah disebutkan sebelumnya, algoritma boyer-moore merupakan kebalikan daripada algoritma brute force. Algoritma boyer-moore memiliki keunggulan ketika teks yang digunakan memiliki variasi alfabet yang sangat tinggi, berbeda dengan algoritma brute force dan KMP dimana hal ini menjadi kelemahan mereka dikarenakan melakukan pengecekan dari sebelah kiri ke kanan. Tetapi kebalikannya, untuk teks yang memiliki tingkat variasi alfabet sangat rendah, algoritma boyer-

moore sangat tidak cocok untuk digunakan karena memiliki performansi yang sangat buruk pada teks tersebut.

Kompleksitas algoritma boyer-moore dapat dilihat sebagai berikut :

Kompleksitas algoritma boyer-moore : O(mn)

Pada kasus terburuk : O(mn + A)

dengan A adalah variasi dari alfabet.

## III. PENERAPAN ALGORITMA STRING MATCHING PADA SIRI

Dalam penerapannya Siri melakukan beberapa tahap terlebih dahulu sebelum sampai pada tahap string matching. Tahap yang pertama adalah voice recognition. Pengguna memberikan inputan kepada Siri berupa suara yang kemudian akan dikirimkan ke server Apple melalui internet untuk sampai ke tahap selanjutnya. Setelah itu, tahap selanjutnya terjadi di dalam server Apple, yaitu mengubah inputan suara tersebut menjadi kata-kata atau sering disebut speech-to-text. Proses ini dilangsungkan di dalam server Apple dikarenakan hasilnya akan didapat lebih cepat dibandingkan diproses di dalam device Apple itu sendiri. Setelah teks dari inputan suara itu diperoleh, lalu akan dikirimkan lagi ke device Apple tersebut, untuk dijalankan perintahnya oleh Siri.

Lalu proses yang selanjutnya adalah proses *string matching*. Terdapat dua algoritma yang sangat popular untuk melakukan hal ini, diantaranya adalah algoritma Knuth-Morris-Pratt dan algoritma Boyer-Moore. Keduanya memiliki kelebihannya masing-masing, dan dalam makalah ini akan dibandingkan kecepatan dari kedua algoritma ini untuk sebuah command umum pada Siri.

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

Dalam implementasi algoritma KMP ini, digunakan dua buah fungsi, yang utama yaitu fungsi KMPMatch yang berguna untuk mencocokan pola dan teks. Dan yang kedua adalah fungsi computeFail yang merupakan algoritma untuk menghitung fungsi pinggiran, yang akan membantu dalam menentukan perpindahan posisi teks dan pola. Di bawah ini, akan ditampilkan implementasi dari fungsi KMPMatch dan computeFail dalam Bahasa pemrograman C#.

### *B. Algoritma Boyer-Moore*

Dalam implementasi algoritma Boyer-Moore ini, digunakan dua fungsi utama juga, yaitu fungsi BMMatch yang berguna untuk mencocokan pola dengan teks. Dan yang kedua fungsi buildLast yang merupakan algoritma untuk mencari kemunculan terakhir dari sebuah karakter, ini akan membantu dalam proses jump character pada algoritma Boyer-Moore. Di bawah ini, akan ditampilkan implementasi dari fungsi BMMatch dan buildLast dalam Bahasa pemrograman C#.

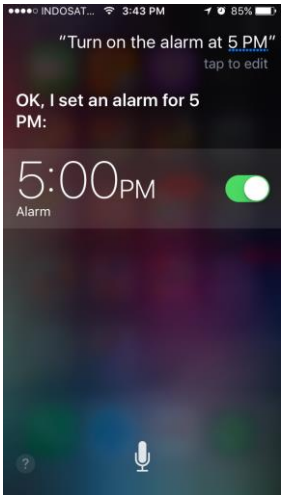
```
public static int[] computeFail(string pattern)
{
    int[] fail = new int[pattern.Length];
    fail[0] = 0;

    int m = pattern.Length;
    int j = 0;
    int i = 1;
    while (i < m)
    {
        if (pattern[j] == pattern[i])
        {
            fail[i] = j + 1;
            i++;
            j++;
        }
        else if (j > 0) { j = fail[j - 1]; }
        else { fail[i] = 0; i++; }
    }
    return fail;
}
```

```
public static int KMPMatch(string text, string
pattern)
{
    int n = text.Length;
    int m = pattern.Length;
    int[] fail = computeFail(pattern);

    int i = 0, j = 0;

    while (i < n)
    {
        if (pattern[j] == text[i])
        {
            if (j == m - 1) { return i - m + 1; }
            i++;
            j++;
        }
        else if (j > 0) { j = fail[j - 1]; }
        else { i++; }
    }
    return -1;
}
```

```
public static int[] buildLast(string pattern)

{

    int[] last = new int[256];

    for (int i = 0; i < 256; i++) { last[i] = -1; }

    for (int i = 0; i < pattern.Length; i++) {

        last[pattern[i]] = i;

    }

    return last;

}
```

```
public static int BMMatch(string text, string pattern)
{
    int[] last = buildLast(pattern);
    int n = text.Length;
    int m = pattern.Length;
    int skip;
    for (int i = 0; i <= n - m; i += skip)
    {
        skip = 0;
        for (int j = m - 1; j >= 0; j--)
        {
            if (pattern[j] != text[i + j])
            {
                try
                {
                    skip = Math.Max(1, j - last[text[i + j]]);
                    break;
                }
                catch (Exception exp)
                {
                    skip = 1;
                    break;
                }
            }
        }
        if (skip == 0) return i;
    }
    return -1;
}
```

Perbandingan akan dilakukan terhadap command “Turn on the alarm at 5 pm.”

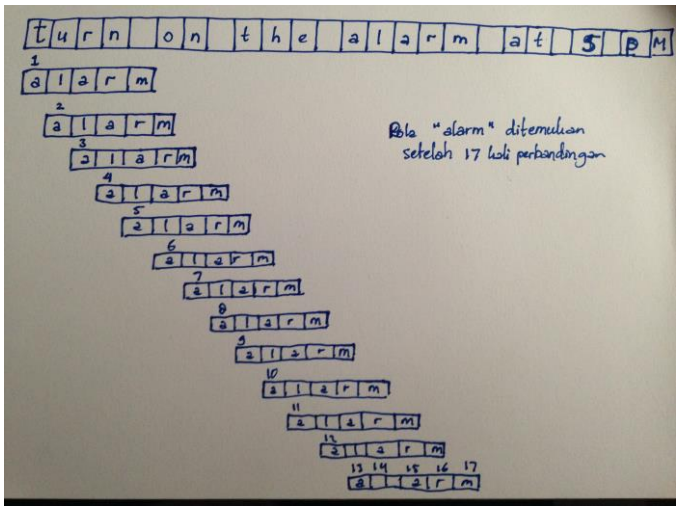



Gambar 2. *Command alarm* pada Siri.
Sumber : Dokumentasi pribadi

1. Algoritma Knuth-Morris-Pratt

| j | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| P[j] | a | l | a | r | m |
| Fail[j] | 0 | 0 | 0 | 1 | 0 |

Tabel 1. Fungsi Pinggiran pada algoritma KMP



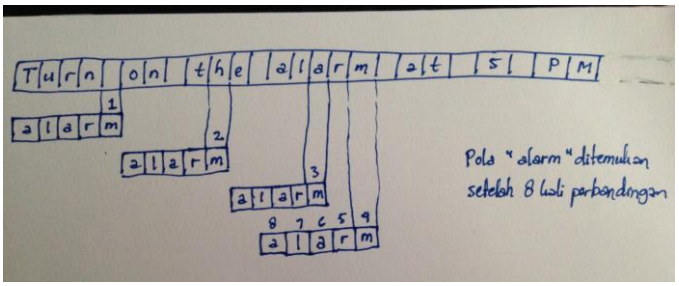

Gambar 3. Hasil pencocokan menggunakan algoritma KMP.

Dari gambar di atas dapat dilihat, bahwa untuk mencocokan string “Turn on the alarm at 5 PM” dengan pola alarm dibutuhkan perbandingan **sebanyak 17 kali** untuk algoritma KMP.

2. Algoritma Boyer-Moore

| c | a | l | m | r |
|---|---|---|---|---|
| L(c) | 2 | 1 | 4 | 3 |

Tabel 2. Fungsi kemunculan terakhir (buildLast) pada algoritma Boyer-Moore




Gambar 4. Hasil pencocokan menggunakan algoritma Boyer-Moore

Dari gambar di atas dapat dilihat, bahwa untuk mencocokan string “Turn on the alarm at 5 PM” dengan pola alarm dibutuhkan perbandingan **sebanyak 8 kali** untuk algoritma Boyer-Moore.

Setelah ditemukan kecocokan antara teks dengan pola command alarm, maka selanjutnya dicari apakah menyalakan atau mematikan alarm, yang berdasarkan percobaan tadi digunakan perintah turn on, maka akan dinyalakan sebuah alarm. Dan dibutuhkan suatu waktu yang pada percobaan sebelumnya ditentukan 5 PM. Maka perintah “Turn on the alarm

at 5 PM" akan membuat Siri secara otomatis mengaktifkan alarm pada pukul 5 sore.

## IV. KESIMPULAN

String matching merupakan suatu algoritma yang sangat berguna untuk memudahkan pekerjaan manusia. Salah satu implementasinya adalah pada *Intelligent Personal Assistant Siri*. Dengan menggunakan string matching ini, Siri bisa mengetahui perintah apa yang dimaksudkan oleh pengguna, sehingga Siri bisa menjalankan perintah pengguna secara presisi dan cepat.

## REFERENSI


[1] Levitin, Anany. 2011. Introduction to the Design and Analysis of Algorithms. Pearson : New York.

[2] Munir, Rinaldi. 2009. Diktat Kuliah IF2211 Strategi Algoritma. Institut Teknologi Bandung : Bandung.

[3] How Siri Works | HowStuffWorks. http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/siri.htm Diakses pada tanggal 18 Mei 2017 pukul 22.01.


## 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 2017

Adya Naufal Fikri
13515130