Penerapan dan Perbandingan Algoritma String Matching pada Aplikasi UUD 1945 dan UU di Indonesia

Christopher Billy Setiawan / 13517050
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517050@std.stei.itb.ac.id

***Abstract*— Undang - undang 1945 adalah hukum dasar konstitusi pemerintahan negara Indonesia. UUD 1945 ini diturunkan dari Pancasila yang merupakan Ideologi dasar dari negara Indonesia. Dari Undang - undang 1945, diturunkan segala undang – undang yang mengatur segala perwenangan dalam Indonesia. String Matching adalah algoritma yang dikhususkan untuk mencocokkan string pattern dengan suatu teks, algoritma *string matching* yang akan dibahas ada tiga jenis, yaitu algoritma Brute Force, *Knuth-Morris-Pratt,* dan *Boyer-Moore,* beserta *Regular Expression* sebagai perbandingan.**

Keywords—UUD 1945, UU, KMP, BM, Brute Force, Regex, String Matching.

I. PENDAHULUAN

Pada zaman ini, pemrograman merupakan hal yang sudah biasa. Apalagi untuk setiap mahasiswa Teknik Informatika, setiap mahasiswa Teknik Informatika harus sudah terlatih dalam pengembangan *software* di Dunia yang sudah memasuki zaman revolusi industri ke 4 dimana teknologi sudah jauh berkembang daripada tahun-tahun silam. Akses menuju Internet sudah menjadi sangat mudah. Dulu, UUD 1945 dan undang – undang Indonesia dicetak secara fisik menjadi buku UUD 1945 dan buku undang - undang. Sekarang, semua hal yang konvensional itu sudah berubah. Dari yang dulu dicetak secara buku fisik, sekarang menjadi Aplikasi android yang dapat diakses oleh semua orang melalui *smartphone* mereka.

Salah satu fitur yang ditambahkan ke dalam *software* tersebut adalah fitur *searching* yang digunakan untuk mencari *keyword* dari pasal tertentu. Hal ini merupakan sesuatu yang sangat membantu dalam dunia hukum di Indonesia. Tentunya dalam membuat fitur *searching* tersebut bukanlah sesuatu yang mudah. Ada *Programmer* yang tidak tidur selama beberapa hari untuk membuat fitur tersebut. Algoritma yang digunakan untuk membuat fitur *search* tersebut adalah algoritma *String Matching*. Fitur *searching* pada aplikasi ini dapat membantu mempersingkat pencarian pasal dalam undang – undang, sehingga orang awam pun dapat mengakses pasal-pasal dalam UUD 1945 dan perundang – undangan Indonesia dengan cepat.

String Matching atau biasa sering disebut algoritma pencocokan string yaitu algoritma untuk melakukan pencarian semua kemunculan string pendek dan panjang, untuk string pendek yang disebut pattern dan string yang lebih panjang disebut teks. Algoritma *String Matching* ini dapat diklasifikasikan menjadi banyak jenis, tetapi pada makalah ini hanya akan dibahas tiga saja yaitu Algoritma *Brute Force*, Algoritma Knuth-Morris-Pratt (KMP), dan Algoritma Boyer-Moore. Setelah itu algoritma tersebut akan dibandingkan dengan menggunakan *Regular Expression*.

Brute force adalah salah satu algoritma *string matching* yang memiliki strategi yang *straight forward* atau lempang dalam mendapatkan solusinya.

Knuth-Morris-Pratt atau KMP adalah algoritma *string matching* yang dibuat oleh Donald E. Knuth, James H. Morris, dan Vaughan R. Pratt. Algoritma ini memiliki kecepatan yang lebih baik dibandingkan dengan algoritma brute force.

Boyer-Moore adalah algoritma *string matching* yang dibuat oleh Robert S. Boyer dan J. Strother Moore. Algoritma ini memiliki kecepatan yang lebih baik dibandingkan dengan algoritma brute force.

Regular Expression adalah konstruksi Bahasa untuk mencocokkan teks berdasarkan pola tertentu, terutama untuk kasus-kasus kompleks. Regex mengandalkan pola atau *pattern* dalam pencariannya, dan proses pencariannya berdasarkan kata dan bukan seluruh string. Oleh karena itu Regex berbeda dengan algoritma *exact matching*. Dimana algoritma *exact matching* harus *exact match* berdasarkan string, sedangkan Regex tidak (*heuristic match*).

Makalah ini akan membahas tentang perbandingan penggunaan ketiga algoritma *String Matching* dengan *Regular Expression*.

II. LANDASAN TEORI

2.1 String Matching

String Matching adalah proses pencarian semua kemunculan *query* yang selanjutnya disebut pattern ke dalam *string* yang lebih Panjang (teks). *Pattern* dilambangkan dengan x = x[0..m-1] dan panjang adalah m. Teks dilambangkan dengan y = y[0..n-1] dan panjangnya adalah n. *String matching* dibagi menjadi dua yaitu *exact matching* dan *heuristic matching.*

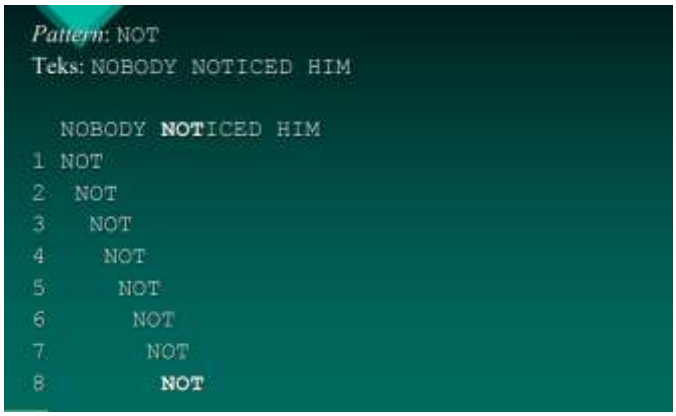
2.1.1 Exact Matching

Exact matching digunakan untuk menemukan *pattern* persis dari satu teks. Dimana *exact match* membandingkan satu string *pattern* secara keseluruhan tanpa mempedulikan kata di dalam *string pattern* tersebut. Contoh pencarian *exact matching* adalah pencarian sebuah *pattern* "Saya pintar" dalam teks "Saya pintar tapi suka merendah" dan "Saya orang pintar". Sistem akan memberikan hasil bahwa teks pertama mengandung *pattern* "Saya pintar" dan teks kedua tidak.

Algoritma *exact matching* yang akan dibahas dalam makalah ini ada tiga jenis, yaitu algoritma brute force, Knuth-Morris-Pratt, dan Boyer-Moore.

2.1.1.1 Algoritma Brute Force

Algoritma *brute force* dalam *string matching* adalah algoritma untuk pencocokkan *string* dengan menggunakan cara yang lempang atau *straight forward*. Dimana *pattern* akan dicari secara satu per satu dalam suatu teks dari kiri atau dari kanan sampai akhir dari *string* tersebut. Jika ada salah satu karakter dari *pattern* yang tidak *match* dengan teks, maka iterator akan bergerak dan mengecek ulang dari awal *pattern* tersebut. Berikut adalah ilustrasi gambar pencocokan string.



Gambar 1 Ilustrasi pencocokkan string

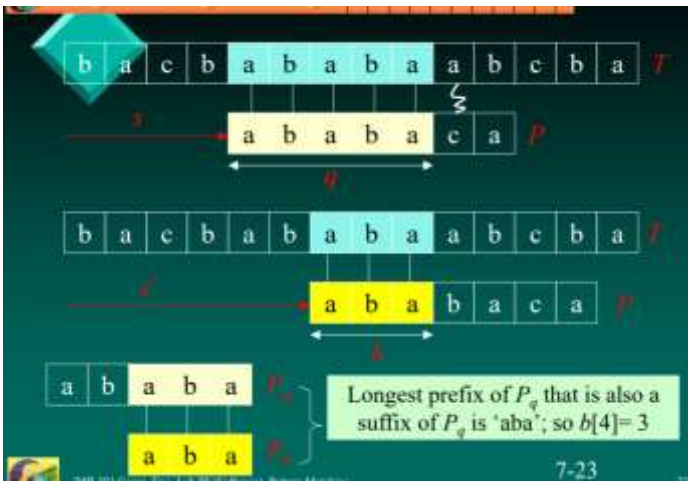
Berikut adalah kode program untuk algoritma *brute force*.

```
def brute(text, pattern):
    textLength = len(text)
    patternLength = len(pattern)

    for i in range(textLength-patternLength+1):
        j = 0
        while((j < patternLength) && (text[i+j] == pattern[j])):
            j+=1;

        if (j == patternLength):
            return i; # match at i

    return -1 # no match
```

Gambar 2 Contoh Kode program brute force

2.1.1.2 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencocokkan *string* yang dibuat oleh Donald E. Knuth, James H. Morris, dan Vaughan R. Pratt. Algoritma ini memiliki kecepatan yang lebih baik dibandingkan dengan algoritma brute force. Algoritma ini lebih efisien daripada brute force dikarenakan oleh metode pencariannya yang berbeda dan lebih pintar dibandingkan dengan *brute force*. KMP mengharuskan program pengecekannya dimulai dari kiri ke kanan. Pertama ia akan melakukan observasi awal atau *preprocessing* terlebih dahulu dengan cara mengecek ulang kata sebelumnya. Setelah itu ia akan memulai proses pencariannya. Ketika system menemukan *mismatch* pada pengecekkan, system akan mencocokkan prefix terpanjang dari *pattern* dan suffix terpanjang dari karakter – karakter teks yang telah di cek. Lalu menggeser iterator sesuai dengan kecocokkan prefix dan suffix tersebut.
Berikut adalah gambar ilustrasi pencocokkan string dalam KMP.



Gambar 3 Ilustrasi pencocokkan string KMP

```
def KMP(text,pattern):
    textLength = len(text)
    patternLength = len(pattern)
    fail = computeFail(pattern) #Kmp match
    i = 0
    j = 0
    if(textLength < patternLength):
        return 0
    while(i < textLength):
        if(pattern[j] == text[i]):
            if(j == patternLength -1):
                return i - textLength +1
            i += 1
            j += 1
        elif(j > 0):
            j = fail[j-1]
        else:
            i += 1
    return -1
```

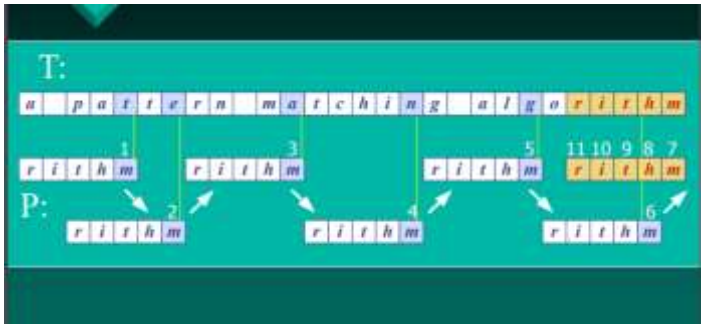
Gambar 4 Contoh Kode program KMP

2.1.1.3 Algoritma Boyer-Moore

Boyer-Moore adalah algoritma *string matching* yang dibuat oleh Robert S. Boyer dan J. Strother Moore. Algoritma ini memiliki kecepatan yang lebih baik dibandingkan dengan algoritma brute force dikarenakan metodenya yang berbeda dan lebih pintar. Jika dibandingkan dengan algoritma KMP, algoritma Boyer-Moore ini tidak jauh berbeda karena sifatnya yang sama. Mungkin perbedaannya hanya pada bagian implementasinya yang memulai pencarian dari kanan ke kiri.

Dalam system Boyer-Moore, pertama penjajaran pattern akan disamakan dengan teks dari kiri. Tetapi pengecekan karakter akan dimulai dari kanan. System akan mengecek dari karakter paling kanan sampai kiri, jika ada yang tidak cocok, system memiliki tiga pilihan dalam pengambilan keputusan. Pertama yaitu jika karakter salah yang ada di teks berada pada posisi kiri *pattern*, maka karakter dalam *pattern* tersebut akan dijajarkan dengan karakter yang sama di dalam teks dan program akan mengecek dari kanan lagi. Kedua, jika karakter salah yang ada di teks berada pada posisi kanan *pattern* maka *pattern* akan di geser ke kanan sebanyak satu karakter dan program akan kembali masuk ke iterasi selanjutnya. Terakhir yang ketiga, jika karakter salah yang ada di teks tidak ada di dalam *pattern* sama sekali, maka *pattern* akan di geser ke kanan sampai melewati karakter tersebut. System akan diiterasi sampai akhirnya dapat *exact match* atau sudah sampai akhir teks dan masih tidak mendapatkan *match*.

Berikut adalah ilustrasi gambar proses pencarian Boyer-Moore



Gambar 5 ilustrasi proses pencarian Boyer-Moore

Berikut adalah contoh kode program dari algoritma Boyer-Moore.

```
def BM(text,pattern):
    last = buildLast(pattern)#Last occurence
    textLength = len(text)
    patternLength = len(pattern)
    i = patternLength-1
    if(i > textLength-1):
        return 0
    j = patternLength-1
    while (i <= textLength - 1):
        if(pattern[j] == text[i]):
            if(j == 0):
                return i
            else:
                i -= 1
                j -= 1
        else:
            lo = last[ord(text[i])]
            i = i + patternLength - min(j,1+lo)
            j = patternLength - 1

    return -1
```

BuildLast merupakan fungsi tambahan dari algoritma boyer-moore yang dilakukan sebelom proses pencocokkan string dimulai (*pre-process)* untuk mencari indeks kemunculan terakhir dari setiap karakter yang ada di dalam suatu *pattern*.

2.1.2 Heuristic Matching

Heuristic matching adalah teknik yang digunakan untuk menghubungkan dua data terpisah ketika *exact matching* tidak mampu mengatasi karena ada pembatasan pada data yang tersedia. Heuristic matching dapat dilakukan dengan perhitungan distance antara pattern dengan teks. Salah satu contoh dari *Heuristic matching* adalah regular expression.

2.1.2.1 Regular Expression

Regular Expression atau yang biasa di sebut juga Regex/Regexp adalah konstruksi Bahasa untuk mencocokkan teks berdasarkan pola tertentu, terutama untuk kasus-kasus kompleks. Regex mengandalkan pola atau *pattern* dalam pencariannya, dan proses pencariannya berdasarkan kata dan bukan seluruh string. Oleh karena itu Regex berbeda dengan algoritma *exact matching*. Dimana algoritma *exact matching* harus *exact match* berdasarkan string, sedangkan Regex tidak (*heuristic match*). Penggunaan regex jauh lebih universal dan lebih mudah dibandingkan dengan algoritma *string matching* lainnya. Karena regex hanya mencari pola dari sebuah kata dan dapat mendeteksi kata yang terpisah di dalam teks sehingga tidak harus persis sama dengan query kalimat string.

Simbol	Keterangan
.	Any character except newline
\.	A period (and so on for *, \(, \\, etc.)
^	The start of the string
$	The end of the string
\d,\w,\s	A digit, word character [A-Za-z0-9], or whitespace
\D,\W,\S	Anything except a digit, word character, or whitespace
[abc]	Character a, b, or c
[a-z]	a through z
[^abc]	Any character except a, b, or c
aa|bb	Either aa or bb
?	Zero or one of the preceding element

*****	Zero or more of the preceding element.
+	One or more of the preceding element
{n}	Exactly n of the preceding element.
{n, }	N or more of the preceding element.
{m,n}	Between m and n of the preceding element
??,*?,+?,{n}?,etc.	Same as above, but as few as possible
(expr)	Capture expr for use with \1, etc.
(?:expr)	Non-Capturing group.
(?=expr)	Followed by expr
(?!expr)	Not followed by expr

Tabel 1 Referensi Regex secara umum

2.2 Undang - Undang Dasar 1945

Konstitusi atau Undang-undang Dasar (bahasa Latin: constitutio) dalam negara adalah sebuah norma sistem politik dan hukum pemerintahan yang dituangkan dalam dokumen tertulis. Konstitusi memuat aturan dan prinsip-prinsip entitas politik dan hukum termasuk dalam bentukan struktur, prosedur, wewenang dan kewajiban pemerintahan negara pada umumnya. Konstitusi umumnya merujuk pada penjaminan hak kepada warga masyarakatnya.

Sebagai hukum dasar, UUD 1945 merupakan sumber hokum tertulis. Dengan demikian setiap produk hokum seperti undang-undang, oeraturan pemerintahan, peraturan presiden, ataupn bahkan setiap tindakan atau kebijakan pemerintah haruslah berlandaskan dan bersumber pada peraturan yang lebih tinggi, yang pada akhirnya semua peraturan perundang-undangan tersebut harus dapat dipertanggungjawabkan dengan ketentuan UUD 1945, dan muaranya adalah Pancasila sebagai sumber dari segala sumber hukum negara.

2.3 Undang – undang Indonesia

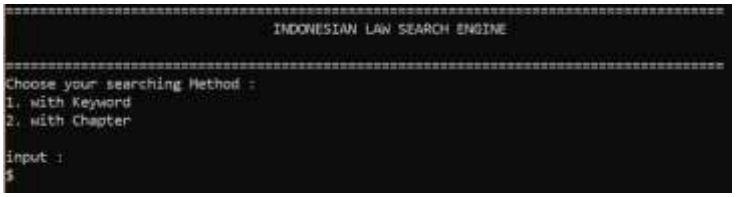
Undang – undang adalah peraturan perundang-undangan yang dibentuk oleh Dewan Perwakilan Rakyat dengan persetujuan Bersama Presiden. Undang – undang memiliki kedudukan sebagai aturan main bagi rakyat untuk berbangsa dan bernegara, untuk mengatur kehidupan antarsesama dalam rangka mewujudkan tujuan dalam bentuk negara. Banyak Undang-undang juga dapat dikatakan sebagai kumpulan prinsip yang mengatur prinsip kekuasaan pemerintah dan hak rakyat dalam berbangsa dan bernegara. Undang-undang yang ada di Indonesia semua bergantung dan berakar pada Undang-undang Dasar 1945.

III. IMPLEMENTASI DAN PENJELASAN ALGORITMA PROGRAM

Secara garis besar, program akan bekerja sebagai berikut:

1. Program akan meminta input untuk memilih metode pencarian pasal.
2. Jika metode yang dipilih adalah dengan menggunakan *keyword* dari isi pasal, maka program akan meminta input lagi untuk memilih algoritma pencarian yaitu Brute Force, KMP, BM, atau Regex. Namun jika metode yang dipilih adalah dengan menggunakan *Chapter* (Pasal), maka program akan langsung mengambil isi dari Pasal yang diminta.
3. Setelah itu, untuk kasus pertama, program akan menjalankan algoritma pencocokkan *String* yang sesuai dengan ketentuan pengguna dan akan mencetak Pasal-pasal yang mengandung kalimat tersebut. Untuk kasus kedua, program akan langsung mengambil *value* dari *key* yang dimasukkan oleh pengguna.

Berikut contoh gambar keluaran program utama.



Gambar 6 Program utama

Dalam kasus pertama, ketika pengguna memilih untuk mencari dengan menggunakan kata kunci, program akan meminta memasukkan inputan kedua untuk mendefinisikan algoritma apa yang mau dipakai oleh user dalam pencarian. Setelah itu, program akan menjalankan fungsi untuk mendapatkan semua pasal yang bersangkutan dengan kata kunci tersebut. Cara program mendapatkan semua pasal yaitu dengan cara mengakses *value* dari *key* lalu melemparkan *value string* tersebut ke dalam fungsi pencocokkan *string* sesuai dengan algoritma pilihan pengguna. Lalu hasil dari pencarian itu akan dikembalikan ke main program lalu dicetak ke command prompt menampilkan seluruh pasal yang bersangkutan dengan kata kunci tersebut.

Dalam kasus kedua, ketika pengguna memilih untuk mencari dengan menggunakan pasal secara langsung, program akan menjalankan fungsi untuk mengambil *value* dari pasal yang bersangkutan. *Value* atau isi dari pasal tersebut akan diambil dari *dictionary* dan dikembalikan ke main program lalu dicetak untuk pengguna.

Dikarenakan keterbatasan waktu, database yang digunakan untuk menguji program ini belum lengkap dan hanya sebagian saja.

3.1 Uji Kasus

3.1.1 Uji Kasus Brute Force

Ketika pengguna memasukkan kombinasi kasus pertama untuk menggunakan metode dengan *matching keyword* dan

memasukkan metode algoritma Brute Force, hal berikut yang akan terjadi.

1. Brute Force akan berjalan dari kiri ke kanan untuk mencocokkan *string* satu demi satu. Jika ada satu karakter saja yang tidak memenuhi kecocokkan, maka iterator pada teks akan digeser sebanyak satu karakter dan proses pencocokkan *string* akan dilakukan ulang dari kiri ke kanan *pattern*.
2. Setelah itu, program akan mengeluarkan hasil dari iterasi dan mencetak waktu yang dibutuhkan.

Contoh Ilustrasi :
Teks : Wakil Presiden
Pattern : Presiden
Iterasi :

Wakil Presiden
Presiden
Presiden
Presiden
Presiden
Presiden
Prseiden
Presiden (matched)

Gambar 7 contoh program Algoritma Brute Force

3.1.2 Uji Kasus Knuth-Morris-Pratt (KMP)

Ketika pengguna memasukkan kombinasi kasus pertama untuk menggunakan metode dengan *matching keyword* dan memasukkan metode algoritma KMP, hal berikut yang akan terjadi.

1. KMP akan berjalan dari kiri ke kanan untuk pencocokkannya. Lalu pertama ia akan mengecek seperti biasa sampai bertemu kesalahan. Setelah itu, KMP akan mencocokkan prefix terpanjang pada pattern dengan suffix terpanjang pada teks. Jika ada, maka prefix dan suffix itu akan dijajarkan dan pengecekan dilakukan setelah jajaran prefix suffix tersebut. Jika tidak ada, maka pattern digeser sampai melewati error tersebut.
2. Setelah itu, program akan mengeluarkan hasil dari iterasi dan mencetak waktu yang dibutuhkan.

Contoh Ilustrasi :
Teks : a b a c a a b a c c a b a c a b a a
Pattern : a b a c a b
Iterasi :

a b a c a a b a c c a b a c a b a a
a b a c a b
a b a c a b
a b a c a b
a b a c a b (matched)

Gambar 8 contoh program Algoritma KMP

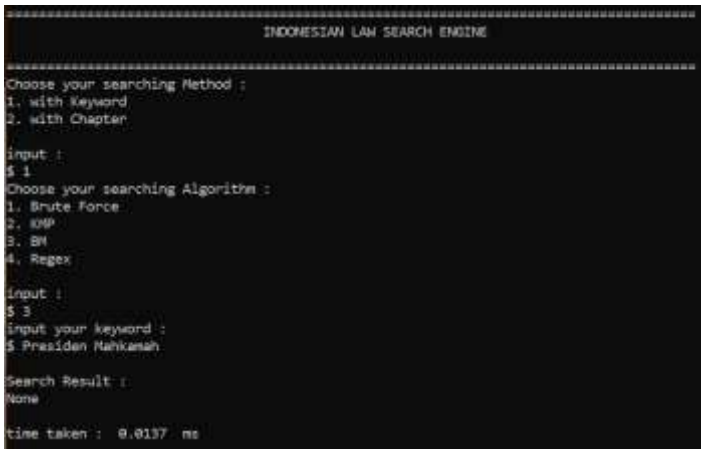
3.1.3 Uji Kasus Boyer-Moore (BM)

Ketika pengguna memasukkan kombinasi kasus pertama untuk menggunakan metode dengan *matching keyword* dan memasukkan metode algoritma BM, hal berikut yang akan terjadi.

1. BM akan berjalan dari kiri ke kanan untuk pergeseran *pattern* dan dari kanan ke kiri untuk pencocokkan *string*. Jika ada satu karakter yang tidak cocok, program akan melihat, apakah karakter itu berada pada sebelah kiri atau kanan atau tidak ada di pattern. Jika ada di sebelah kiri, maka *pattern* akan digeser sampai sejajar ke kanan. Jika di sebelah kanan, maka *pattern* akan digeser satu karakter ke kanan. Jika tidak ada, maka *pattern* digeser sampai melewati karakter tersebut.
2. Setelah itu, program akan mengeluarkan hasil dari iterasi dan mencetak waktu yang dibutuhkan.

Contoh Ilustrasi :
Teks : a b a c a a b a c c a b a c a b a a
Pattern : a b a c a b
Iterasi :

a b a c a a b a c c a b a c a b a a
a b a c a b
a b a c a b
a b a c a b
a b a c a b
a b a c a b
a b a c a b
a b a c a b
a b a c a b (matched)



Gambar 8 contoh program Algoritma BM

Program akan mencari dengan syarat *exact match* pada teks yang diberikan. Dimana jika tidak ada *String* yang cocok dalam teks tersebut, keluaran akan mencetak "None". Dalam kasus ini, BM tidak dapat menemukan *String* "Presiden Mahkamah" pada UUD 1945.

3.1.4 Uji Kasus Regular Expression (RegEx)

Berbeda dengan *exact matching*, Regular Expression menggunakan *heuristic matching* yang dapat mendeteksi *pattern* per kata. Oleh karena itu akan lebih mudah untuk mendapatkan hasil di bandingkan *exact matching*.

Ketika pengguna memasukkan kombinasi kasus pertama untuk menggunakan metode dengan *matching keyword* dan memasukkan metode Regex, hal berikut yang akan terjadi.

1. Program akan mengecek bukan satu kesatuan *string* tapi per kata dari *string* yang akan di pisah oleh spasi.
2. Jika semua kata di dalam *pattern* terdapat dalam teks, maka fungsi pencarian *regex* akan mengembalikan true. Jika tidak maka false.
3. Setelah itu program akan membuat sebuah list yang berisi pasal mana saja yang mana *regex* mengembalikan true dan mencetaknya.

Gambar 9 contoh program Regular Expression

Dapat kita lihat bahwa dengan menggunakan *regex*, input "Presiden Mahkamah" dapat mendapatkan solusi. Hal ini dikarenakan pencariannya dilakukan per kata.

IV. KESIMPULAN

Dari penilitian yang saya lakukan, saya mendapati bahwa fitur *searching* merupakan sesuatu hal yang sudah lazim dan wajib ada di setiap aplikasi. Karena dengan fitur tersebut memudahkan kita semua dalam mencari sesuatu yang mungkin sulit untuk dicari karena banyaknya data yang diberikan. Beberapa algoritma *string matching* yang cocok untuk hal tersebut adalah brute force, KMP, dan Boyer-Moore.

Dari uji kasus diatas, dapat kita lihat bahwa algoritma brute force jika dibandingkan dengan algortima string matching lainnya sangat berbeda kompleksitas dan kecepatannya, sedangkan KMP dan BM tidak bisa dipastikan karena bergantung pada teksnya juga. Lalu, jika kita bandingkan dengan menggunakan *regex*, dapat kita ketahui bahwa hasil solusi dengan menggunakan regex mendapatkan hasil yang lebih cepat dan akurat dibandingkan dengan algoritma *exact matching* lainnya. Tetapi kekurangan dari *regex* yang tidak dimiliki oleh algoritma *exact matching* adalah dengan menggunakan *regex*, kita tidak bisa menghitung berapa persen kemiripan dari *pattern* tersebut terhadap teks.

V. UCAPAN TERIMA KASIH

Pertama-tama penulis ingin mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena dengan rahmat dan karunia-Nya penulis dapat menyelesaikan makalah dengan judul "Penerapan dan Perbandingan Algoritma String Matching dalam Aplikasi UUD 1945 dan UU di Indonesia" ini dengan baik. Penulis juga berterima kasih kepada dosen yang memberikan tugas ini, Dr. Ir. Rinaldi Munir, M.T., dan kepada dosen pengajar, Dr. Masayu Leylia Khodra, S.T., M.T. selaku dosen IF2211 Strategi Algoritma Kelas 02, atas bimbingan beliau selama ini dalam mengajar dan memberikan ilmu dalam mata kuliah strategi algoritma sehingga penulis mampu membuat makalah ini. Selain itu, penulis juga berterima kasih

kepada rekan-rekan yang telah memberikan semangat dan dorongan kepada penulis sehingga makalah ini dapat diselesaikan.

REFERENCES

[1] Donald E. Knuth, James H. Morris and Vaughan R. Pratt, 1977 in: "Fast Pattern Matching in Strings." In SIAM Journal on Computing, 6(2): 323–350. 24 April 2019.

[2] Levitin, A. 2012. *Introduction to The Design and Analysis of Algorithm, 3rd Ed.* United States of America: Pearson Inc. 25 April 2019

[3] Undang-undang Dasar Negara Republik Indonesia 1945

[4] Slide Kuliah IF2211 Strategi Algoritma Teknik Informatika Insititut Teknoogi Bandung (ITB)

[5] https://www.w3schools.com/python/python_regex.asp. diakses pada 25 April 2019

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 26 April 2019

Christopher Billy Setiawan
13517050